Universitat Autònoma de Barcelona

Escola Tècnica Superior d'Enginyeria

Departament d'Arquitectura de Computadors

i Sistemes Operatius

Meta-Planificador Predictivo para Entornos Multicluster no Dedicados

Memoria presentada por **Josep Lluís Lérida Monsó** bajo la dirección de Dr. Porfidio Hernández Budé y de Dr. Francesc Solsona Tehàs para el grado de Doctor por la Universidad Autónoma de Barcelona.

Barcelona, 23 de junio de 2009

Meta-Planificador Predictivo para Entornos Multicluster no Dedicados

Memoria presentada por **Josep Lluís Lérida Monsó** para el grado de Doctor por la Universidad Autónoma de Barcelona. Trabajo realizado en el Departamento de Arquitectura de Computadores y Sistemas Operativos (DACSO) de la Escuela Técnia Superior de Ingeniería de la Universidad Autónoma de Barcelona, dentro del programa de Doctorado en Informática, opción A: "Arquitectura de Computadores y Procesamiento Paralelo", bajo la dirección del Dr. Porfidio Hernández Budé y Dr. Francesc Solsona Tehàs.

Barcelona, 23 de junio de 2009

Vo. Bo. Director:

Dr. Porfidio Hernández Budé

Vo. Bo. Director:

Dr. Francesc Solsona Tehàs

A mi padre

Agradecimientos

Cuando se toma la decisión de emprender un viaje como el desarrollo de una tesis doctoral, uno no es consciente del reto que esto supone y de las dificultades que se va a encontrar por el camino. Largas horas de trabajo, duras noches de café y interminables deadlines. No obstante, también existen elementos gratificantes, los buenos momentos de tertulia y trabajo en equipo con los miembros del grupo, la estrecha relación con los directores y compañeros de trayecto, viajes que aportan nuevas vivencias y el conocimiento de nuevas gentes y culturas. En definitiva, una aventura que proporciona un crecimiento personal y intelectual incomparable que te abre las puertas a nuevas etapas de la vida. Por este motivo quiero dedicar estas líneas a aquellas personas que me han acompañado durante este trayecto.

En primer lugar quiero agradecer a Porfidio Hernández y a Francesc Solsona su magistral dirección y constante dedicación a la supervisión del trabajo. Especialmente a Francesc por guiarme y ayudarme a superar los obstáculos, por su constancia y especial interés en que todo saliera bien.

A Emilio Luque y a todo el grupo de investigación del departamento de Arquitectura de Computadores y Sistemas Operativos de la Universidad Autónoma de Barcelona, por su apoyo e interés en este trabajo. A mis compañeros de doctorado en la UAB con los que he compartido viajes y muy buenos momentos.

A mis compañeros del área de Arquitectura de Computadores de la Universidad de Lleida, Fernando Guirado, Fernando Cores, Francesc Giné y Concepció Roig, por el apoyo mostrado en todo momento, por sus acertados consejos y los buenos momentos compartidos. Al resto de compañeros del Grup de Computació distribuïda, especialmente a Ivan Teixidó por su comprensión y colaboración durante el desarrollo de este trabajo. A Josep Conde por atender tan pacientemente todas mis consultas, por su colaboración y por sus lecciones magistrales de estadística.

A todos mis amigos, con los que tan agradables momentos he podido compartir en mi tiempo libre.

Y muy especialmente a mi familia, que aún sin comprender porqué tantas horas de trabajo, me siguen animando y apoyando en todo momento. A mi padre Baldomero que siempre lo dio todo por nosotros y que donde esté sé que estará orgulloso de todo esto. A mi madre Carmen y a mi hermano Oscar que me han dado la fuerza necesaria para llegar hasta aquí y que siempre están ahí. A Sonia por el empujón vital que me ha ofrecido en la recta final de este proyecto.

Prólogo

A principios de los años 90 muchas de las características y capacidades proporcionadas únicamente por sofisticados y costosos Supercomputadores se lograron superar mediante el uso adecuado de conjuntos de ordenadores de sobremesa agrupados en sistemas denominados Cluster.

En la actualidad, el continuo aumento de las capacidades de cómputo y la disminución del coste de adquisición de los ordenadores de sobremesa ha provocado un uso extendido de los clusters en centros de investigación, instituciones, organizaciones, etc. No obstante, el crecimiento constante de los requerimientos de las aplicaciones científicas hacen necesaria la búsqueda de sistemas aún más potentes y/o con mayor número de recursos.

En la última década, la posibilidad de unir los recursos (clusters) de una misma organización para obtener mayor capacidad de cómputo ha despertado un gran interés. Esto ha propiciado la aparición de nuevos sistemas Multicluster que incorporan sofisticadas técnicas de planificación de trabajos y gestión de recursos orientadas a aprovechar al máximo los recursos de cómputo distribuidos en varios clusters.

Aunque el bajo coste de los recursos de cómputo facilita el crecimiento del número de clusters y recursos por cluster, este crecimiento tiene sus limites. En primer lugar debemos considerar los problemas de espacio que no siempre son fáciles de conseguir y en segundo lugar el elevado coste del consumo energético producido por las unidades de cómputo y por los sistema de refrigeración que se necesitan.

Bajo este panorama, un modo de proporcionar mayor número de recursos sin aumentar los costes es incorporar al Multicluster recursos de cómputo ociosos de los usuarios de una organización para la ejecución de aplicaciones paralelas, como por ejemplo los ordenadores del laboratorio de una universidad.

La tarea de meta-planificación en un entorno Multicluster es un tarea compleja y una línea de trabajo en pleno auge en la actualidad. En primer lugar, la cantidad de recursos que se deben gestionar es muy grande y los recursos pueden ser heterogéneos. En segundo lugar, deseamos considerar la planificación en un sistema no dedicado donde los recursos se comparten con otros usuarios, aumentando la dificultad de la meta-planificación.

La tarea de meta-planificación esta formada por un conjunto de subtareas importantes que definen el propósito de nuestro sistema. Algunas de estas tareas son la selección de recursos, la selección de trabajos a ejecutar, la gestión de las colas en cada cluster, la gestión de la co-asignación, etc. Proponer una solución a la meta-planificación significa proponer soluciones en las distintas fases de la tarea de meta-planificación. El objetivo de la meta-planificación en el presente trabajo es obtener el máximo rendimiento de las aplicaciones paralelas sin perjudicar al usuario local.

En primer lugar se ha propuesto una arquitectura jerárquica de planificación, denominada M-CISNE. Se ha analizado el rendimiento bajo distintas condiciones de carga y configuración del Multicluster y se compara con una arquitectura con una única cola global. Los resultados obtenidos demuestran que la jerarquía de colas permite tomar decisiones de meta-planificación más inteligentes y aumentar el rendimiento de las aplicaciones paralelas con respecto a un sistema totalmente centralizado.

En el contexto de esta nueva arquitectura, se han propuestos distintas técnicas de *selección de clusters* que permiten seleccionar el *cluster* más adecuado para ejecutar una aplicación paralela basándose tanto en la estimación del tiempo de *turnaround* como en la ocupación de los recursos por el usuario local. Se realiza una análisis del comportamiento de esta técnica bajo distintas configuraciones de carga y configuración del Multicluster. Además se compara nuestra propuesta con otras técnicas de la literatura.

El objetivo de las políticas de *selección de clusters* propuestas, es realizar asignaciones que permitan obtener el máximo rendimiento de las aplicaciones paralelas sin perjudicar al usuario local. Estas políticas se basan en la estimación del tiempo de *turnaround* en cada uno de los clusters. El rendimiento de estas políticas radica en la obtención de estimaciones lo más precisas posibles. La heterogeneidad y la naturaleza no dedicada de los recursos hacen aún más difícil la tarea de estimación.

En el presente trabajo se han propuesto varias técnicas de estimación, un método estadístico basado en el *aprendizaje de casos* y un método de simulación guiado por un *modelo analítico* del tiempo de ejecución de una aplicación paralela que considera la heterogeneidad y la ocupación de los recursos tanto de cómputo como de comunicación. Se ha analizado el comportamiento de ambas técnicas de estimación bajo distintas condiciones de carga y se han comparado con las técnicas más habituales de la literatura.

Una arquitectura jerárquica de planificación dispone de un sistema jerárquico de colas que se debe gestionar adecuadamente para cumplir con los objetivos establecidos. En primer lugar, se ha realizado un estudio para determinar si debía existir o no un límite máximo de trabajos en las colas locales y cual era su efecto sobre el rendimiento de las aplicaciones paralelas y las desviaciones en las estimaciones llevadas a cabo por el motor de predicción de cada cluster. En segundo lugar, se ha propuesto una nueva técnica de selección de trabajos, con el fin de eliminar los efectos negativos del bloqueo de trabajos en la cola del meta-planificador. Se ha realizado un estudio de los resultados de rendimiento de las aplicaciones paralelas y desviación en las predicciones comparando nuestra técnica con las más habituales en la literatura.

Con el fin de poder compartir los recursos libres de distintos clusters para la ejecución de aplicaciones paralelas, se ha desarrollado una técnica de *asignación de recursos entre clusters* basada en un modelo de programación entera binaria (*Mixed*

Integer Programming). El modelo realiza la asignación de una aplicación paralela a un conjunto de recursos de distintos clusters con dos objetivos fundamentales, la obtención del mejor rendimiento de la aplicación paralela y la no saturación de los canales de comunicación.

Mediante el uso de esta técnica y con el fin de poder sacar el máximo provecho de los recursos libres, en el presente trabajo proponemos un refinamiento de la tarea de meta-planificación. La técnica consiste principalmente en tratar de co-asignar aquellos trabajos que se encuentran en la cola de algún cluster a la espera de recursos libres. El objetivo principal es obtener mayores rendimientos de las aplicaciones paralelas y un mayor aprovechamiento de los recursos del Multicluster, reduciendo los tiempos de espera y aprovechando los recursos libres entre clusters.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

Históricamente, los avances en la computación científica han ocasionado un incremento de las necesidades de cómputo. A medida que las prestaciones de los sistemas computacionales han ido creciendo, científicos e ingenieros han trabajado en nuevas tecnologías que les permitieran sacar el máximo rendimiento de los recursos computacionales. Por otro lado, en esta carrera por aprovechar al máximo los recursos y aumentar así las prestaciones de las aplicaciones que se ejecutan, se utilizan modelos de aplicación cada vez más sofisticados y complejos que requieren aumentar también el número y capacidad de los recursos.

A lo largo de la historia, se han aplicado varias alternativas para poder abordar este problema. La primera ha sido incrementar la potencia de cálculo de los ordenadores. Esta opción es tan solo viable hasta cierto punto, más allá del cual no es posible realizar mejoras debido a limitaciones físicas de los propios materiales con que se construyen los ordenadores.

Otra alternativa al aumento de la capacidad computacional ha sido la paralelización de los cálculos necesarios para resolver un problema y el aumento de las unidades de cómputo. En este caso, el aumento de la potencia de cálculo se encuentra limitado por el coste del equipamiento o por la estructura de las propias aplicaciones y no por limitaciones físicas.

En la figura 1.1 se observa la taxonomía de los sistemas paralelos presentada por Tanenbaum en [Tan05]. En el primer nivel de esta taxonomía encontramos la clasificación de Flynn de las arquitecturas de computadores, donde se clasifican los sistemas según el modo que tienen de procesar las instrucciones y los datos. Según esta clasificación, en una arquitectura SIMD (Single Instruction Multiple Data) se explota el paralelismo natural de los datos en aplicaciones que no dejan de ser secuenciales. En cambio, en una arquitectura MIMD (Multiple Instructions Multiple

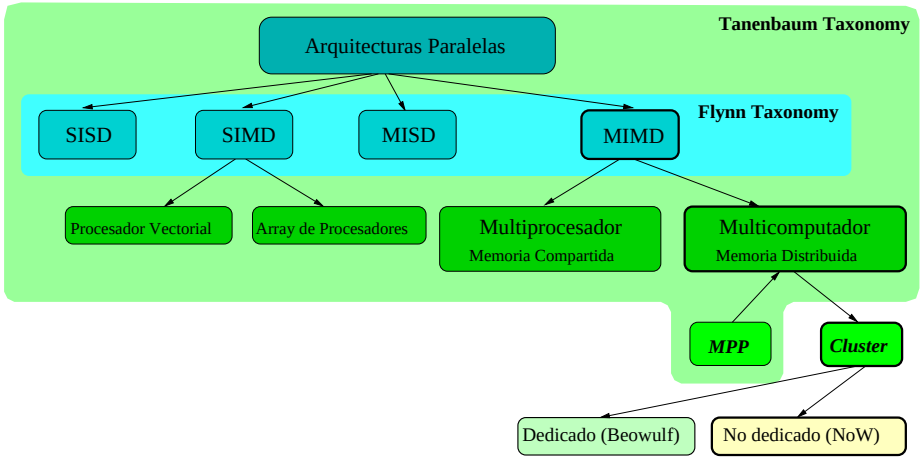


Figura 1.1: Taxonomía de los sistemas paralelos

Data), varios procesadores autónomos ejecutan simultáneamente instrucciones diferentes sobre datos diferentes. En la actualidad, el 99.6 % de sistemas en la lista de supercomputadores más potentes del mercado [TOP] utilizan una arquitectura MIMD.

Las máquinas MIMD pueden ser a su vez Multiprocesador (memoria compartida) o Multicomputador (memoria distribuida). En un sistema Multiprocesador todos los procesadores comparten la memoria y el sistema operativo es el encargado de mantener la coherencia. Aunque las aplicaciones son más sencillas de programar, este tipo de sistemas ofrecen una escalabilidad limitada y poca flexibilidad. En un sistema Multicomputador, en cambio, cada procesador dispone de su propia memoria sin tener acceso directo a la memoria de otros procesadores, compartiendo la información mediante mecanismos de paso de mensajes. Este tipo de sistemas ofrecen mayor flexibilidad y escalabilidad que los anteriores, convirtiéndose en los sistemas de cómputo paralelo de altas prestaciones más populares.

Los primeros sistemas de cómputo de altas prestaciones que aparecieron fueron los *MPP* (Massively Parallel Processors). Sistemas formados por un gran número de procesadores con un alto nivel de acoplamiento que les confiere una gran capacidad de cómputo y una alta velocidad de comunicación. El problema principal de estos supercomputadores es su elevado coste.

Con el tiempo, muchas de las características y capacidades proporcionadas únicamente por sofisticados y costosos supercomputadores se han logrado superar mediante el uso adecuado de conjuntos de ordenadores de sobremesa agrupados en sistemas denominados *Cluster*.

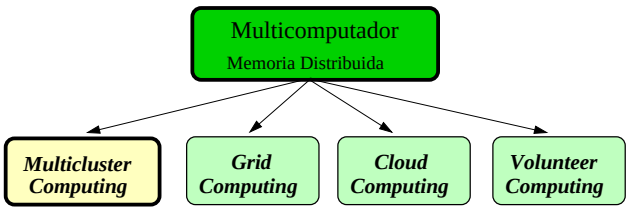


Figura 1.2: Taxonomía actual del Multicomputador

Un **Cluster** *es una colección de recursos de computación formado normalmente por ordenadores comunes de sobremesa conectados entre sí y que se comportan como una única computadora, alcanzando rendimientos comparables al de los Supercomputadores.*

Los sistemas *Cluster* se clasifican en *dedicados* o *no dedicados*. En un cluster dedicado los recursos se destinan exclusivamente a realizar tareas (aplicaciones) del cluster. Un cluster no dedicado o *NOW* (***N**etwork **O**f **W**orkstations*) hace uso de los ciclos de reloj que el usuario del computador no está utilizando para realizar sus tareas.

La posibilidad de poder alcanzar altas prestaciones de cómputo mediante el uso de recursos de bajo coste y los grandes avances en las redes de comunicaciones provocaron a principios de los 90 un gran interés por el estudio y el uso de los sistemas *Cluster* [LLM88, SSB+95, RBMS97, ELvD+96].

En la actualidad, la tecnología *multicore* ofrece la posibilidad de cuadriplicar las unidades de procesamiento de un *Cluster* sin un elevado coste adicional. La continua mejora de la relación coste-rendimiento y el gran esfuerzo dedicado al desarrollo de nuevas estrategias para el uso eficiente de los recursos y el aumento de las prestaciones, los sistemas *Cluster* han seguido creciendo en número y tamaño, convirtiéndose en una alternativa a los supercomputadores tanto en entornos industriales como académicos. Buena prueba de ello es que a fecha de hoy, el 17,6 % de supercomputadores de la lista de los TOP500 [TOP] está formada por *MPPs* y el 82 % está formado por entornos *Cluster*.

El uso extendido de los clusters en centros de investigación e industrias junto con su bajo coste ha provocado que cada vez sea más frecuente encontrar varios de ellos en una misma organización. La posibilidad de unir los recursos de una misma organización o de distintas organizaciones para obtener mayor capacidad de cómputo ha despertado un gran interés en la última década [IEEb, DA06]. Esto ha propiciando la aparición de nuevos modelos de cómputo distribuido tales como *MultiCluster Computing* [IEEa, BB01], *Grid Computing* [IEEb, Sto07], *Cloud Computing* [VRMCL09], *Volunteer Computing* [Vol, PCS07] que incorporan sofisticadas téc-

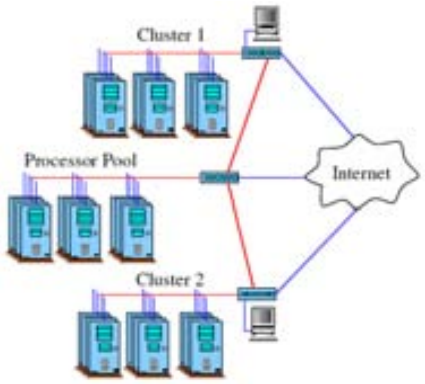


Figura 1.3: MultiCluster

nicas de planificación de trabajos y gestión de recursos orientadas a aprovechar al máximo los recursos de cómputo distribuidos en varios clusters.

La aparición de estos nuevos modelos conlleva un replanteamiento de la taxonomía de los sistemas multicomputador, que ahora pasa a estar formada por sistemas capaces de gestionar recursos (*MPPs* y/o *Clusters*) de una o varias organizaciones tal como se muestra en la Figura 1.2. Aunque todos estos sistemas son capaces de gestionar recursos de múltiples clusters, la diferencia principal entre el *Multicluster* y el resto de sistemas es el ámbito de las comunicaciones.

Un **Multicluster** *lo podemos definir como la unión de un conjunto de Clusters mediante una red dedicada de comunicaciones dentro de una misma organización, formando un recurso computacional aún mayor.*

A simple vista puede parecer que un *Multicluster* se diferencia del resto de sistemas únicamente por el alcance de las comunicaciones. Mientras que el Multicluster se encuentra limitado al ámbito de una organización, institución, empresa o un campus universitario, el resto de sistemas suelen tener nodos localizados en ámbitos a nivel nacional e internacional.

Si realizamos un análisis más exhaustivo, podemos observar que un *Multicluster* tiene una característica arquitectónica distintiva: las redes de comunicaciones entre clusters se conectan mediante enlaces dedicados. Esto tiene varias implicaciones importantes. En primer lugar, un *Multicluster* dispone de un ancho de banda predecible y fiable entre recursos de distintos clusters, contrariamente a lo que sucede con el resto de sistemas, los cuales se conectan a través de Internet. En segundo lugar, en un entorno *Multicluster* la disponibilidad de los recursos es predecible y controlable, mientras que el resto de sistemas son altamente dinámicos y mutables de modo que no podemos garantizar su disponibilidad en el tiempo.

La posibilidad de disponer de un mayor número de recursos para la ejecución de aplicaciones con mayores requerimientos y las ventajas que se pueden obtener del uso compartido de los mismos hace de la planificación en entornos *Multicluster* un área de estudio emergente.

Aún así, el aumento en la disponibilidad de recursos no garantiza una buena utilización de los mismos. Krueger demostró en [KC91] que en promedio, el 91 % de los recursos en un sistema NOW están infrautilizados y Arpaci [ADV+95] que entre un 60 % y un 70 % de los recursos se encontraban infrautilizados incluso en las horas de mayor demanda. Esto es debido a que, tal como se demuestra en [RH00], el perfil mayoritario entre los usuarios de entornos como oficinas, laboratorios docentes, etc., es el de personas que utilizan su ordenador como herramienta ofimática, editando documentos, leyendo el correo, navegando por internet o utilizando una aplicación administrativa o de contabilidad. Esto provoca que los recursos estén libres la mayor parte del tiempo y que sean utilizados totalmente en ocasiones excepcionales.

Por otro lado, en estudios como [GLD04] y [DMS05] se ha analizado el confort del usuario con la capacidad de interacción de su ordenador, ante el uso de los recursos por parte de aplicaciones externas. Ambos estudios concluyen que un sistema que cede los recursos para el uso de aplicaciones externas, puede llegar a recibir grandes cantidades de carga externa sin comprometer la interactividad del usuario. Además, en la actualidad la tecnología *multicore* cuadriplica la cantidad de unidades de procesamiento de modo que aumenta aún más la capacidad de cómputo y se hace más notable es desaprovechamiento de los recursos. Esto nos lleva a pensar que no sólo es posible sino razonable aprovechar la gran cantidad de recursos disponibles en un entorno *Multicluster* para llevar a cabo tareas de cómputo paralelo.

El presente trabajo se centra en el estudio de nuevas técnicas que nos permitan obtener el máximo rendimiento de las aplicaciones paralelas en entornos *Multicluster* no dedicados. Esto implica minimizar el tiempo de ejecución de las aplicaciones paralelas sin reducir la interactividad de las tareas locales.

Existen multitud de formas de organizar un entorno *Multicluster*, según la naturaleza de los recursos, el modo de compartición de los recursos, los objetivos de rendimiento, etc. Las condiciones de partida guían el rumbo del trabajo de investigación y determinan las conclusiones finales.

En el presente trabajo estamos interesados en entornos *Multiclusters* formados en el contexto de una organización, institución, empresa o un campus universitario, con recursos procedentes de aulas de docencia y/o investigación, así como áreas o secciones de una empresa o institución, tales como gerencia, contabilidad, producción, recursos humanos, etc. En general, para cada cluster, asumiremos las siguientes condiciones de partida que consideraremos a lo largo de todo el trabajo:

- Los recursos pueden ser heterogéneos tanto de cómputo como de comunicación.
- Los recursos pueden estar compartidos por usuarios locales con distintos requerimientos de procesamiento y comunicaciones.
- Los clusters permiten la ejecución concurrente de varias aplicaciones paralelas.
- Los clusters se pueden gestionar de forma independiente, con distintas políticas de planificación y objetivos de rendimiento.

En este capítulo se presenta una visión global del paradigma *Multicluster*, presentando las consideraciones a tener en cuenta tanto desde el punto de vista de su arquitectura como de las técnicas de planificación y su estado del arte. Por último, se darán a conocer los objetivos que se desarrollan en este trabajo.

1.1. Planificación en Clusters no dedicados

A principios de los 90, surgieron gran numero de sistemas con la idea de aprovechar los recursos infrautilizados en entornos no dedicados para el cómputo intensivo de aplicaciones paralelas como Condor [LLM88, PL95, ELvD+96, PL96] que fue la base tecnológica para el IBM LoadLeveler, PBS [Ope], Piranha [GK92, CFGK95], Utopia [ZZWD93] comercializado como Load Sharing Facility (LSF) [LSF], el Stealth scheduler [KC91], Amber [FBCL91], etc.

El principal objetivo de estos sistemas era mejorar la utilización de los recursos sin comprometer la interactividad de los usuarios locales. Para poder cumplir con este objetivo se desarrollaron técnicas muy diversas como la migración [LLM88], la reserva de recursos [SCJG00] o la planificación de trabajos basada en modelos de compartición de recursos [Fei97]. La Figura 1.4 muestra una taxonomía de éstas técnicas, que pueden aplicarse tanto individual como conjuntamente y pueden ser implementadas a nivel de sistema operativo o a nivel de aplicación.

En esta sección hacemos un repaso de estas técnicas para identificar las que nos pueden ser útiles en los entornos Multicluster no dedicados, centro de estudio del presente trabajo.

La **migración** [RL04, BSA05] consiste en aprovechar los recursos de cómputo cuando el usuario local está inactivo, utilizando técnicas de expropiación y migración de las aplicaciones paralelas cuando el usuario local reanuda su actividad. El

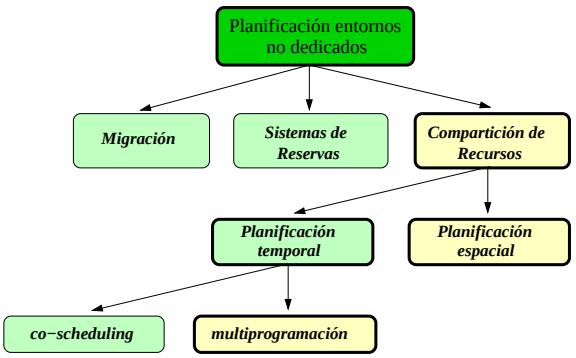


Figura 1.4: Taxonomía de las técnicas de planificación en entornos no dedicados

principal inconveniente del proceso de migración es su elevado coste que, en un entorno Multicluster, puede verse agravado si se migran procesos entre distintos clusters. La migración no es siempre necesaria para aprovechar los ciclos de cómputo libres como se demuestra en [AS97, GSB+03, HGH+04].

Algunos sistemas comerciales [BAG00, Xu01, Moa, JSC01, US04] utilizan **sistemas de reservas** sobre los recursos. A la llegada de una aplicación paralela, mediante el uso de técnicas predictivas, el planificador busca intervalos con recursos disponibles y los reserva para la ejecución de la nueva aplicación. El inconveniente de esta técnica es que las aplicaciones locales y paralelas utilizan de forma exclusiva los recursos, de modo que siguen sin aprovecharse los ciclos de cómputo disponibles. Por otro lado, como demuestra Snell en [SCJG00], la reserva de recursos provoca fragmentación temporal y espacial, ya que existen recursos que no pueden ser aprovechados porque disponen de reservas activas.

Aunque las técnicas de *migración* y *sistemas de reservas* se pueden aplicar en entornos heterogéneos, las aplicaciones se ejecutan en recursos dedicados exclusivamente para ellas. Estas técnicas no explotan la compartición de los recursos entre las tareas locales y paralelas con el fin de aumentar la utilización de los recursos y maximizar el rendimiento de las aplicaciones paralelas.

Un conjunto de técnicas que permiten explotar esta relación son las técnicas de planificación basadas en modelos de compartición de recursos, las cuales se dividen en técnicas de ***planificación temporal*** y ***planificación espacial***. Estas técnicas deciden cuando y dónde deben ejecutarse las tareas de una aplicación paralela respectivamente. Aunque estas técnicas han sido ampliamente utilizadas en sistemas cluster necesitamos analizar de que modo se pueden aplicar a un entorno formado por la unión de varios clusters no dedicados.

Planificación temporal

La **planificación temporal** (*time-sharing*) es la técnica mediante la cual varias aplicaciones comparten un mismo recurso de cómputo alternando su ejecución en el tiempo. Esto significa usar un grado de paralelismo de aplicaciones paralelas (*Multiprogrammed Parallel Level*, MPL) mayor que 1 (MPL>1) para garantizar que los ciclos de cómputo que una aplicación no utiliza mientras está realizando operaciones de entrada/salida sean aprovechados por otras aplicaciones.

Distintos autores han aplicado esta técnica de formas muy diversas. Ousterhout presentó en [Ous82] una técnica que ha sido ampliamente usada en entornos cluster [Fei96, GSHL03, HGH$^+$04], la *co-planificación*, mediante la cual todas las tareas de una misma aplicación paralela son ejecutadas de forma simultánea en todo el sistema. Esto permite poder ejecutar aplicaciones paralelas en sistemas no dedicados como si se tratara de un sistema dedicado, reduciendo el tiempo de espera por eventos de comunicación y en consecuencia el tiempo de ejecución de la aplicación paralela.

El inconveniente de la *co-planificación* es la dificultad de su implementación en un entorno Multicluster. Las tareas paralelas de una misma aplicación requieren sincronización y éstas pueden estar repartidas entre distintos clusters. Las elevadas latencias en los canales de comunicación entre clusters y la necesidad de una coordinación global entre clusters independientes requiere de una implementación muy compleja que puede no verse compensada por los rendimientos obtenidos en este tipo de entornos.

Otro modo de utilizar la planificación temporal, sin complejas implementaciones, es permitir la ejecución concurrente (MPL>1) de distintas tareas en un mismo recurso de cómputo pero sin necesidad de sincronización con el resto de tareas de la misma aplicación paralela.

Los buenos resultados obtenidos en el aprovechamiento de los recursos y el rendimiento de las aplicaciones con el aumento del grado de paralelismo (MPL>1) en entornos paralelos distribuidos [GSB$^+$03, HGH$^+$04, HGH$^+$05a], hace pensar que es posible y deseable ejecutar aplicaciones paralelas de forma concurrente en entornos Multicluster no dedicados con el fin de obtener buenos resultados de rendimiento sin perjudicar a los usuarios locales.

Arpaci en [ADV$^+$95] demuestra que es aconsejable limitar el grado de paralelismo de las aplicaciones paralelas. En estudios anteriores dentro de nuestro grupo de investigación [GSHL03, HGH$^+$04], se demuestra que el límite en el grado de paralelismo tiene que ver con la capacidad de la memoria de los recursos y con su capacidad de cómputo. Abawajy presenta en [AD03] un método para determinar el máximo grado de paralelismo de cada nodo en función de su capacidad de cómputo.

Planificación espacial

Mientras la *planificación temporal* se centra en cuando se ejecuta una determinada aplicación, la **planificación espacial** (*space-sharing*) se encarga de decidir qué recursos se deben usar para ejecutar una determinada aplicación. En general, la *planificación espacial* se divide en dos fases, el *particionamiento* y *la selección de recursos* dentro de una determinada partición:

1. El ***particionamiento*** consiste en dividir el conjunto de recursos disponibles en subconjuntos donde poder asignar una o varias aplicaciones. El tamaño de estas divisiones puede venir determinado por la arquitectura del sistema, por las características de la aplicación o bien por la capacidad del software de gestión de los recursos, y en algunos casos el tamaño puede variar en el tiempo. Básicamente, podemos distinguir cuatro tipos de particionamiento:

   - Estático: el tamaño es definido por el administrador. Se suele utilizar ante restricciones de acceso a determinados recursos o ante políticas que pretenden realizar una clasificación de trabajos o usuarios. Algunos sistemas comerciales realizan particiones para procesos por lotes y para procesos interactivos.
   - Variable: el tamaño se determina a la llegada de la aplicación basándose en los requerimientos de la misma. En ocasiones, el rango de posibles tamaños está predeterminado por restricciones en la arquitectura. Por ejemplo, en un hipercubo las aplicaciones no se pueden ejecutar en un conjunto arbitrario de recursos y se particiona el hipercubo en subcubos de distintos tamaños [Fei97, Fei96]. En otras ocasiones las particiones pueden ser totalmente arbitrarias.
   - Adaptable: el tamaño se determina en el momento en que la aplicación se va a ejecutar, atendiendo a los requerimientos de la aplicación y al estado de carga de los recursos, y no varia durante su ejecución. Este tipo de particionamiento solo se puede llevar a cabo con aplicaciones moldeables, es decir que aceptan que la cantidad de recursos venga determinada por el planificador en el momento de la ejecución.
   - Dinámico: el tamaño de la partición puede variar durante la ejecución de la aplicación atendiendo a los cambios en los requerimientos de la aplicación o de la carga del sistema. Este particionamiento requiere que las aplicaciones sean maleables, es decir, que acepten que los recursos puedan variar en tiempo de ejecución. Estos sistemas utilizan técnicas de expropiación y migración para poder reasignar los recursos.

2. La ***selección de recursos*** consiste en seleccionar el conjunto de recursos más adecuado para llevar a cabo la ejecución de una aplicación paralela, considerando el conjunto de recursos disponibles y su particionamiento. Para llevar a cabo esta selección podemos encontrar diversas técnicas en la literatura. Entre las más comunes encontramos los esquemas de tipo *buddy* y la utilización de modelos de rendimiento.

   - Los esquemas tipo ***buddy*** [LLWN94, SKSJ02] mantienen un registro de los recursos libres. Para poder determinar si un recurso está libre se suelen utilizar distintos criterios como la ocupación de Memoria [SSN99, AS99], la cantidad de aplicaciones asignadas a la misma partición [SHb00, HBLO+03], etc. En un entorno no dedicado, debemos tener en cuenta criterios adicionales que consideren el uso de los recursos por parte del usuario local.

     En [ADV+95] se establece mediante contrato social la cantidad de recursos destinados a las aplicaciones paralelas y se seleccionan los recursos en función de su capacidad disponible.

     En estudios anteriores, dentro del grupo de investigación, propusimos el uso de criterios como la cantidad de usuarios locales en una partición [LSG+06] o la cantidad de recursos disponibles para la ejecución de aplicaciones paralelas [HGH+05b] mediante el uso de un contrato social.

   - Los ***modelos de rendimiento*** se basan en modelar el comportamiento de las aplicaciones y/o del sistema [EHS+02, TRA+05]. El uso de estos modelos permiten establecer criterios de selección basados en la duración de la carga (*LRW - Least Remaining Work*), en la duración del trabajo paralelo (*MCT - Minimum Completion Time*), etc. *LRW*, por ejemplo, selecciona los nodos que se espera que completen la carga previamente asignada lo antes posible, en cambio *MCT* selecciona los nodos que se espera ejecuten la aplicación paralela en el menor tiempo.

La tendencia hacia un particionamiento dinámico responde a la necesidad de eliminar la fragmentación interna producida por las técnica de particionamiento estático. Por otro lado, esto supone pérdida de localidad y de independencia entre particiones, y una mayor complejidad en el modelo de programación, por lo que se suele utilizar en sistemas de menor escala y memoria compartida [Fei97].

Anastasiadis muestra en [AS97] la complejidad de aplicar un esquema dinámico a un entorno Cluster. Abawajy [AD03] y He [HJSN04] proponen dividir en particiones estáticas la gran cantidad de recursos de un entorno Multicluster para posteriormente aplicar métodos más flexibles dentro de cada partición.

En el presente trabajo, tomaremos como punto de partida un particionamiento estático de los recursos donde cada cluster forma una partición. En la sección 1.2.2 analizaremos distintas técnicas de selección de recursos a nivel de Multicluster y en la sección 1.2.3 estudiaremos el uso de técnicas de planificación que permiten la compartición de determinados recursos entre distintas particiones.

En cuanto a la selección de recursos a nivel de cluster, utilizaremos técnicas ampliamente usadas en estudios previos dentro de nuestro grupo de investigación basadas en el uso combinado de técnicas tipo *buddy* y modelos de rendimiento [HGH+04, HGH+05b].

1.2. Paradigma Multicluster

En las dos últimas décadas son muchos los trabajos de investigación centrados en la optimización del proceso de planificación en entornos cluster. En un entorno Multicluster formado por la unión de varios clusters de una misma organización, el número de recursos crece sustancialmente. Estos recursos pueden ser de naturaleza muy distinta y/o estar compartidos con usuarios locales de cada uno de los clusters.

Esta nueva configuración ofrece la posibilidad de aumentar la capacidad de cómputo y en consecuencia la ejecución de aplicaciones más complejas y con mayores requerimientos. Sin embargo, la gestión eficiente de los recursos de cómputo y de comunicación aumenta enormemente. Esto ha provocado que la tarea de planificación en entornos Multicluster sea un campo de estudio emergente en la actualidad.

Un componente verdaderamente importante y fundamental para la gestión de los recursos y de las aplicaciones paralelas es el *Planificador*. El objetivo principal del *Planificador* es la gestión eficiente de los recursos y de los trabajos cumpliendo con los objetivos marcados por usuarios y administradores del sistema. Para realizar aportaciones que permitan mejorar el rendimiento de la tarea de planificación es necesario conocer los elementos que intervienen en el proceso de planificación y analizar aquellos factores que puedan afectar a su rendimiento.

En la sección 1.2.1, analizamos las distintas posibilidades de organización de un planificador en un entorno Multicluster y tratamos de evaluar la mejor opción. En la sección 1.2.2, describimos las principales tareas del proceso de planificación en un entorno Multicluster, analizamos su problemática y identificamos aquellos aspectos que tienen mayor incidencia en la eficacia y el rendimiento de la planificación. En las secciones 1.2.3 y 1.2.4 se presentan las bases de las técnicas de *co-asignación* y *predicción* respectivamente, varias de las técnicas más comunes en la planificación en entornos Multicluster y centro de nuestro interés en el presente estudio.

1.2.1. Organización del planificador

La organización del planificador en un sistema Multicluster se puede dividir en tres categorías: ***centralizado***, ***descentralizado*** y ***jerárquico***. Un Multicluster con un esquema ***centralizado*** (Figura 1.5) dispone de un único planificador que administra de forma centralizada todos los recursos y trabajos del sistema.

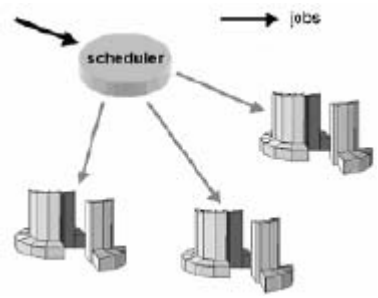


Figura 1.5: Esquema Centralizado

En un esquema *centralizado* el planificador dispone de un conocimiento global de todo el sistema y eso le permite una gestión muy eficiente de los recursos y de las aplicaciones. El inconveniente principal es que cuanto mayor es el sistema, mayor es la probabilidad de que se sature y por tanto es menos escalable y menos tolerante a fallos.

Un Multicluster con un esquema ***descentralizado*** (Figura 1.6) dispone de más de un planificador repartido por el sistema, donde cada uno se encarga de la gestión de un subconjunto de recursos y interactúa con el resto de planificadores intercambiando información y trabajos.

La principal ventaja de un sistema descentralizado es su escalabilidad, evitando la saturación producida por una gestión centralizada de toda la información, soportando además una mejor tolerancia a fallos. El principal inconveniente es la complejidad de su implementación, ya que los planificadores necesitan comunicarse entre sí para intercambiar información sobre su estado, especialmente cuando se desea compartir recursos de distintos clusters para la ejecución de una aplicación paralela.

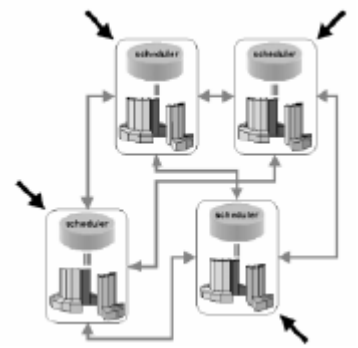


Figura 1.6: Esquema Descentralizado

Una solución intermedia que permite resolver algunos inconvenientes del sistema centralizado sin llegar a la complejidad de un sistema descentralizado son los sistemas ***jerárquicos***. En un sistema Jerárquico (Figura 1.7) la planificación y la gestión de los recursos se reparte en distintos niveles. En el nivel mas alto se encuentra un planificador global encargado de gestionar un conjunto de planificadores de nivel inferior. Los planificadores de nivel inferior controlan a su vez otros planificadores de niveles aún más inferiores, y así sucesivamente hasta llegar a los recursos finales.

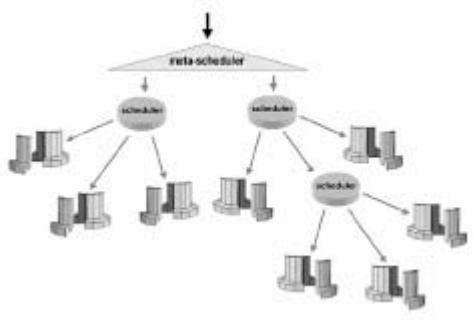


Figura 1.7: Esquema Jerárquico

Mediante una *organización jerárquica*, el planificador de nivel superior (*Meta-Planificador*) ve los planificadores de nivel inferior como un conjunto de recursos a gestionar pero no interacciona directamente con sus recursos finales. De esta forma, la información que debe gestionar cada planificador es mucho menor, quedando distribuida a lo largo de toda la jerarquía. Esto favorece la escalabilidad del sistema y la posibilidad de aplicar técnicas de tolerancia a fallos.

Una ***organización centralizada*** es la más adecuada cuando todos los recursos se encuentran bajo un mismo dominio administrativo, donde la heterogeneidad y la

cantidad de los recursos es controlable evitando problemas de latencia en las comunicaciones. Una ***organización* descentralizada** es la más adecuada cuando disponemos de un conjunto de clusters distribuidos geográficamente conectados mediante una red de comunicaciones de ámbito global, donde las latencias pueden llegar a ser significativas. Una ***organización jerárquica*** es una solución intermedia con las ventajas de las organizaciones anteriores. Esta estructura resulta muy útil en el ámbito de una organización con recursos separados en varios dominios administrativos, y donde el número de recursos y su heterogeneidad puede llegar a ser considerable.

En el presente trabajo estamos interesados en aprovechar los recursos de cómputo ociosos, en un entorno académico o de investigación, compuesto por un conjunto de clusters agrupados en distintos dominios administrativos y gestionados por sus propios sistemas de planificación, unidos mediante una red de interconexión de ámbito local. Para llevar a cabo la tarea de meta-planificación sobre el conjunto total de los recursos se requiere de un nuevo nivel de planificación. Esto sugiere la utilización de una estructura jerárquica con dos niveles de planificación.

En el primer nivel disponemos de un planificador global, también llamado ***Meta-Planificador***, encargado de recoger información sobre el estado de los recursos y decidir el conjunto de recursos idóneo para lanzar una determinada aplicación paralela. En un segundo nivel disponemos de planificadores locales en cada cluster. Cada planificador local gestiona sus propios recursos y dispone de sus propias políticas de planificación.

En la sección 1.2.2 describimos las principales tareas del *Meta-Planificador* y en la sección 1.3 se detalla el estado del arte de la organización del planificador en entornos Multicluster. En el capítulo 2 describimos el tipo de organización que hemos asumido en el presente trabajo, los componentes principales y su funcionalidad.

1.2.2. Meta-Planificación

Aunque la selección de recursos ha sido ampliamente estudiada en los sistemas paralelos y distribuidos tradicionales [Fei97], sistemas multiprocesador simétricos (SMPs), computadores paralelos masivos (MPPs) y clusters (NOW), hay ciertas consideraciones que no son válidas en entornos Multicluster, donde existen gran cantidad de recursos de naturaleza muy distinta distribuidos en distintos clusters, y compartidos con usuarios locales. En un entorno de estas características, se hace evidente la necesidad de añadir un nuevo nivel de planificación (*Meta-Planificador*) encargado de gestionar gran cantidad de recursos distribuidos y asignar los más adecuados a cada aplicación paralela.

La principal tarea del *Meta-Planificador* es la selección adecuada de los recursos para la ejecución eficiente de una aplicación paralela. El rendimiento del sistema

y su buen comportamiento, desde el punto de vista del usuario local y paralelo, dependen de la correcta asignación de recursos a las aplicaciones paralelas. Los recursos de cómputo y comunicación en un entorno Multicluster, pueden ser de naturaleza muy distinta, con parámetros de capacidad y rendimiento muy diversos. En un entorno no dedicado, además, estos parámetros pueden variar en el tiempo debido a problemas de saturación, variaciones en la carga local, fallos de recursos, etc.

Para poder llevar a cabo una selección eficiente de los recursos, en primer lugar debemos decidir el método de ***particionamiento*** más adecuado con el fin de facilitar la gestión de los mismos. En segundo lugar, es necesario establecer ***mecanismos de selección*** inteligentes que, en base al particionamiento realizado, nos permitan una selección adecuada de los recursos en el menor tiempo posible.

El método de particionamiento común en la mayoría de entornos Multicluster es un ***particionamiento estático*** donde cada cluster forma una única partición de tamaño fijo. Esto es así debido al particionamiento natural de los clusters. Cada cluster es independiente del resto, puede disponer de recursos de distinta naturaleza y utilizar políticas de planificación propias. En este tipo de particionamiento, si se permite la ejecución de aplicaciones paralelas en recursos de distintos clusters, en situaciones de elevada carga, los canales de comunicación entre clusters pueden convertirse en un cuello de botella debido a las elevadas latencias.

A continuación, el *Meta-Planificador* debe establecer ***mecanismos de selección*** que permitan valorar y comparar los recursos de distintas particiones con el fin de poder seleccionar la partición más adecuada para la ejecución de una aplicación paralela. Las técnicas más comunes para llevar a cabo esta valoración son las ***heurísticas*** y los ***modelos de rendimiento***.

Las técnicas ***heurísticas*** son más generales, no requieren de un detallado conocimiento acerca de la estructura de las aplicaciones y son más sencillas de implementar. Los modelos de rendimiento utilizan mayor información acerca de la estructura de las aplicaciones y modelan su comportamiento bajo determinadas condiciones de estado de los recursos.

Cuando el número de soluciones no es trivial, las heurísticas se convierten en una herramienta imprescindible. En la literatura podemos encontrar un gran número de estas *Heurísticas* [HSSY00, EHYS02, JLPS05]. Sin embargo, en los casos en que el tiempo necesario en explorar las posibles soluciones es suficientemente pequeño comparado con el tiempo de ejecución de las aplicaciones paralelas, es deseable la utilización de técnicas basadas en *modelos de rendimiento* [Aba04, TRA$^+$05, HHL$^+$06].

Los ***modelos de rendimiento*** pretenden llevar a cabo una valoración de los recursos, utilizando alguna métrica que permita cuantificar el rendimiento de una aplicación

en un determinado recurso. Esta valoración se lleva a cabo para cada aplicación antes de ser ejecutada, convirtiendo la planificación en una actividad predictiva. En la actualidad, muchos de los sistemas comerciales se basan en el uso de mecanismos de predicción [WSH99, BAG00, RI01, YSF03, Wol03].

El uso de la ***predicción*** permite realizar mapeos más adecuados de las aplicaciones paralelas obteniendo mejores resultados de rendimiento. Además ofrece la posibilidad de establecer mecanismos de QoS en donde se establecen ciertos rangos de rendimiento que se deben garantizar, como por ejemplo el tiempo máximo de ejecución de una aplicación, e incluso permite el uso de modelos económicos en donde el uso de determinados recursos y/o la definición de determinados límites llevan un coste asociado [CC00, AAB+00, ABG02, SAL+04]. Aunque son varias las métricas de rendimiento que pueden considerarse como el costo, la fiabilidad, la utilización de los recursos, etc, o bien enfoques multi-objetivo que combinan dos o más métricas [KNP01], nosotros nos centraremos en minimizar el tiempo de ejecución y el tiempo de espera de las aplicaciones paralelas.

En un entorno Multicluster donde los planificadores locales disponen de colas de espera y políticas de planificación propias, es tan importante poder estimar tanto el tiempo de ejecución de las aplicaciones paralelas como el tiempo de espera. Por este motivo, en el presente trabajo centramos nuestro interés en la predicción del tiempo que una aplicación paralela emplea desde que entra en el sistema hasta que finaliza su ejecución. Este tiempo que denominaremos ***turnaround time***, se define como la suma del tiempo transcurrido en las colas de espera más el tiempo de ejecución.

En el presente trabajo estamos interesados en el uso de la predicción para llevar a cabo una selección eficiente de los recursos que nos permita minimizar el tiempo de *turnaround* de las aplicaciones paralelas en entornos no dedicados. Dado que disponemos de un particionamiento estático de los recursos a nivel de cluster, utilizaremos la predicción para determinar el tiempo de *turnaround* de una aplicación en cada uno de los clusters. Además, en base a esta predicción propondremos nuevas meta-políticas que permitirán al *Meta-Planificador* una selección eficiente de los recursos más adecuados para cada aplicación en función de su estado.

En la sección 1.2.4 analizamos las técnicas de predicción actuales y en la sección 1.3 describimos el estado del arte en la predicción del *turnaround time* y las aportaciones del presente trabajo en este sentido.

1.2.3. Co-asignación

El principal inconveniente del uso del particionamiento estático es la posibilidad de producirse fragmentación a nivel de nodo. Esto significa que cuando una nueva

aplicación llega al sistema y es asignada a un cluster puede suceder que existan recursos disponibles que no sean suficientes para ejecutar la nueva aplicación y en consecuencia está deba esperar a la liberación de nuevos recursos, quedando recursos libres infrautilizados.

Un modo de paliar el efecto negativo de la fragmentación es permitir la compartición de los recursos libres entre distintos clusters. En general, definimos ***co-asignación*** como la asignación de recursos a las aplicaciones paralelas a través de *múltiples dominios* o clusters, de modo que las comunicaciones entre procesos se llevan a cabo mediante los canales de comunicación entre clusters.

Las principales ventajas del uso de la co-asignación son las siguientes:

1. Desaparición de la fragmentación a nivel de nodo. Aprovechando los recursos disponibles de distintos clusters podríamos obtener suficientes recursos para la nueva aplicación. Esto nos permitiría aprovechar mejor el conjunto de recursos y aumentar el rendimiento del sistema reduciendo el tiempo de espera de las aplicaciones.

2. Ejecución de aplicaciones con mayores requerimientos. Se puede dar el caso de la llegada de una aplicación paralela que no pueda ser asignada a ningún cluster en particular porque los recursos requeridos superan los disponibles en cualquiera de los clusters. Compartiendo los recursos entre clusters, el rango de recursos disponibles es muy superior y podemos atender aplicaciones con requisitos mucho mayores que de otro modo deberían ser desatendidos.

Así pues, Un *Meta-Planificador* que utiliza *co-asignación* puede incrementar de forma significativa su rendimiento. Aprovechando la fragmentación en los recursos de los distintos clusters que forman el Multicluster podemos reducir el tiempo de espera de las aplicaciones paralelas, aumentar la utilización de los recursos y disponer de un mayor número de recursos para la ejecución de aplicaciones con grandes requerimientos de recursos de cómputo.

Sin embargo, con el uso de la *co-asignación* se deben tener en cuenta otros factores que pueden perjudicar enormemente el rendimiento global del sistema. Cuando se realiza una asignación de recursos entre varios clusters, las comunicaciones entre procesos se llevan a cabo mediante canales de comunicación entre clusters, y estos canales son compartidos entre las aplicaciones paralelas y los usuarios locales. En una estructura Multicluster con recursos de comunicación heterogéneos y no dedicados, los canales de comunicación entre clusters podrían verse fácilmente saturados convirtiéndose en verdaderos cuellos de botella. En una situación de este tipo, el tiempo de ejecución de una aplicación co-asignada puede verse incrementado drásticamente.

Un planificador que pretenda usar *co-asignación* para obtener beneficios en el rendimiento, deberá tener en consideración información referente tanto a la utilización de los canales como a los requisitos de comunicación de las aplicaciones paralelas con el fin de mitigar el impacto negativo de la co-asignación sobre el tiempo de ejecución.

1.2.4. Técnicas de Predicción

La predicción se ha convertido en un componente fundamental en multitud de sistemas actuales [Dow97, WSH00, Wol03, LGTW04, LS05, TEF07] para llevar a cabo una selección eficiente de los recursos por parte del *Meta-Planificador*. Se han realizado grandes esfuerzos en predecir el tiempo de ejecución de una aplicación, tanto en sistemas dedicados [Gib97, Dow97, SFT98, IOP99, KFB99, SW02] como no dedicados [DI89, SB98, WSH99, Din01].

La estimación del tiempo de ejecución de una aplicación paralela en un determinado sistema es un campo de estudio emergente. Algunas de las técnicas más utilizadas hasta el momento son *Benchmarking*, *Cross-platform prediction*, *Application code analysis*, *Cycle-accurated simulators*, *Parameter estimation*s, etc.

Las técnicas de *Benchmarking* y *Cross-platform prediction* permiten estimar el rendimiento de un sistema bajo determinadas condiciones, mediante la ejecución de unas aplicaciones paralelas concretas o partes de ellas. La técnica *Cross-platform prediction* se utiliza además para estimar la ganancia en el rendimiento cuando se ejecuta la aplicación en una nueva arquitectura [YMM05]. Estas técnicas están orientadas al estudio del rendimiento de los sistemas y requieren información sobre el conjunto de aplicaciones paralelas y modelos de los sistemas difíciles de desarrollar [MSST08].

Las técnicas *Application code analysis* permiten modelar, mediante el análisis del código, el rendimiento de las aplicaciones paralelas basándose únicamente en los parámetros de entrada de la aplicación, sin considerar las características del sistema ni su ocupación. Los *cycle-accurated simulators* son modelos detallados del sistema y se utilizan habitualmente para el estudio de los microprocesadores. Estos sistemas requieren de información detallada del funcionamiento interno del sistema. Su implementación es costosa y el tiempo necesario en obtener una estimación es considerable [MSST08].

El principal inconveniente de estas técnicas es que no consideran las condiciones variables de los sistemas y/o requieren de un elevado coste en la obtención de las predicciones. Estas técnicas, que pueden ser muy útiles para caracterizar las aplicaciones paralelas, no son útiles para estimar de modo on-line su tiempo de ejecución considerando las características de los recursos y su ocupación.

Otro conjunto de técnicas que permiten reducir el tiempo de respuesta de las predicciones son las denominadas *Parameter estimation*. Estas técnicas consisten en la utilización de métodos de estimación que incorporan información sobre las aplicaciones y sobre la ocupación de los recursos y sus características.

Los métodos de estimación utilizados comúnmente en la literatura los podemos dividir en ***estadísticos*** -utilizan algún tipo de análisis estadístico sobre las aplicaciones que ya han sido ejecutadas-, y ***analíticos*** -construyen modelos de ejecución basados en ecuaciones que describen el comportamiento de las aplicaciones-.

Los *métodos estadísticos* llevan a cabo predicciones utilizando el análisis de *series temporales* [DI89, Din99, WSH00, Din01], métodos de *categorización* de las aplicaciones [Dow97, Gib97, SFT98], o métodos de *aprendizaje basados en casos* [KFB99, IOP99, SW02], denominados *IBL* (***I****nstance-****B****ased* ***L****earning*).

El análisis de ***series temporales*** es muy útil para predecir la disponibilidad de los recursos en el tiempo. Esta técnica se utiliza en algunos trabajos [Din01, Wol03] para predecir el rendimiento de las aplicaciones paralelas en función de la ocupación de los recursos donde son ejecutadas. Los métodos de categorización y aprendizaje llevan a cabo la predicción del tiempo de ejecución de una aplicación paralela basándose en la historia pasada. Estos métodos utilizan análisis estadísticos como la media, la regresión lineal, la regresión logarítmica, etc, de información contenida en una base de datos.

La información que se almacena en la base de datos determina el tipo de búsqueda y el tipo de análisis que podemos realizar. Cuando en la base de datos se almacena únicamente información sobre la aplicación paralela (Usuario, Cola de espera, Ejecutable , número de Nodos, etc), nos encontramos ante un método de ***categorización*** [Dow97, Gib97, SFT98]. La categorización se basa en la idea de que aplicaciones similares tienen tiempos de ejecución similares. Por lo tanto, se divide la información de la base de datos históricos en categorías, por ejemplo *(Usuario, Cola)* o bien *(Usuario, Ejecutable, Nodos)*, etc. Cuando una nueva aplicación llega al sistema, se busca la categoría en que se encuentra y se devuelve una estimación del tiempo de ejecución basándose en los datos de la misma categoría.

Cuando la base de datos históricos además de almacenar información sobre la aplicación se almacena información sobre el estado de los recursos, nos encontramos ante un método de predicción basado en el ***aprendizaje de casos*** [KFB99, IOP99, SW02], ***IBL*** (*Instance Based Learning*). Mediante esta técnica, la base de datos almacena un conjunto de experiencias. Una experiencia consiste en un conjunto de información de la aplicación más algún tipo de información sobre el estado de los recursos. Cuando una nueva aplicación llega al sistema, predecimos el tiempo de ejecución basándonos en la búsqueda de experiencias pasadas similares al estado

actual. La precisión en la estimación dependerá del número de experiencias registradas y de su similitud.

Los ***métodos analíticos*** desarrollan modelos que obtienen una medida de rendimiento bajo determinadas condiciones de carga de los recursos [SB98, JA05, HGH+06, JSK+06]. La medida de rendimiento utilizada depende del objetivo que se persigue. Distintos factores de rendimiento pueden ser el coste, la utilización de los recursos, el rendimiento global del sistema, el tiempo de ejecución de una aplicación, etc. Estos suelen utilizarse conjuntamente con un motor de simulación para llevar a cabo múltiples predicciones en el tiempo.

En el presente trabajo estamos interesados en estimar el tiempo de espera y ejecución de una aplicación paralela, es decir el tiempo de *turnaround*. Para llevar a cabo esta tarea, proponemos dos técnicas de predicción distintas. La primera basada en un método de aprendizaje por casos (sección 3.2.2) y la última basada en un modelo analítico (sección 3.3.1) que será utilizado junto con un simulador para estimar el tiempo de ejecución tanto de las aplicaciones actualmente en ejecución como de las que están en espera.

Realizaremos un estudio comparativo de la precisión de los distintos métodos y propondremos un método predictivo basado en una combinación de las mejores técnicas propuestas y estudiadas.

1.3. Estado del arte

Las técnicas de planificación han ido evolucionando según la arquitectura de los sistemas. El rendimiento de un sistema está directamente relacionado con la organización del planificador y las técnicas de planificación utilizadas. Los primeros estudios sobre distintas arquitecturas en sistemas de memoria distribuida vienen de la mano de Feitelson y Rudolph [FR90], aunque la discusión se centra únicamente en sistemas centralizados o distribuidos. Hasta principios de la presente década no se propone una arquitectura de planificación jerárquica para sistemas distribuidos [TD01].

En la actualidad, gran número de sistemas comerciales [Con, Ope, Mau, Xu01, Loa, BAG00] utilizan ***organizaciones jerárquicas***. Bucur analiza en [BE02, BBE03] distintas organizaciones del planificador y concluye que la mejor opción es disponer de planificadores en cada cluster y un planificador global con información del resto del sistema para poder llevar a cabo planificaciones más eficientes. Numerosos estudios [SvANS00, SKSS02, EHS+02, Aba04, TRA+05] avalan los buenos resultados de rendimiento obtenidos mediante el uso de una organización jerárquica y el uso adecuado de las políticas de selección de recursos.

En el presente trabajo nos basamos en una estructura jerárquica con dos niveles de planificación. En el primer nivel disponemos de un planificador global (Meta-Planificador), encargado de recoger información sobre el estado de los recursos y decidir el conjunto de recursos idóneo para lanzar una determinada aplicación paralela. En un segundo nivel disponemos de planificadores locales en cada cluster, con sus propias políticas de planificación y sus propios recursos.

A nivel de planificador local utilizaremos algunas de las técnicas más comunes para el aprovechamiento de los recursos de cómputo libres, el aumento del ***grado de paralelismo*** y la utilización del ***contrato social***, descritas con detalle en la sección 1.1 en las páginas 8 y 10. Estas técnicas, ampliamente estudiadas con anterioridad en nuestro grupo de investigación, las consideramos la base de los planificadores locales y el punto de partida del presente trabajo.

En la última década han surgido numerosos estudios entorno a las selección de recursos en sistemas geográficamente distribuidos [HSSY00, TRA+05, DA06]. Una técnica que ha suscitado especial atención y que está siendo ampliamente utilizada en este tipo de entornos es la ***predicción***. La estimación de la utilización de determinados recursos o el rendimiento de una determinada aplicación es información muy valiosa para poder llevar a cabo una planificación que cumpla con los requisitos de rendimiento deseados (utilización del sistema, rendimiento de las aplicaciones, costes económicos, etc).

Los ***métodos de predicción*** generalmente se dividen en ***estadísticos*** [Dow97, Gib97, WSH00, Din01, SW02, LS05, YA07], los que utilizan algún tipo de análisis estadístico sobre aplicaciones que ya han sido ejecutadas, y ***analíticos*** [SB98, JA05, HGH+06, JSK+06], los que construyen ecuaciones que describen el modelo de ejecución de las aplicaciones.

La predicción se ha convertido en un componente fundamental en multitud de sistemas actuales [Dow97, WSH00, Wol03, LGTW04, LS05, TEF07]. Se han realizado grandes esfuerzos en predecir el tiempo que una aplicación invierte en su ejecución, tanto en sistemas no dedicados [DI89, SB98, WSH99, Din01] como dedicados [Gib97, SFT98, IOP99, KFB99], prestando menor atención a las predicciones de los tiempos de espera en las colas del sistema [Dow97, SW02, LGTW04].

En un entorno Multicluster con técnicas de Meta-Planificación basadas en predicción, donde los planificadores locales disponen de colas de espera y políticas de planificación propias, es tan importante poder estimar el *tiempo de ejecución* como el *tiempo de espera* en las colas antes de ser ejecutado. Entendemos por *tiempo de ejecución* el tiempo transcurrido desde el inicio de la ejecución de la aplicación hasta su finalización (*wall-clock time*).

Mediante el uso de métodos estadísticos se han obtenido buenos resultados en cuanto a la predicción del tiempo de ejecución, ya que aplicando técnicas estadísticas a

la historia pasada de una determinada aplicación se puede aproximar con una desviación aceptable el tiempo de ejecución. Sin embargo, no sucede lo mismo con la predicción del tiempo de espera en las colas. Dado que la espera en las colas tiene que ver tanto con la ocupación del sistema como con las políticas de planificación que utiliza cada planificador local se requieren mecanismos de predicción más sofisticados.

En el presente trabajo proponemos dos nuevos métodos de predicción. El primero está basado en un modelo analítico, presentado en la sección 3.3.1, donde consideramos tanto las características de cómputo como de comunicación del sistema para obtener una predicción del tiempo de ejecución. El segundo se basa en el análisis estadístico de la información obtenida a través de una base de datos históricos donde además de guardar información sobre el rendimiento de la aplicación se guardan algunos parámetros sobre el estado del sistema (sección 3.2.2).

Analizamos el comportamiento de ambos métodos bajo distintas situaciones de carga tanto paralela como local y comparamos los resultados obtenidos con otras propuestas de la literatura. A continuación, realizamos un estudio de la precisión en la estimación tanto del tiempo de ejecución como del tiempo de espera en las colas de los planificadores locales. Finalmente proponemos un módulo de predicción, que combina las técnicas que ofrecen los mejores resultados, reduciendo así la desviación en la predicción tanto del tiempo de espera como de ejecución de una aplicación paralela.

Por lo general, los trabajos anteriores basados en la estimación del tiempo de ejecución sólo tienen en cuenta el estado de los recursos de cómputo, sin considerar la ocupación de los canales de comunicación [BAG00, Wol03, HHL+06]. En [DZ97, BE03b, JA05] se presentan distintos ***modelos de comunicación***. Aunque estos modelos permiten valorar el rendimiento del sistema asumiendo valores predeterminados de tamaño y latencia de envío de mensajes, no son útiles para predecir el tiempo de ejecución y tomar decisiones de planificación on-line, es decir, a medida que las aplicaciones paralelas van llegando al sistema.

Jones en [JLPS05] presenta un modelo que considera los requisitos de ancho de banda de las aplicaciones para medir el impacto de la saturación en el tiempo de ejecución de las aplicaciones paralelas. Sin embargo, el modelo asume un entorno homogéneo y dedicado no considerando la diversidad en la capacidad de los recursos de cómputo ni el dinamismo de la carga. Hanzich en [HGH+06] presenta un ***modelo analítico de cómputo*** para estimar el tiempo de ejecución de un trabajo paralelo, basándose únicamente en el número de trabajos paralelos ejecutándose concurrentemente. A pesar de los buenos resultados obtenidos, no se consideran los recursos de comunicación.

En el presente trabajo proponemos una extensión de los modelos analíticos de Hanzich y Jones, que nos permita realizar una estimación del tiempo de ejecución de una aplicación paralela considerando la utilización de ambos recursos, cómputo y comunicación, por parte de los usuarios locales y paralelos.

A finales de los 90, coincidiendo con la aparición de los primeros sistemas que aprovechan los recursos de cómputo geográficamente distribuidos [FK97, Glo], surge la necesidad de establecer mecanismos que permitan la gestión de los recursos distribuidos [CFK+98] y la ejecución de aplicaciones utilizando recursos procedentes de distintas ubicaciones [FK98]. A raíz de esta necesidad, surge un gran número de estudios [SCJG00, BE02, EHS+02, BE00, BE01, JLPS05] que analizan la influencia en el rendimiento producida por la ***co-asignación*** de aplicaciones a través de distintos dominios administrativos.

En [BE03a] se analiza el efecto de la *co-asignación* sobre la utilización y se demuestra que la co-asignación permite aumentar la utilización de los sistemas Multicluster. Ernemann [EHS+02] analiza el efecto de la co-asignación sobre el tiempo de ejecución de las aplicaciones paralelas, y afirma que el uso de la co-asignación sigue siendo beneficioso para el usuario paralelo incluso con una penalización en el tiempo de ejecución de las aplicaciones paralelas del 25 %.

Por otro lado, Bucur [BE01] analiza la influencia en el rendimiento de la cantidad de comunicación y la ratio entre la capacidad de comunicación interna y externa a los cluster demostrando que el aumento de los requisitos de comunicación y la ratio entre los canales de comunicación tienen una influencia negativa sobre el rendimiento. Jones [JLPS05] analiza el efecto sobre el tiempo de ejecución que tienen los requisitos de comunicación y las políticas de selección de recursos. Asegura que la penalización que pueden sufrir las aplicaciones paralelas en el tiempo de ejecución depende del número de clusters donde la aplicación se expande y que la tarea de co-asignación debe minimizar al máximo el número de canales de comunicación que las aplicaciones han de utilizar.

La conclusión principal de todos estos estudios es que el uso de la co-asignación puede provocar la obtención de un rendimiento muy pobre cuando se produce ***la saturación de los canales de comunicación*** entre clusters.

Con el fin de evitar esta situación, en la literatura se proponen dos soluciones distintas. La primera consiste en ***limitar la co-asignación*** a aquellas aplicaciones que cumplan con determinados criterios, basados normalmente en las necesidades de comunicación de las aplicaciones. La segunda consiste en la utilización de mecanismos de búsqueda de la ***asignación óptima***, es decir, aquella que minimice el tiempo de ejecución de la aplicación paralela y que no produzca saturación en los canales de comunicación.

Bucur [BE02] afirma que considerar los recursos de un Multicluster como un único super-cluster, donde cualquier trabajo puede ser co-asignado, perjudica enormemente el rendimiento y propone limitar la co-asignación a trabajos con requisitos de comunicación reducidos. Por otro lado, Jones [JLPS05] propone una heurística muy conservadora limitando la co-asignación a aquellos trabajos donde el 85 % de las tareas pueda ser asignado a un solo cluster.

Sin embargo, aunque el uso de estas *heurísticas* permite obtener buenos resultados en algunos casos, estas técnicas no consideran las características particulares de las aplicaciones paralelas ni el estado de los recursos, y por tanto no garantizan que los canales de comunicación estén libres de saturación.

La asignación de un conjunto fijo de tareas sobre un conjunto finito de recursos con distintas capacidades, se puede plantear como un ***problema de optimización***. Este tipo de problemas se caracterizan por un coste computacional elevado, debido a que el número de posibles asignaciones crece exponencialmente con el número de recursos. Sin embargo, este tipo de problemas obtienen una solución óptima y su uso puede resultar beneficioso cuando el número de recursos no es excesivamente elevado o cuando el tiempo de cómputo de la solución es factible comparado con el tiempo de ejecución de las aplicaciones paralelas.

Trabajos anteriores [Oma77, BBC+04, NLYW05] ilustran los beneficios de la utilización de ***modelos de programación entera*** en la resolución de problemas de planificación. Sin embargo, Naik [NLYW05] proporciona una solución óptima al rendimiento del sistema asumiendo que la cola de trabajos paralelos es conocida de antemano, y Banino [BBC+04], propone un modelo para la búsqueda de la temporización óptima de una tarea master-slave con el fin de reducir el tiempo de ejecución, difícil de llevar a la práctica.

Con el fin de paliar los efectos negativos de la co-asignación, en el presente trabajo, proponemos dividir la tarea de planificación en dos fases. En la primera fase, los trabajos que llegan al Meta-Planificador se asignan, mediante el uso del mecanismo de predicción, al cluster que retorna el menor tiempo de turnaround. En una segunda fase, el Meta-Planificador recorre la cola de trabajos que ya han sido asignados a algún cluster y que están en espera de ser ejecutados por falta de recursos. Cuando se encuentra algún trabajo que puede se ejecutado utilizando los recursos libres del resto de clusters, entonces se entrega el trabajo al módulo de co-asignación.

El módulo de co-asignación es el encargado de decidir si se puede llevar a cabo la co-asignación de la aplicación paralela y qué recursos se deben utilizar para obtener el menor tiempo de ejecución. Para llevar a cabo esta tarea, en el presente trabajo, capítulo 4 sección 4.6.2, proponemos un ***modelo de programación entera binaria****, que denominaremos* BIP *(***B**inary **I**nteger **P**rogramming*), basado en el mo-*

delo analítico que hemos propuesto para la estimación del tiempo de ejecución de una aplicación paralela. Estudiaremos su comportamiento bajo distintas condiciones de carga y compararemos el rendimiento obtenido con el uso de la heurística propuesta por Jones en [JLPS05].

Con este modelo pretendemos evitar la saturación de los canales de comunicación durante la co-asignación de aplicaciones paralelas, obteniendo con ello una mejora de la utilización de los recursos y del rendimiento global del sistema, así como una mayor precisión en las predicciones del tiempo de ejecución, ya que las situaciones de saturación producen resultados erróneos en la predicción. Por otro lado, el modelo tiene en cuenta la ocupación de recursos que forman el entorno Multicluster, garantizando un bajo impacto sobre el rendimiento de las aplicaciones de los usuarios locales.

1.4. Objetivos

En la actualidad, existe una gran disponibilidad de recursos computacionales, repartidos en distintas unidades departamentales dentro de una misma organización, empresa, centro de investigación o universidad. Esta oportunidad lleva a los investigadores a pensar en la posibilidad de aprovechar los recursos de forma global para poder ejecutar aplicaciones con mayores requisitos computacionales y por lo general mucho más complejas.

En el caso que nos ocupa, estamos interesados en aprovechar los recursos en un entorno universitario, empresa, organismo u organización, donde podemos disponer de varios clusters formados por recursos procedentes de aulas docentes y/o áreas de investigación, redes de áreas distintas (i.e. gerencia, recursos humanos, producción, contabilidad, etc.), formando un Multicluster. El desafío principal en este tipo de entornos es la gestión eficiente de los recursos. Los recursos se encuentran agrupados en clusters independientes con sus propias técnicas de planificación y restricciones. La cantidad de recursos crece sustancialmente y pueden ser de características muy distintas. Además, los recursos se deben compartir con usuarios locales a los cuales se les debe asegurar el confort necesario para que se presten a ceder sus recursos.

Como se ha presentado anteriormente, es posible encontrar en la literatura distintas soluciones al problema de la Meta-planificación, pero la mayoría de estas soluciones se centran en entornos Grid, donde el ámbito de las comunicaciones es mayor al de una única organización (WAN), lo que dificulta la gestión y las posibilidades de aprovechamiento de los recursos. En un entorno Multicluster, donde las comunicaciones se producen en el ámbito de una organización, las bajas latencias y la

predicibilidad en la utilización de los recursos, permiten un mayor control y aprovechamiento de los recursos.

El objetivo principal de este trabajo es el estudio de nuevas técnicas de Meta-planificación que nos permitan obtener el máximo rendimiento de las aplicaciones paralelas en entornos Multicluster no dedicados, minimizando el tiempo de ejecución de las aplicaciones paralelas sin interferir en la interactividad de las tareas locales.

Este objetivo global se divide en los siguientes objetivos más concretos:

1. Estudiar las técnicas habituales de predicción y proponer nuevas técnicas que incrementen la precisión en las estimaciones. Para llevar a cabo una gestión eficiente de los recursos por parte del Meta-planificador, es habitual el uso de la predicción para estimar el rendimiento de una aplicación con el uso de determinados recursos bajo ciertas condiciones de carga. En nuestro caso concreto, estamos interesados en estimar el tiempo de espera y ejecución de una aplicación paralela, también denominado *turnaround time*. Cuanto mayor sea la precisión en la predicción, mayor será la eficiencia del Meta-planificador y mejores resultados de rendimiento podremos obtener. Para cumplir con este objetivo, llevaremos a cabo las siguientes tareas:

    - Proponer un nuevo ***modelo analítico*** para estimar el tiempo de ejecución de una aplicación paralela, que considera además de la capacidad de cómputo la capacidad de comunicación de los recursos.
    - Proponer dos nuevas técnicas de estimación. La primera basada en un ***método estadístico*** aplicado a una base de datos históricos y la segunda basada en el uso de un motor de simulación guiado por el ***modelo analítico*** propuesto en el punto anterior.
    - Analizando la precisión de las técnicas propuestas, para la estimación del tiempo de espera y de ejecución, proponemos una nueva ***estrategia de predicción*** basada en una combinación de las técnicas de estimación anteriores que nos permita obtener mejores resultados de precisión en la estimación del *turnaround time*.

2. Estudiar las técnicas de meta-planificación más habituales y proponer una nueva técnica que permita aprovechar al máximo los recursos distribuidos y una ejecución eficiente de las aplicaciones paralelas. Esta nueva técnica deberá considerar el uso de la predicción para seleccionar el cluster que ejecutará más eficientemente la aplicación paralela sin perjuicio del usuario local. Además, deberá considerar el uso de la co-asignación cuando las aplicaciones

asignadas a los clusters locales deban esperar excesivamente o bien no existan suficientes recursos en los distintos clusters para ejecutar la aplicación paralela. El uso de la co-asignación implica llevar a cabo una selección óptima de los recursos que considere tanto las capacidades de cómputo como de comunicación de los mismos. Para cumplir con este objetivo, llevaremos a cabo las siguientes tareas:

- Proponer un nuevo modelo de selección de recursos, que nos proporcione la mejor solución para minimizar el tiempo de ejecución de una aplicación paralela durante un proceso de co-asignación.
- Proponer un técnica eficiente de meta-planificación que combinando el uso de las técnicas de predicción y co-asignación permita minimizar el tiempo de ejecución de las aplicaciones paralelas aprovechando al máximo los recursos del entorno Multicluster.
- Estudiar el rendimiento de la meta-política propuesta y comparar con otras de la literatura.

1.4.1. Estructura del documento

El resto de la memoria se organiza de la forma siguiente. En el capítulo 2, se describen los fundamentos de la plataforma *M-CISNE* (***M**eta-**C**ooperative and **I**ntegral **S**cheduler for **N**on-dedicated **E**nvironments*).

M-CISNE es un framework de meta-planificación de aplicaciones paralelas en entornos Multicluster no dedicados, diseñado y desarrollado durante el presente trabajo. Esta plataforma ha servido de base para el desarrollo de las nuevas propuestas de meta-planificación y el estudio de su rendimiento.

En este capítulo se presenta la organización jerárquica de planificación (sección 2.2), los módulos que componen los distintos subsistemas de planificación y sus principales funcionalidades (sección 2.3 y 2.4). Para finalizar, se describe de forma detallada las operaciones básicas de la tarea de planificación en cada nivel (sección 2.5). En cada fase de la tarea de planificación se indican los componentes que intervienen y la información que requieren.

En el capítulo 3 se realiza un análisis detallado de los distintos métodos de predicción existentes en la literatura. A continuación se proponen dos nuevos métodos de predicción con el fin de aumentar la precisión de las estimaciones en un entorno Multicluster no dedicado. El primero es un método estadístico basado en el aprendizaje de casos (sección 3.2) y el segundo es un método analítico integrado en un motor de simulación (sección 3.3).

En la sección 3.5 se analiza el comportamiento de los métodos predictivos propuestos para distintas situaciones de carga y políticas de planificación, y se comparan los resultados obtenidos con los mejores métodos de la literatura.

Finalmente, en la sección 3.6, se propone un nuevo mecanismo para la estimación del tiempo de *turnaround* y se evalúan los resultados de precisión obtenidos.

En el capítulo 4 se presentan nuestras propuestas en el contexto de la tarea de meta-planificación. En primer lugar, se presentan los principales pasos de la tarea de meta-planificación y se sitúan las distintas propuestas realizadas (sección 4.2).

En la sección 4.3 se propone una solución para la *selección de recursos* basada en el mecanismo de predicción presentado en el capítulo anterior. En la sección 4.4 se propone un mecanismo para la *selección de trabajos* que permita la re-asignación de aplicaciones paralelas en función de los cambios de estado del sistema. En la sección 4.5 se propone un método para establecer el *tamaño óptimo de la cola* de cada cluster de forma dinámica en función de su nivel de carga.

En la sección 4.6 se propone una solución para la *co-asignación* de aplicaciones paralelas a varios clusters. Mediante el uso de un modelo de programación entera binaria (*Mixed Integer Programming*) se propone la búsqueda de la mejor asignación posible considerando las características y la ocupación de los recursos de cómputo y comunicación.

Finalmente, en la sección 4.7 se presenta un refinamiento a la *tarea de meta-planificación,* presentada en la sección 4.2, que amplía el rango de aplicaciones co-asignadas bajo determinados criterios con el fin de obtener un mayor rendimiento y considerando la no saturación de los canales entre clusters.

En el capítulo 5 se presentan las principales conclusiones que se extraen del trabajo realizado en los distintos ámbitos de la tarea de meta-planificación y se exponen el conjunto de líneas de trabajo abiertas que consideramos interesante abordar en un trabajo futuro.

Capítulo 2

Entorno M-CISNE

En el presente capítulo presentamos la plataforma ***M-CISNE***(*Meta-Cooperative & Integral Scheduling for Non-dedicated Environments*), una herramienta de planificación de trabajos paralelos en entornos Multicluster no dedicados que ha servido de base para el desarrollo del presente estudio.

M-CISNE se ha construido a partir de un esquema de planificación jerárquico con dos subsistemas distintos de planificación. En el primer nivel se encuentra *MetaLoRaS*, un *meta-planificador* desarrollado durante el presente trabajo [LSG$^+$06], encargado de recibir las tareas paralelas y distribuirlas según una determinada *meta-política* a los planificadores de segundo nivel.

En el segundo nivel se encuentra *LoRaS*, un planificador de trabajos paralelos para entornos cluster no dedicados desarrollado con anterioridad en nuestro grupo de investigación [HGH$^+$05a, HGH$^+$04], que ha sido adaptado y integrado dentro de *M-CISNE* para interactuar con el meta-planificador de nivel superior *MetaLoRaS*.

En la sección 2.1 se describen los objetivos de la nueva plataforma *M-CISNE* y sus principales características. En la sección 2.2 se identifican los subsistemas principales de la plataforma M-CISNE y el modo de interacción. En la sección 2.3 se describen los componentes principales del subsistema de meta-planificación (*MetaLoRaS)* y sus funcionalidades. En la sección 2.4 se describen los componentes principales del subsistema de planificación a nivel de cluster (*LoRaS*), prestando especial atención a los componentes necesarios para la interacción con el subsistema de nivel superior, *MetaLoRaS*. En la sección 2.5 se describen las operaciones básicas de la tarea de planificación en cada uno de los niveles. En cada fase de la tarea de planificación se indican los componentes que intervienen y la información de configuración que se necesita.

2.1. Introducción

La tarea principal de la plataforma *M-CISNE* es la planificación de aplicaciones paralelas en entornos Multicluster no dedicados que permita maximizar el rendimiento de las aplicaciones paralelas garantizando al mismo tiempo la capacidad de respuesta del usuario local. A continuación se explica de forma detallada las principales funcionalidades de la plataforma M-CISNE:

1. *Administrar tareas y recursos en un entorno Multicluster no dedicado*. Necesitamos disponer de un entorno donde se pueda incorporar de forma sencilla nuevas técnicas de meta-planificación y selección de recursos.

2. *Proporcionar mecanismos eficaces de monitorización* que permitan la obtención de información sobre el estado de los recursos del sistema y el estado de las aplicaciones paralelas.

3. *Proporcionar un repositorio de datos* donde almacenar información detallada sobre el proceso de ejecución de las aplicaciones paralelas. El uso de un repositorio persigue básicamente dos finalidades, proporcionar información on-line a los sistemas de planificación para la toma de decisiones y/o el posterior análisis del rendimiento de las estrategias utilizadas.

Dado el carácter experimental del presente trabajo, se necesita que la plataforma sea lo suficientemente flexible como para adaptarla a nuevos requerimientos o cambios en las estrategias de planificación de la forma más sencilla posible. Para poder llevar a cabo esta tarea necesitamos que nuestra plataforma cumpla con los siguientes requisitos:

- **Arquitectura modular**. Necesitamos que el entorno nos permita implementar y evaluar nuevas técnicas de planificación y/o predicción de la forma más sencilla posible. Esto requiere de un diseño sin muchas interdependencias que permita ampliar, modificar o eliminar componentes al menor coste. La implementación de estas características permitirá además analizar el rendimiento de nuevas aportaciones y ajustarlas a nuestras necesidades.

- **Escalabilidad y Portabilidad**. Un Multicluster está formado por recursos de procesamiento repartidos entre distintos departamentos de una misma organización. La cantidad de estos recursos puede ser muy grande y/o aumentar progresivamente con el tiempo. Se pretende diseñar un sistema escalable que permita aumentar el número de recursos sin afectar negativamente al rendimiento. Por otro lado, los recursos de que se dispone pueden renovarse o

incluso cambiar de ubicación con el tiempo y es por eso que necesitamos diseñar un sistema que atienda la diversidad tanto en la composición como en la cantidad de recursos disponibles.

- **Flexibilidad**. Ha de proporcionar mecanismos que permitan incorporar de forma sencilla nuevos sistemas de predicción y planificación. Esto nos permitirá analizar el rendimiento de nuevas aportaciones y ajustarlas a nuestras necesidades.
- **Predicibilidad.** Para que el sistema sea útil para el usuario paralelo es importante dotarlo de mecanismos que permitan predecir el comportamiento de una aplicación: tiempo de espera, tiempo de ejecución, consumo de recursos, etc.
- **Fácilmente configurable.** Dado el carácter experimental de la herramienta, se necesita una interfaz flexible para la gestión y control de las aplicaciones paralelas que permita la configuración del sistema de forma sencilla.

En la actualidad, no disponemos de herramientas comerciales para la planificación de aplicaciones paralelas en entornos Multicluster que nos permita incorporar nuevas estrategias de meta-planificación, de selección de recursos y mecanismos de predicción, de una forma rápida y eficaz para su posterior estudio de rendimiento.

Esto nos ha llevado al diseño y desarrollo de una nueva plataforma de planificación de aplicaciones paralelas en entornos Multicluster no dedicados denominada *M-CISNE*, partiendo de una herramienta para la planificación de aplicaciones paralelas en entornos cluster no dedicados diseñada dentro de nuestro grupo de investigación, denominada *LoRaS*.

La nueva plataforma dispone de una arquitectura jerárquica con dos niveles de planificación. En el primer nivel disponemos de *MetaLoRaS*, subsistema de meta-planificación encargado de comunicarse con el usuario paralelo, recibir las cargas paralelas, decidir donde se deben planificar según la meta-política escogida, y proporcionar mecanismos para la gestión de los recursos y de las aplicaciones paralelas.

En el segundo nivel disponemos de *LoRaS*, subsistema de planificación de trabajos paralelos a nivel de cluster. Este subsistema implementa políticas de selección de trabajos y recursos a nivel de cluster y proporciona mecanismos de monitorización del estado de los recursos y de las aplicaciones paralelas.

LoRaS ha sido adaptado para dar servicio al meta-planificador de nivel superior (*MetaLoRaS*) añadiendo una nueva interfaz de comunicaciones y ampliando las funcionalidades del motor de predicción y del sistema de monitorización.

A continuación, en la sección 2.2 presentamos la arquitectura de la nueva plataforma *M-CISNE*.

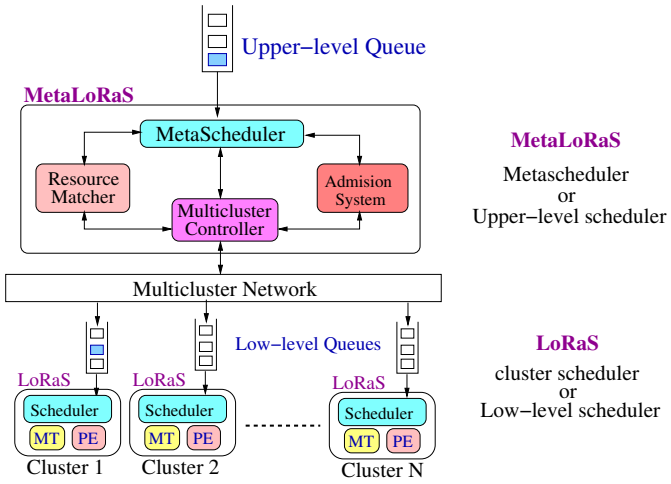


Figura 2.1: Arquitectura de M-CISNE

2.2. Arquitectura del entorno M-CISNE

Una de las primeras cuestiones que hay que plantearse en entornos formados por varios clusters es el modo en que se organiza el sistema de planificación. En el capítulo 1, hemos descrito distintas organizaciones de planificación y hemos justificado el uso de una arquitectura jerárquica para entornos Multicluster.

M-CISNE se compone de una arquitectura jerárquica con dos subsistemas de planificación distintos, *MetaLoRaS* en el nivel superior y *LoRaS* en cada uno de los clusters. En la figura 2.1 se muestra la jerárquica de planificación y la interacción entre niveles.

MetaLoRaS es el subsistema encargado de la *meta-planificación* y se compone principalmente de los módulos ***Upper-level Queue****(UQ)*, ***Metascheduler*** *(MS)*, ***Admission*** *(AS),* ***Resource Matcher*** *(RM),* y ***Multicluster Controller*** *(MC)*. El subsistema *LoRaS* se encarga de la planificación a nivel de cluster y de la monitorización de los recursos. Los componentes principales que interactúan con el planificador de nivel superior (*MetaLoRaS*) son el gestor de colas local ***Low-level Queue*** *(LQ)*, el módulo de monitorización de los recursos ***Monitoring ToolKit*** *(MT)* y el motor de predicción ***Prediction Engine*** *(PE)*.

El subsistema de primer nivel, **MetaLoRaS**, recibe las aplicaciones del usuario paralelo y decide donde deben lanzarse. Además, proporciona al usuario paralelo información sobre el estado de las aplicaciones, el estado de los recursos y una estimación del tiempo que la aplicación permanecerá en el sistema.

Para poder llevar a cabo la tarea de meta-planificación de forma eficaz, se necesita disponer de información acerca del estado de los recursos y de las aplicaciones que se están ejecutando.

MetaLoRaS se comunica con los planificadores de segundo nivel (*LoRaS*) a través del ***Multicluster Controller*** con el fin de mantener información actualizada sobre el estado de los recursos, el estado de las aplicaciones paralelas y de solicitar estimaciones del tiempo de *turnaround* que permitan al *MetaLoRaS* tomar mejores decisiones de planificación.

El subsistema de segundo nivel, **LoRaS**, recibe las aplicaciones paralelas de *MetaLoRaS* y selecciona los recursos más adecuados para su ejecución respetando el uso de los recursos por parte del usuario local.

Para poder llevar a cabo una selección adecuada de los recursos, en ambos niveles de planificación, se necesita disponer de información actualizada sobre la capacidad y ocupación de los nodos de cada cluster. ***Monitoring Toolkit (MT)*** es el módulo encargado de la monitorización de los recursos. *MetaLoRaS* accede a esta información mediante el *Multicluster Controller*.

LoRaS está dotado de un motor de predicción que le permite estimar el tiempo de *turnaround* de una aplicación paralela. El módulo encargado de llevar a cabo las predicciones es ***Prediction Engine*** (***PE***). *MetaLoRaS* solicita las predicciones a cada cluster a través del *Multicluster Controller*.

El subsistema **LoRaS** proporciona un entorno flexible para la definición y incorporación de nuevas políticas de planificación en entornos cluster no dedicados. Además, ha sido dotado de un nuevo interfaz de comunicaciones que facilita su configuración, el control de las aplicaciones, y el acceso a información sobre el estado de los recursos y las estimaciones. Esto ha facilitado el intercambio de información con el planificador de nivel superior, MetaLoRaS, y por lo tanto su completa integración dentro de la nueva plataforma *M-CISNE*.

En la sección 2.3 se describen los componentes principales del subsistema MetaLoRaS y su funcionalidad en el proceso de meta-planificación. En la sección 2.4 describimos el subsistema *LoRaS* y la funcionalidad de los módulos que lo integran, prestando especial atención a los encargados de la predicción por ser uno de los objetivos principales del presente trabajo.

2.3. Arquitectura del sistema MetaLoRaS

Estamos interesados en el desarrollo de un meta-planificador para sistemas Multicluster no dedicados con una organización jerárquica de dos niveles. En una ar-

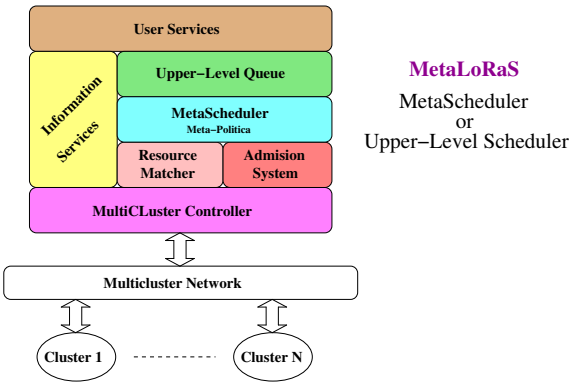


Figura 2.2: Arquitectura de MetaLoRaS

quitectura jerárquica, la planificación se realiza de forma distribuida entre varios niveles de planificación, reduciendo así el *overhead* computacional del proceso de planificación de los sistemas centralizados y/o descentralizados [BE02, TRA+05].

En el primer nivel se encuentra el meta-planificador, denominado **MetaLoRaS**, encargado de recibir cada trabajo paralelo, seleccionar un cluster y enviar el trabajo al planificador de nivel más bajo (*LoRaS*) del cluster seleccionado. En el segundo nivel se encuentra el subsistema *LoRaS*, encargado de planificar los trabajos a nivel de cluster seleccionando los recursos más adecuados para la ejecución de las aplicaciones paralelas respetando siempre unos recursos mínimos para el usuario local.

Los componentes principales de **MetaLoRaS** (Figura 2.2) son el sistema de colas ***Upper-level Queue***(*ULQ*), el meta-planificador ***Metascheduler***, el módulo de asignación de recursos ***Resource Matcher*** (*RM*), el sistema de admisiones ***Admission System*** (*AS*) y el módulo controlador de los recursos del Multicluster ***Multicluster Controller*** (*MCC*).

El ***Upper-level Queue*** (*ULQ*) es el gestor de colas del meta-planificador. Cuando una aplicación paralela llega al sistema se introduce en el sistema de colas del meta-planificador a través del módulo *Upper-level Queue*. Este módulo facilita la definición de varias colas permitiendo la clasificación de las aplicaciones según la prioridad, el número de tareas, el tipo de aplicación, los requisitos de comunicación, etc. En el presente trabajo, asumimos la utilización de una única cola donde se admiten trabajos paralelos de cualquier índole.

MetaScheduler (*MS*) es el meta-planificador del sistema Multicluster. Su función principal es la asignación adecuada de recursos a los trabajos paralelos a fin de cumplir con los objetivos de rendimiento marcados por el usuario o el administrador del sistema. La tarea de meta-planificación se compone de varias fases, la selección

del siguiente trabajo a ejecutar, la pre-selección de los recursos candidatos para la ejecución y finalmente asignar el trabajo a dichos recursos. En cada una de estas fases se pueden aplicar distintas estrategias según los objetivos perseguidos. El conjunto de estrategias aplicadas durante el proceso de planificación lo denominamos *meta-política*.

Durante el presente trabajo propondremos nuevas estrategias que nos permitan definir nuevas meta-políticas con el fin de proporcionar al usuario paralelo el mejor rendimiento sin perjudicar a los usuario locales.

El ***Admision System*** (*AS*) es el módulo responsable de admitir o rechazar los trabajos que llegan al sistema. Una vez el *MetaScheduler* ha seleccionado un nuevo trabajo para ser planificado, el *Admision System* comprueba si los requisitos del trabajo paralelo (sistema operativo, cantidad total de memoria, cantidad total de nodos, ancho de banda, capacidad de cómputo de los recursos, etc) pueden ser satisfechos por los recursos del sistema. Cuando los recursos del sistema no cumplen con los requisitos de un trabajo paralelo, éste se rechaza. Por otro lado, si existen suficientes recursos en el sistema pero no están disponibles, el *Admision System* permite establecer criterios de permanencia de los trabajos en el sistema de colas, de modo que trabajos con grandes requerimientos y largos tiempos de espera no permanezcan indefinidamente en el sistema de colas.

El ***Resource Matcher*** (*RM*) es el responsable de la asignación (*mapping*) de recursos a los trabajos paralelos. La tarea de asignación se compone de las fases siguientes: (1) llegada de una petición de mapping procedente del *MetaScheduler*, (2) consulta del estado actual de los recursos a través del *Multicluster Controller*, (3) obtención de la información del trabajo paralelo, (4) obtención de una solución de mapping según la estrategia seleccionada y (5) mapping del trabajo paralelo según los resultados obtenidos.

En la literatura existe una gran cantidad de estrategias para llevar a cabo esta asignación, modelos heurísticos, modelos de rendimiento aplicados a la predicción, modelos de rendimiento para la búsqueda de la solución óptima, etc. En el presente trabajo proponemos nuevas estrategias basadas en el uso de técnicas de predicción y búsqueda de la asignación óptima. Además, evaluaremos su comportamiento y lo compararemos con las técnicas más comunes de la literatura.

MultiCluster Controller (*MCC*) es el sistema encargado de gestionar los clusters que componen el Multicluster. MCC mantiene una lista actualizada de los clusters que componen el Multicluster gestionado por MetaLoRaS. Para cada cluster mantiene información del estado de sus recursos y el estado de las aplicaciones que se están ejecutando. El MCC proporciona mecanismos de comunicación con los módulos LoRaS de cada cluster. Esto le permite realizar consultas sobre el estado de los recursos y las aplicaciones paralelas y recibir de los módulos LoRaS notificacio-

nes sobre los cambios en el estado tanto de los recursos como de las aplicaciones. Esto permite al MetaScheduler poder determinar en tiempo real la disponibilidad de recursos en el momento de planificar una nueva tarea.

El subsistema *MetaLoRaS* se ha desarrollado completamente durante el presente trabajo. En el Apéndice A se describe con detalle el diseño y la implementación de este subsistema, así como el proceso de interacción entre los dos niveles de planificación, *MetaLoRaS* y *LoRaS*.

2.4. Arquitectura del sistema LoRaS

El subsistema *LoRaS* es el punto de partida de la nueva arquitectura *M-CISNE*. *LoRaS* fue diseñado anteriormente en nuestro grupo de investigación [HGH+05a, HGH+04] como una herramienta de planificación para entornos Cluster no dedicados.

El trabajo realizado en este subsistema se ha dividido en dos fases. En primer lugar se ha dotado a *LoRaS* de una nueva interfaz (***L**oras **I**nteraction **S**ervices*) que facilita el control de la planificación a aplicativos de nivel superior. Esto ha permitido la integración de *LoRaS* como un componente más de la plataforma *M-CISNE*. En segundo lugar, se ha dotado a *LoRaS* de un nuevo mecanismo de predicción capaz de proporcionar estimaciones del tiempo de *turnaround* con desviaciones suficientemente pequeñas.

Los componentes principales del subsistema *LoRaS* (Figura 2.3) son la interfaz con los aplicativos de nivel superior ***Loras Interaction Services*** (*LIS*), el gestor de colas local ***Low-level Queue*** *(LQ*), el planificador ***LoRaS Scheduler*** (*LS*), el módulo de monitorización de los recursos ***Monitoring ToolKit*** (*MT*) y el motor de predicción ***Prediction Engine*** (*PE*).

Loras Interaction Services (*LIS*) es el módulo encargado de recibir y gestionar las peticiones dirigidas al subsistema LoRaS. En nuestro caso las peticiones son realizadas por *MetaLoRaS* a través del *Multicluster Controller*. Las peticiones más comunes son la planificación y/o la estimación del tiempo de turnaround de una aplicación paralela, y la solicitud de información sobre la capacidad de los recursos y el estado de las aplicaciones paralelas. Cuando se produce una petición de planificación de un trabajo paralelo, *LIS* procesa el trabajo recibido y lo entrega al sistema de colas local ***Low-level Queue*** (*LQ*).

LoRaS Scheduler (*LS*) es el módulo encargado de planificar los trabajos paralelos a nivel de cluster. Cuando un nuevo trabajo llega al sistema de colas local (*LQ*) o bien cuando un trabajo en ejecución finaliza, se envía un evento que despierta el proceso

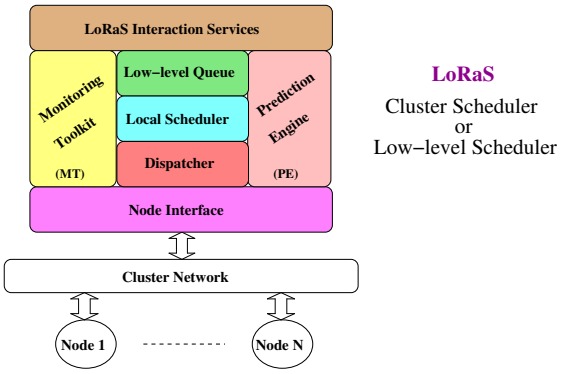


Figura 2.3: Arquitectura LoRaS

de planificación. Entonces el *LoRaS Scheduler* selecciona un nuevo trabajo paralelo de la *Low-level Queue* y le asigna los recursos más adecuados para su ejecución.

Los criterios utilizados durante el proceso de selección de trabajos y durante el proceso de selección de los recursos se definen mediante la *política de planificación*. *LoRaS* permite la definición de múltiples políticas de planificación en función de los objetivos perseguidos por el usuario y/o el administrador del sistema.

Prediction Engine es el encargado de estimar el tiempo de *turnaround* de un trabajo paralelo. Estas estimaciones se llevan a cabo considerando las características propias del trabajo paralelo, el estado actual de los recursos, y la política de planificación utilizada por *LoRaS*.

En el capítulo 3 (sección 3.6), se propone un nuevo mecanismo para llevar a cabo la predicción del tiempo de *turnaround*. Este mecanismo utiliza técnicas de predicción diferentes para la estimación del tiempo de ejecución y del tiempo de espera. Las técnicas de predicción utilizadas consisten en el análisis estadístico de ejecuciones pasadas almacenadas en un repositorio de datos históricos y el uso de un simulador guiado por un nuevo modelo analítico propuesto en el presente trabajo. En la sección 2.4.1 se describen los componentes principales del *Prediction Engine* que han permitido desarrollar las técnicas predictivas propuestas en el presente trabajo.

Las predicciones sobre el tiempo de *turnaround* son útiles en ambos niveles de planificación. *LoRaS* utiliza estas predicciones para determinar cuando un trabajo paralelo puede adelantar a otro en la cola de espera (*LQ*) aplicando la técnica del *backfilling*. *MetaLoRaS* utiliza estas predicciones para seleccionar el cluster más adecuado para ejecutar un determinado trabajo paralelo. *MetaLoRaS* solicita la estimación del tiempo de *turnaround* de los trabajos paralelos a través de *LoRaS Interaction Services*.

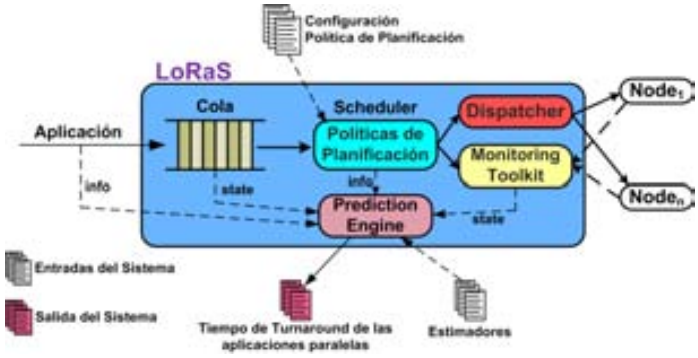


Figura 2.4: Interacción entre *LoRaS* y *Prediction Engine*

Para poder predecir el tiempo de *turnaround* de un trabajo paralelo y llevar a cabo de forma eficaz las tareas de planificación es necesario disponer de información actualizada sobre el estado de ocupación de los recursos, el estado de los trabajos paralelos, la configuración del sistema, etc. El ***Monitoring Toolkit*** es el módulo encargado de monitorizar el estado de los recursos y de las aplicaciones paralelas en ejecución.

La información sobre el estado de los recursos y las aplicaciones paralelas es solicitada directamente por el *LoRaS Scheduler* y por el *Prediction Engine* para las tareas de planificación y predicción respectivamente. *MetaLoRaS* a través de *LoRaS Interaction Services* también tiene acceso a esta información.

Dispatcher es el encargado de ejecutar la aplicación paralela en los nodos seleccionados por LoRaS. Este módulo en función del formato de la aplicación determina el modo de ejecución, soportando tanto aplicaciones MPI como PVM.

2.4.1. Componentes del Prediction Engine (*PE*)

La labor principal del ***P**rediction **E**ngine* es obtener una estimación eficaz del tiempo de turnaround de una aplicación paralela con el mínimo error posible. La estimación del tiempo de turnaround permite a ambos niveles de planificación tomar mejores decisiones aumentando el rendimiento del sistema y de las aplicaciones paralelas.

Para poder realizar las estimaciones con la mayor precisión posible el *Prediction Engine* necesita información sobre el estado actual del sistema, que de ahora en adelante denominaremos *escenario de planificación*.

El *escenario de planificación* está formado por el ***estado de los recursos*** del cluster, la ***naturaleza de las aplicaciones paralelas***, la ***política de planificación*** utilizada y el ***estado de las colas de ejecución y espera*** del sistema. En la Figura 2.4 se observa

el flujo de información que compone el *escenario de planificación* requerido por el *Prediction Engine*.

En el presente trabajo proponemos un mecanismo de predicción del tiempo de *turnaround* basado en la combinación de varios métodos predictivos: *métodos estadísticos* para la estimación del *tiempo de ejecución*, y un motor de simulación guiado por un nuevo *modelo analítico* para la estimación del *tiempo de espera*.

Los componentes principales del *Prediction Engine*, necesarios para implementar los mecanismos propuestos en este trabajo, son el ***motor estadístico*** y el ***motor de simulación***.

En la sección 2.4.1.1 se describe brevemente los componentes fundamentales y la información requerida por el ***motor estadístico*** y en la sección 2.4.1.2 se describen los componentes y la información necesaria para la utilización del ***motor de simulación***.

2.4.1.1. Motor estadístico

El *motor estadístico* (*Predictior Engine*) permite obtener estimaciones del tiempo de ejecución y/o espera de un aplicación paralela realizando un análisis estadístico sobre la información de ejecuciones pasadas almacenadas en un repositorio de datos.

La idea básica de este método predictivo es que aplicaciones similares tienen tiempos de ejecución similares. Para obtener una nueva estimación, el *motor estadístico* trata de buscar en un *repositorio de datos,* ejecuciones pasadas de aplicaciones iguales o similares en un estado del sistema lo más parecido posible.

Los componentes principales del *motor estadístico* (Figura 2.5) son el ***analizador estadístico*** y el ***repositorio de datos históricos***.

El ***analizador estadístico*** es el encargado de buscar en el *repositorio de datos* ejecuciones pasadas similares y analizar estadísticamente los datos obtenidos para obtener una estimación del tiempo de ejecución y/o del tiempo de espera. Para poder llevar a cabo las estimaciones debemos configurar el **método de búsqueda** de ejecuciones pasadas similares y el **método de análisis estadístico** que debe utilizar el *analizador estadístico*.

El ***repositorio de datos históricos*** almacena información sobre las aplicaciones paralelas ejecutadas y el estado del sistema durante su ejecución. La información almacenada debe mantenerse actualizada para disponer de una representación lo más precisa posible de la realidad y realizar predicciones con el mínimo error. Por este motivo *LoRaS* en cada nuevo evento de planificación mantiene actualizada la información del *repositorio de datos*.

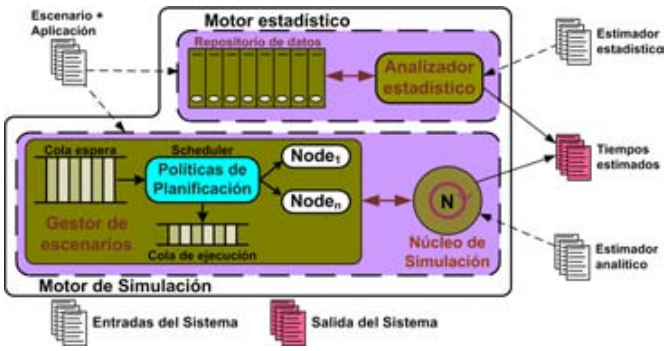


Figura 2.5: Componentes del *Prediction Engine*

La información que almacena el *repositorio de datos* tiene que ver con el ***estado de ocupación de los recursos*** *del cluster,* la ***configuración del planificador***, el ***estado de las colas*** *de ejecución y espera* del sistema, y la información sobre las ***aplicaciones paralelas*** durante su ejecución. En el Apéndice B se describe con detalle el diseño del repositorio de datos históricos y el modo en que se almacena y se relaciona toda esta información.

2.4.1.2. Motor de Simulación

El *motor de simulación* permite estimar el tiempo de ejecución y/o espera de una aplicación paralela recién llegada al sistema, simulando la ejecución de todas las aplicaciones paralelas que están por delante en la cola de espera.

Los componentes principales del *motor de simulación* (Figura 2.5) son el ***gestor de escenarios*** y el ***núcleo de simulación***.

Para poder simular la ejecución de las aplicaciones paralelas necesitamos de un modelo analítico que nos permita evaluar el tiempo de ejecución de una aplicación paralela considerando las características propias de la aplicación y el estado del sistema. Aunque existen gran cantidad de modelos en la literatura, muchos de ellos consideran que los recursos son homogéneos y otros solo tienen en cuenta la capacidad de los recursos de procesamiento sin considerar los recursos de comunicaciones. En el capítulo 3 presentamos un nuevo modelo analítico para entornos heterogéneos que considera la capacidad de los recursos tanto de cómputo como de comunicaciones.

El modelo analítico propuesto es utilizado por el ***núcleo de simulación*** para guiar el proceso de simulación estimando el tiempo remanente de ejecución de las aplicaciones paralelas en cada nuevo paso de simulación. El *núcleo de simulación* se

puede configurar para utilizar distintos modelos analíticos. En el capítulo 3 se realiza un estudio comparativo de los resultados obtenidos utilizando distintos modelos analíticos.

Para poder simular la ejecución de las aplicaciones paralelas que se encuentran en el sistema, el *núcleo de simulación* necesita disponer de cierta información acerca del estado de *ocupación de los recursos*, el *estado de las colas de ejecución y espera*, la *política de planificación* y ciertas características sobre la *naturaleza de las aplicaciones paralelas*. Además, el *núcleo de simulación* necesita alterar el estado de los recursos y de las colas del sistema durante el proceso de simulación. Para llevar a cabo el proceso de simulación sin interferir en el proceso de planificación real, el ***gestor de escenarios*** dispone de una reproducción de las unidades funcionales básicas del sistema de planificación real, el sistema de colas de trabajos en espera, la cola de trabajos en ejecución, la política de planificación y los recursos del sistema.

Cuando un nueva solicitud de estimación llega al motor de simulación, el *gestor de escenarios* recopila información sobre el escenario de planificación real y lo reproduce en sus estructuras internas. En la sección 3.4.2 se describe con detalle el procedimiento del motor de simulación.

2.5. Planificación en un entorno Multicluster

En esta sección se describen las operaciones básicas de la tarea de planificación en cada uno de los niveles. En cada fase de la tarea de planificación se indican los componentes que intervienen y la información de configuración que se necesita.

2.5.1. Meta-planificación

Cuando un usuario desea lanzar una nueva aplicación paralela lo hace a través de una interfaz definida para tal efecto. A través de esta interfaz se proporciona la información necesaria sobre la aplicación paralela que se desea lanzar. La información que se debe especificar hace referencia a la localización y los parámetros necesarios para lanzar la aplicación (directorio, ejecutable, parámetros, etc.), al tipo de aplicación (librería de paso de mensajes que utiliza) y los recursos que consume.

En la Figura 2.6 se muestra un esquema con las distintas fases de la meta-planificación, los componentes que intervienen en cada fase y la información necesaria para su configuración.

Como se observa en la Figura 2.6 la meta-política define las técnicas que el MetaScheduler va a utilizar en cada una de las fases. MetaLoRaS permite configurar

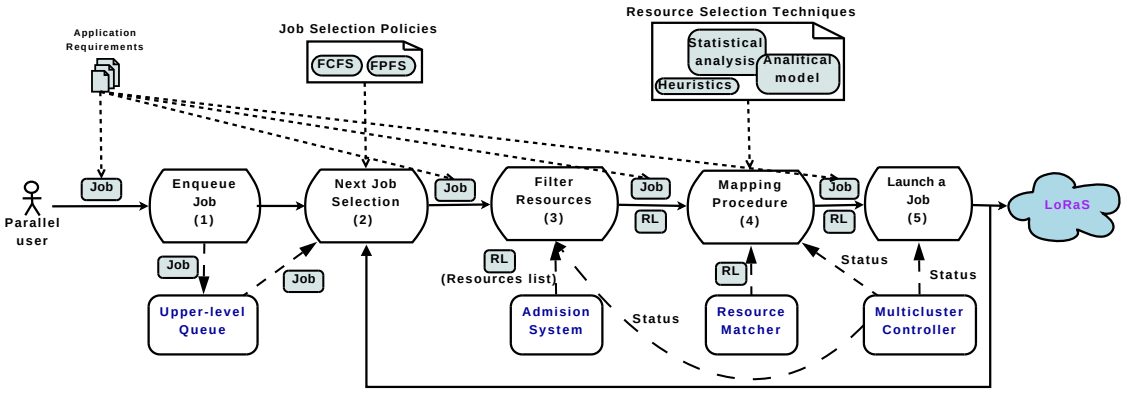


Figura 2.6: Esquema de meta-planificación (*MetaLoRaS*)

múltiples meta-políticas y seleccionar aquella que deseamos utilizar en cada experimento.

Una vez definida la aplicación paralela se procede a la fase **(1)** de inserción en la cola de espera. El sistema de colas, *Upper-level Queue*, se encarga de introducir la nueva aplicación en la cola de espera correspondiente. El *Upper-level Queue* permite la definición de múltiples colas gestionadas por distintas meta-políticas de planificación. En cada cola se definen los criterios que deben cumplir las aplicaciones que va a gestionar. Cuando llega una nueva aplicación paralela es clasificada según los criterios definidos en cada cola y se queda en espera de ser planificada.

Cuando una nueva aplicación paralela llega al sistema de colas o bien cuando finaliza su ejecución se produce un evento de planificación. A la llegada de un nuevo evento de planificación, el *MetaScheduler* activa una nueva etapa de meta-planificación. Una etapa de meta-planificación consiste principalmente en las fases siguientes: **(2)** selección del siguiente trabajo paralelo a ser ejecutado (*Next Job Selection*), **(3)** filtrado de los recursos válidos para ejecutar el trabajo seleccionado (*Filter Resources*), **(4)** asignación de los mejores recursos para ejecutar el trabajo (*Mapping Procedure*), **(5)** y finalmente la entrega del trabajo paralelo al cluster seleccionado (*Launch a Job*). Una vez el trabajo paralelo es entregado correctamente a los recursos asignados el *MetaScheduler* vuelve a la fase (2) en busca de un nuevo trabajo paralelo.

En la fase de selección del trabajo paralelo **(2)** el *MetaScheduler* selecciona del sistema de colas, ***Upper-level Queue***, el siguiente trabajo paralelo a ser ejecutado según la política de selección de trabajos (***J**ob **S**election **P**olicy*) con que se ha configurado la meta-política de planificación. *MetaLoRaS* permite la implementación y configuración de distintas políticas de selección de trabajos. En el presente tra-

bajo estamos interesados en el uso de las políticas más utilizadas en el ámbito del Multicluster *FCFS*, *FPFS* y *Backfilling*.

La técnica más popular utilizada tanto en entornos Cluster [LLM88, Fei97, AKN98] como Multicluster [BE03a, BE03b, BBE03] es la ***F**irst **C**ome **F**irst **S**erved* (FCFS). Sin embargo, con esta técnica se produce la infrautilización de los recursos [HSSY00] cuando el número de procesadores disponibles es inferior a los procesadores solicitados por el primer trabajo en la cola de espera, ya que no se planifica ningún otro trabajo hasta que los requisitos del primero se han satisfecho. Varias técnicas han surgido para evitar los inconvenientes de la FCFS, entre las más populares se encuentran la *FPFS* [SME04, JLPS05, BE03a, BE03b] y el *Backfilling* [HSSY00, Yue04, TEF07].

Cuando no existen suficientes recursos para ejecutar el primer trabajo en la cola de espera, la técnica ***F**it **P**rocessors **F**irst **S**erved* (*FPFS*) recorre la cola de principio a fin buscando trabajos que puedan ser ejecutados con los recursos disponibles. Cuando no existen suficientes recursos libres para ejecutar el primer trabajo en la cola de espera, en la técnica de ***Backfilling***, el planificador puede adelantar la ejecución de otros trabajos siempre que este adelanto no suponga un retraso en el inicio de la ejecución del primer trabajo en la cola. Ambas técnicas permiten aumentar la utilización de los recursos adelantando las aplicaciones paralelas.

Una vez seleccionado el trabajo paralelo a ser planificado, se procede a la fase de filtrado **(3)** de los recursos válidos. El ***Admision System*** obtiene las características de los recursos de cada cluster a través del ***Multicluster Controller*** y filtra aquellos recursos que por sus características, sistema operativo, librería de paso de mensajes, capacidad máxima de los recursos, etc, no pueden ejecutar el trabajo paralelo seleccionado. El resultado de este filtrado es una lista de recursos válidos para ejecutar el trabajo paralelo. Cuando no existen suficientes recursos válidos el trabajo paralelo es descartado.

Cuando existen suficientes recursos válidos es el momento de decidir que recursos son los más adecuados para ejecutar el trabajo paralelo. Esta decisión se toma en la fase de Mapping **(4)** de la tarea de planificación. El ***Resource Matcher*** actualiza el estado de los recursos a través del ***Multicluster Controller*** y asigna los recursos al trabajo paralelo según la técnica de selección de recursos (***R**esource **S**election **P**olicy*) con que se ha configurado la meta-política de planificación.

MetaLoRaS permite la implementación de distintas técnicas de selección de recursos (***R**esource **S**election **P**olicy*) para su asignación a los trabajos paralelos. Estas técnicas las podemos dividir en dos grupos, técnicas de *selección de cluster*, que consideran un particionamiento de los recursos en clusters y seleccionan el cluster más adecuado para ejecutar el trabajo paralelo, y técnicas de *co-asignación*, que

permiten la compartición de recursos entre distintos clusters y seleccionan el conjunto de recursos más adecuado para ejecutar el trabajo paralelo (*co-asignación*).

En ambos casos la cuestión está en cómo realizar una valoración de las distintas opciones que permita determinar la selección más adecuada cumpliendo con los objetivos de rendimiento del sistema. En nuestro caso maximizar el rendimiento de las aplicaciones paralelas sin perjudicar al usuario local.

Las aportaciones realizadas en el presente trabajo se han centrado en la propuesta de nuevas técnicas de selección basadas en *modelos de rendimiento*. Concretamente se han propuestos dos soluciones para la *selección del cluster* más adecuado basadas en la predicción del rendimiento del trabajo paralelo (sección 4.3) y un técnica de *co-asignación*, que guiada por un nuevo modelo del tiempo de ejecución de una trabajo paralelo trata de buscar la mejor asignación posible (sección 4.6).

En el capítulo 1 (sección 1.2.3), descubrimos que se debe limitar el uso de la *co-asignación* para evitar la saturación de los canales de interconexión entre clusters. En el presente trabajo asumimos que los trabajos paralelos son asignados a nivel de cluster siempre que esto sea posible. Entonces, durante la fase de Mapping (4), el ***Resource Matcher*** busca el cluster que le permite ejecutar la aplicación paralela en el menor tiempo posible mediante el uso de métodos predictivos.

En los casos en que no existan recursos suficientes en ningún cluster para ejecutar un trabajo paralelo se aplicará la co-asignación. En el presente trabajo proponemos además la *co-asignación* de aquellos trabajos paralelos que han sido asignados a algún cluster y se encuentran a la espera de recursos libres. Este procedimiento de *co-asignación* se realiza únicamente cuando no existen trabajos paralelos en las colas de espera del *MetaLoRaS*. El objetivo principal de esta propuesta es reducir los tiempos de espera de los trabajos paralelos y aumentar la utilización de los recursos aprovechando los recursos disponibles de varios clusters (véase la sección 4.6).

Una vez el ***Resource Matcher*** ha decidido los recursos más adecuados para la ejecución del trabajo paralelo se procede a la fase de lanzamiento **(5)** del trabajo paralelo al cluster encargado de su ejecución. En esta fase el *MetaScheduler* entrega el trabajo paralelo al cluster correspondiente a través del ***Multicluster Controller***. Cuando un trabajo paralelo es co-asignado el ***Multicluster Controller*** crea una instancia virtual de un nuevo cluster que controla los recursos asignados al trabajo paralelo. Esta nueva instancia virtual controla la ejecución de la aplicación y comunica a los clusters con los que comparte los recursos la existencia de un nuevo trabajo paralelo. Una vez finalizada la ejecución del trabajo paralelo la instancia virtual es eliminada del ***Multicluster Controller***.

En la sección 2.5.2 se explica el proceso de planificación de un trabajo paralelo una vez entregado al cluster correspondiente.

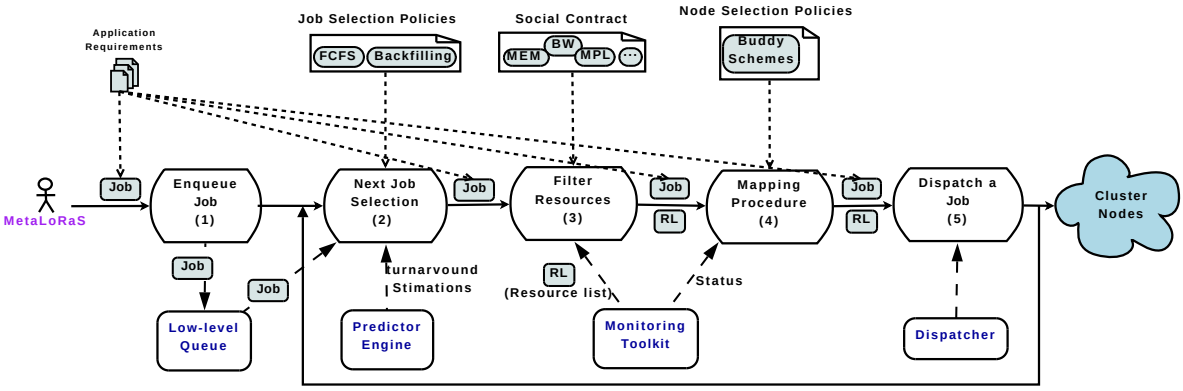


Figura 2.7: Esquema de planificación LoRaS

2.5.2. Planificación a nivel de cluster

Cuando un nuevo trabajo paralelo es asignado a un cluster del Multicluster, el subsistema LoRaS es el encargado de planificar, controlar su ejecución y notificar los cambios de estado del trabajo paralelo al MetaLoRaS.

En la Figura 2.7 se observan las distintas fases de planificación a nivel de LoRaS, los componentes que intervienen en cada fase y la información necesaria para su configuración.

Una política de planificación define las técnicas que el *Scheduler* de LoRaS aplica en cada una de las fases de planificación. LoRaS permite definir múltiples políticas de planificación y seleccionar aquella que se quiere utilizar en cada cluster.

Las distintas fases de planificación a nivel de LoRaS son las siguientes: **(1)** encolamiento del trabajo paralelo en el ***L**ow-level **Q**ueue **S**ystem*, **(2)** selección del siguiente trabajo paralelo a ejecutar, **(3)** filtrado de los recursos, **(4)** asignación de recursos al trabajo paralelo y **(5)** ejecución del trabajo paralelo.

Cuando un trabajo paralelo llega al cluster, se inicia la fase de encolamiento **(1)** donde el ***L**ow-level **Q**ueue **S**ystem* almacena el trabajo paralelo en una de las colas de espera. A continuación, se procede a la fase de selección del siguiente trabajo paralelo **(2)**. En esta fase *LoRaS* selecciona el siguiente trabajo paralelo a ejecutar según la política de selección de trabajos que se ha seleccionado al configurar la política de planificación.

En el presente trabajo se han implementado las políticas de selección de trabajos más utilizadas comúnmente en la literatura *FCFS* [LLM88, Fei97, Gib97, AKN98] y *Backfilling* [Lif95, MF01, TEF07, AFKT00].

Una vez seleccionado el trabajo paralelo se procede a la fase de filtrado **(3)** de los recursos. El filtrado consiste en utilizar únicamente aquellos recursos donde la

ejecución del nuevo trabajo paralelo no perjudique al confort del usuario local. Para realizar el filtrado el administrador del sistema define un contrato social [ADV+95, HGH+05b]. El contrato social establece los recursos máximos que las aplicaciones paralelas pueden utilizar (memoria, grado de paralelismo, ancho de banda, etc). Para más información acerca del contrato social, léase [GSHL03].

El Scheduler obtiene el estado de los recursos a través del **Monitoring Toolkit** y se filtran los recursos que considerando los requisitos del trabajo paralelo no cumplen con el contrato social. El proceso de filtrado devuelve una lista de nodos con suficientes recursos para la ejecución del trabajo paralelo sin perjudicar el confort del usuario local.

En aquellos casos en que no existen suficientes recursos disponibles para lanzar el trabajo paralelo, se mantiene el trabajo en la cola de espera y se vuelve a la fase (2) donde se comprueba si existen otros trabajos que se puedan ejecutar según la política de selección de trabajos escogida.

Cuando existen suficientes recursos para la ejecución del trabajo paralelo, se decide los recursos más adecuados para su ejecución en la fase de Mapping **(4)**. El *Scheduler* a través del **Monitoring Toolkit** obtiene el estado de los recursos y los ordena en función de su disponibilidad (memoria libre, grado de paralelismo, ancho de banda, etc). Como resultado de esta fase se devuelve el conjunto de recursos con mayor disponibilidad para la ejecución del trabajo paralelo.

Una vez seleccionados los recursos se procede al lanzamiento del trabajo paralelo. En la fase de lanzamiento **(5)** el **Dispatcher** determina el tipo de librería de paso de mensajes que utiliza el trabajo paralelo y utiliza los mecanismos de lanzamiento adecuados según se trate de un trabajo PVM o MPI.

A continuación, el *Scheduler* vuelve a la fase (2) en busca de un nuevo trabajo paralelo. Cuando un trabajo paralelo finaliza, el Scheduler almacena en el repositorio de datos toda la información referente al trabajo paralelo, el estado de las colas, el estado de los recursos utilizados, y los tiempos de espera y ejecución tanto reales como estimados.

En el Apéndice A se describe con detalle el diseño y la implementación del subsistema LoRaS, así como el proceso de interacción con el planificador de nivel superior *(MetaLoRaS)*.

Capítulo 3

Sistema de Predicción

La predicción aplicada a entornos cluster ha sido un campo ampliamente estudiado en la década de los noventa y continúa ganando importancia en la presente década, aplicándose en entornos Multicluster, Grid, etc. En la actualidad, muchos de los sistemas comerciales se basan en el uso de mecanismos de predicción [WSH99, BAG00, RI01, YSF03, Wol03]. El motivo principal de este interés por la predicción son las grandes ventajas que ofrece poder considerar el estado futuro de un sistema o el rendimiento de una aplicación, en el desarrollo de nuevas y mejores políticas de planificación.

Conocer el estado futuro del sistema o el comportamiento de un aplicación con la utilización de determinados recursos permite tomar decisiones de planificación que se ajustan mejor a los objetivos que cada sistema tiene marcados, coste, rendimiento, utilización, etc. Cuando la precisión en las estimaciones es suficientemente grande, el sistema de planificación puede incluso establecer mecanismos de QoS en donde se establecen ciertos rangos de rendimiento que se deben garantizar, tiempo máximo de ejecución de una aplicación, grado de utilización de determinados recursos, no saturación, etc. Estas garantías permiten a los administradores establecer modelos económicos en donde el uso de determinados recursos y/o la definición de determinados límites llevan un coste asociado [CC00, AAB$^+$00, ABG02, SAL$^+$04].

Otra ventaja importante que aporta el uso de la predicción es la información que ofrece al usuario. Esta información puede ser útil para decidir de qué modo y con qué recursos se desea ejecutar una determinada aplicación, pero también para conocer ciertas características, desconocidas por el propio usuario, sobre el comportamiento de la aplicación en determinadas condiciones (utilización de recursos, eficiencia y escalabilidad de la aplicación, etc).

En el presente trabajo estamos interesados en minimizar el tiempo que una aplicación paralela permanece en el sistema. Este tiempo lo denominamos ***turnaround***

time y lo definimos como la suma del *tiempo de espera* en las colas del sistema y el *tiempo de ejecución* (*wall-clock time*). Para ello se ha prestado especial atención en al aumento de la precisión en la estimación de ambas métricas y en la propuesta de nuevos mecanismos para que esto sea posible, de modo que el planificador pueda tomar mejores decisiones aumentando así el rendimiento de las aplicaciones y del sistema en general.

La tarea de predicción en un entorno Multicluster no dedicado es un gran desafío. En primer lugar existe una gran cantidad de recursos con características de cómputo y comunicación muy diversas. En segundo lugar, los recursos pueden estar compartidos con otros usuarios. Por otro lado, el volumen de información en los canales de interconexión entre clusters puede crecer muy rápidamente convirtiéndose en un cuello de botella del sistema. Todas estas características se deben tener en cuenta para llevar a cabo una correcta estimación y en consecuencia una correcta asignación de las aplicaciones a los recursos. En el presente capítulo proponemos nuevos métodos de predicción que tratan todas estas consideraciones.

En un entorno Multicluster donde los planificadores locales disponen de colas de espera y políticas de planificación propias, es tan importante poder estimar tanto el tiempo de ejecución como el tiempo de espera en las colas de las aplicaciones paralelas. En las últimas décadas se han realizado grandes esfuerzos en predecir el tiempo que una aplicación invierte en su ejecución [Gib97, SB98, WSH99, SFT98, IOP99, Din01], prestando menor atención a las predicciones de los tiempos de espera en las colas del sistema [Dow97, SW02, LGTW04]. En un trabajo anterior [LSG+08] observamos grandes diferencias de precisión en la estimación del tiempo de espera y el tiempo de ejecución, y demostramos que mejorando la precisión en la estimación del tiempo de espera pueden obtenerse mejoras significativas de precisión en la estimación del tiempo de *turnaround*.

En el presente capítulo presentamos dos nuevos métodos predictivos cuya principal aportación está en considerar el estado actual del sistema, tanto la ocupación de las colas como la capacidad de los recursos de cómputo y comunicación. Analizamos los resultados obtenidos en la precisión de las estimaciones del tiempo de espera y de ejecución, y los comparamos con otras propuestas de la literatura. Finalmente, proponemos un nuevo mecanismo de predicción del tiempo de *turnaround* de una aplicación paralela basado en una combinación de las técnicas que obtienen los mejores resultados.

En la sección 3.1 se hace una breve introducción a los métodos predictivos más utilizados en la literatura y sus características principales. En la sección 3.2.1 describimos el funcionamiento de un método estadístico utilizado comúnmente en la literatura que utilizaremos para realizar las comparaciones. En la sección 3.2.2 se propone un nuevo método predictivo basado en el análisis estadístico denominado

***I**nstance-**B**ased **L**earning* (*IBL*). En la sección 3.3.1 proponemos un nuevo modelo analítico y describimos como se integra en un motor de simulación para obtener las predicciones. En la sección 3.5 se analiza el comportamiento de los métodos predictivos propuestos para distintas situaciones de carga y políticas de planificación, y se comparan los resultados obtenidos con otros métodos comunes en la literatura. En la sección 3.6, se propone un nuevo mecanismo para la estimación del tiempo de *turnaround* y se evalúan los resultados de precisión obtenidos.

3.1. Introducción a los métodos predictivos

Los métodos de predicción más utilizados en la literatura los podemos dividir en ***estadísticos*** [Dow97, Gib97, WSH00, Din01, SW02, LS05, YA07], los cuales utilizan algún tipo de análisis estadístico sobre las aplicaciones que ya han sido ejecutadas, y ***analíticos*** [SB98, JA05, HGH$^+$06, JSK$^+$06], métodos que construyen ecuaciones que describen el modelo de ejecución de las aplicaciones.

Los métodos estadísticos se pueden clasificar en el análisis de *series temporales* [DI89, Din99, WSH00, Din01], métodos de *categorización* de las aplicaciones [Dow97, Gib97, SFT98], o métodos de *aprendizaje basados en casos* [KFB99, IOP99, SW02], denominados *IBL* (***I**nstance-**B**ased **L**earning*).

El análisis de ***series temporales*** es muy útil para predecir la disponibilidad de los recursos en el tiempo. Esta técnica es utilizada en muchos estudios previos [Din01, Wol03] para determinar el rendimiento de las aplicaciones paralelas en función de la disponibilidad estimada de los recursos. Aunque este método va en la misma línea que nuestra investigación, en el presente trabajo nos hemos centrado en los métodos estadísticos más utilizados para la predicción del tiempo de espera y ejecución de una aplicación paralela, la categorización y el aprendizaje basado en casos (*IBL*).

Los ***métodos de categorización*** y ***aprendizaje basado en casos*** (*IBL*) llevan a cabo la predicción del tiempo de ejecución de una aplicación paralela basándose en la historia pasada. Estos métodos, y en especial el de categorización, han sido muy utilizados para la predicción de los tiempos de ejecución, obteniendo en general buenos resultados. Sin embargo, no se han mostrado tan eficaces en la predicción de los tiempos de espera.

Los ***métodos analíticos*** devuelven una medida del tiempo de ejecución bajo determinadas condiciones de estado de los recursos. Aunque estos modelos son difíciles de desarrollar y precisan de mayor información, tienen un bajo coste computacional, proporcionan una gran información sobre el comportamiento de las aplicaciones y pueden llegar a obtener muy buenos resultados. En general, los métodos analíticos

se desarrollan manualmente [SB98, JA05, HGH+06, JSK+06] o bien con la ayuda de herramientas de análisis automático de código o instrumentación [TWG+01].

En este capítulo proponemos dos técnicas de estimación, una técnica estadística basada en el aprendizaje de casos (*IBL*) y un nuevo modelo analítico. Analizamos el rendimiento de ambas técnicas y lo comparamos con otras técnicas de la literatura.

En la sección 3.2.1 presentamos el funcionamiento básico de un método estadístico de categorización utilizado frecuentemente en la literatura con el fin de realizar las comparaciones. En la sección 3.2.2 proponemos nuestro método estadístico basado en el aprendizaje de casos y en la sección 3.3.1 proponemos un nuevo modelo analítico.

3.2. Métodos estadísticos

Los métodos estadísticos utilizados comúnmente en la predicción de los tiempos de espera y ejecución de las aplicaciones paralelas los podemos dividir en métodos de ***categorización*** y métodos de ***aprendizaje basados en casos*** o *Instance-based learning*. La diferencia principal entre ambos métodos es la información que se almacena en la base de datos históricos y el método de búsqueda de ejecuciones pasadas similares a la situación actual que se quiere predecir.

Cuando en la base de datos históricos se almacena únicamente información sobre la aplicación paralela (Usuario, Cola de espera, Ejecutable, número de Nodos, etc), nos encontramos ante un método de ***categorización*** [Dow97, Gib97, SFT98]. La categorización se basa en la idea de que aplicaciones similares tienen tiempos de ejecución similares. La información de la base de datos históricos se divide en categorías como por ejemplo <*Usuario, Cola*>, <*Usuario, Cola, Ejecutable*> o bien <*Usuario, Ejecutable*>, <*Usuario, Ejecutable, Nodos*>, etc. Cuando una nueva aplicación llega al sistema se busca la categoría en que se encuentra y se devuelve una estimación del tiempo de ejecución basándose en un análisis estadístico de los datos de la misma categoría.

Cuando la base de datos históricos además de almacenar información sobre la aplicación almacena información sobre el estado de los recursos, nos encontramos ante un método de predicción basado en el ***aprendizaje de casos*** [KFB99, IOP99, SW02], ***IBL***. Mediante esta técnica, la base de datos almacena un conjunto de experiencias. Una experiencia consiste en un conjunto de información de la aplicación más algún tipo de información sobre el estado de los recursos. Cuando una nueva aplicación llega al sistema, se utilizan métricas para medir la distancia de similitud y se predice el tiempo de ejecución aplicando un análisis estadístico al conjunto de

experiencias pasadas más cercanas al estado actual. La precisión en la estimación dependerá del número de experiencias registradas y del grado de similitud.

En la sección 3.2.1 presentamos un método de categorización muy utilizado en la literatura con el que compararemos los resultados obtenidos con nuestras propuestas estadística y analítica. En la sección 3.2.2 presentamos nuestro método de aprendizaje basado en casos (*IBL*).

3.2.1. Categorización

La categorización es un método utilizado frecuentemente para la predicción del tiempo de ejecución de una aplicación [SFT98, Dow97, Gib97]. Este método se basa en la idea de que aplicaciones parecidas tienen mayor probabilidad de tener tiempos similares que aplicaciones que no tienen nada en común.

La categorización utiliza las características de las aplicaciones para definir distintas categorías. Cuando se solicita una nueva estimación se busca información de ejecuciones pasadas de aplicaciones dentro de la misma categoría, en la base de datos históricos. Es decir, ejecuciones pasadas de aplicaciones con características similares. A continuación se realiza un análisis estadístico con los datos de la misma categoría para obtener una estimación.

Uno de los inconvenientes principales en el desarrollo de técnicas de predicción basadas en la similitud es que existen multitud de formas distintas de comparar dos aplicaciones. Podemos comparar aplicaciones según el nombre de la aplicación (E), el nombre del usuario que la lanza (U), los argumentos de la aplicación (A), la cola de espera que administra el trabajo (C), el momento en que la aplicación llega al sistema(T), el número de nodos requeridos (N), etc. En el presente trabajo hemos utilizado aquellas características de la aplicación que se vienen utilizando en la literatura y que corresponden con los parámetros registrados en los ficheros de traza de conocidos centros de supercomputación como por ejemplo, Argone National Laboratory (ANL), Cornell Theory Center (CTC), San Diego Supercomputer Center (SDSC), etc. Estos archivos de traza tienen en común un conjunto reducido de características de las aplicaciones paralelas como son, el nombre de usuario, el nombre de aplicación, el nombre de la cola de espera, etc.

Gibbons propone en [Gib97] que la mejor solución es categorizar las aplicaciones según el patrón (*Ejecutable, User, Nodos*). Smith en [SFT98] propone varios algoritmos de búsqueda para obtener las combinaciones de características (*patrones*) más representativas para definir las categorías. Los algoritmos se aplican sobre ficheros de traza de distintos sistemas y se obtiene que los patrones con los mejores resultados son (*User, Ejecutable, Nodos*) y (*Cola, User, Nodos*). Finalmente, Li en

[LGTW04] concluye que los mejores resultados los obtiene para los patrones (*User, Cola, Ejecutable, Nodos*). En el presente trabajo, con el objetivo de poder llevar a cabo una comparación significativa con los patrones de la literatura utilizaremos el patrón (User, Ejecutable, Nodos) propuesto por *Gibbons* y *Smith* y el patrón (User, Cola, Ejecutable, Nodos) propuesto por *Li.*

Una vez definido el patrón de categorización necesitamos definir la *función estadística* (f_a) para el cálculo de la estimación y el número *máximo de experiencias pasadas* α a tener en cuenta en el cálculo de la estimación. En el presente trabajo dado que el número de experiencias no es muy grande y que el motor de la base de datos resuelve las consultas en tiempos del orden de milisegundos, hemos optado por no limitar el número máximo de experiencias. No obstante se ha implementado un parámetro ajustable α que permite definir el grado máximo de experiencias en los casos que se considere necesario. En un trabajo futuro seria interesante estudiar como determinar el valor más adecuado para el parámetro α en función de la naturaleza de las experiencias y las desviaciones obtenidas en la predicción.

La *función estadística* determina el método estadístico que se utiliza para llevar a cabo la predicción. Los métodos estadísticos más utilizados en la literatura son la media, la regresión lineal, la regresión logarítmica, etc. Smith en [STF99] realiza pruebas de precisión con distintos predictores y concluye que la media es el mejor predictor. En el presente trabajo aunque definiremos el *tipo de predicción* como un parámetro ajustable nos basaremos en la *media* como predictor $(f_a = media)$.

3.2.2. Instance-based Learning

En el presente trabajo proponemos una técnica de aprendizaje basado en casos para predecir el tiempo de ejecución y/o espera de una aplicación paralela.

El aprendizaje basado en casos [AMS97] requiere de una base de datos donde se almacenan experiencias o casos y que denominaremos **base de experiencias**. Una experiencia se compone de un vector de atributos de entrada y un vector de salida. Los atributos de entrada describen las condiciones del sistema bajo las que se observa la experiencia y los de salida describen que sucede bajo estas condiciones. Cada atributo consiste en un nombre y un valor que puede ser entero, real o una cadena de caracteres. En el presente trabajo proponemos utilizar como atributos de entrada los mismos utilizados en el método de la categorización pero añadiendo información sobre el estado de las colas de espera. Así pues, el vector de entrada que proponemos estará formado por los atributos siguientes: usuario, cola de espera, nombre de la aplicación, número de nodos, numero de *jiffies* en la cola de espera y número de trabajos en la cola de espera. El tiempo de espera y el tiempo de ejecución de la aplicación paralela forman los atributos del vector de salida de la experiencia.

Cuando el motor de predicción solicita una estimación al motor estadístico, éste realiza una consulta a la base de experiencias (*repositorio de datos*) utilizando la información de la aplicación paralela y el estado actual del sistema como vector de entrada de la experiencia a consultar. Entonces, el motor estadístico examina la base de experiencias midiendo mediante funciones de distancia, la cercanía al nuevo vector de entrada y utilizando las experiencias más cercanas para estimar el nuevo vector de salida.

Existen en la literatura distintas métricas de distancia [WM97] entre las cuales hemos escogido la denominada *Heterogenous Euclidean-Overlap Metric* (HEOM), ver el apéndice C. Esta métrica permite medir la distancia entre dos elementos descritos por una combinación de atributos lineales (números) y nominales (cadenas de caracteres). En nuestro caso, el nombre de usuario, la cola de espera y el nombre de la aplicación son atributos nominales. El número de nodos requeridos es un atributo discreto, y el total de jiffies y el número de trabajos en la cola de espera forman los atributos continuos.

Una vez calculadas las distancias entre el vector de entrada y las experiencias de la base de datos debemos determinar cómo calcular las estimaciones de los atributos de salida. Para ello debemos definir el tipo de función estadística (f_a) que deseamos aplicar sobre el vector de salida y el radio de cercanía (r_a) de las experiencias que deseamos considerar en el cálculo de la función estadística.

En [SFT98, LGTW04, Smi04] se demuestra que la función estadística con la que en general se obtienen mejores resultados es la *media* y se apunta que es una buena práctica limitar el número de experiencias utilizadas a la hora de calcular la *media*. Limitar el número de experiencias permite, en primer lugar, aumentar la precisión en las estimaciones ya que cuanto mayor sea el número de experiencias, mayor es la probabilidad de alejarse de las experiencias más cercanas, y en segundo lugar, permite reducir el tiempo de acceso al repositorio de experiencias y el tiempo de cálculo de las estimaciones.

En [SFT98, LGTW04], la búsqueda del tamaño de ventana más adecuado se realiza de forma empírica. En [Smi04] se demuestra que el límite adecuado de experiencias depende de las aplicaciones que se ejecutan y del sistema donde se ejecutan. Para poder encontrar este valor proponen una búsqueda mediante algoritmos genéticos sobre la base de experiencias.

En el presente trabajo se ha definido la *media* como función estadística $(f_a = media)$ con un un radio de cercanía $(r_a = 0)$ y sin limite en el número de experiencias. Tras varios experimentos se ha demostrado que estos valores son suficientes dado que el conjunto de experiencias en la base de datos no es muy elevado y las estimaciones se resuelven de forma rápida (*milisegundos*).

3.3. Método analítico

Los modelos analíticos permiten capturar mediante formulación matemática el rendimiento de una aplicación en un sistema bajo determinadas condiciones del entorno. Aunque el desarrollo de estos modelos y su ajuste es una tarea compleja resultan muy útiles en la obtención de información sobre el comportamiento de las aplicaciones y/o sistemas así como de su posible evolución.

Durante la ejecución de una aplicación paralela, debido a la compartición de los recursos con otras aplicaciones, la disponibilidad de los recursos de cómputo y comunicación es variable. Los métodos estadísticos asumen que el estado permanece estable durante el tiempo de ejecución y esto puede perjudicar a la precisión en la predicción. La posibilidad de integrar el modelo analítico en un motor de simulación permite capturar el comportamiento dinámico del sistema durante la ejecución de las aplicaciones paralelas.

Por lo general, los trabajos anteriores basados en la estimación del tiempo de ejecución [BAG00, Wol03, HHL+06] sólo tienen en cuenta el estado de los recursos de cómputo, sin considerar la ocupación de los canales de comunicación. En [DZ97, BE03b, JA05] se presentan distintos *modelos de comunicación* que nos permiten valorar el rendimiento del sistema asumiendo valores predeterminados de tamaño de mensaje y latencias, pero estos modelos no consideran los recursos de cómputo y no son útiles para estimar el tiempo total de comunicación de cualquier aplicación paralela que llega al sistema.

En [BE01, BBE03, EHS+02] analizan mediante simulación el efecto de las comunicaciones sobre el rendimiento de determinadas aplicaciones, pero no establecen un método generalista válido. Jones en [JLPS05] presenta un modelo que considera los requisitos de ancho de banda de las aplicaciones para medir el impacto de la saturación en el tiempo de ejecución de las aplicaciones paralelas. Sin embargo, este modelo asume un entorno homogéneo y dedicado. Hanzich en [HGH+06] presenta un *modelo analítico de cómputo* para estimar el tiempo de ejecución de un trabajo paralelo, considerando únicamente el número de trabajos paralelos ejecutándose concurrentemente. A pesar de los buenos resultados obtenidos tampoco se consideran los recursos de comunicación.

En esta sección proponemos un nuevo modelo analítico que permite expresar el tiempo de ejecución de una aplicación paralela considerando la heterogeneidad tanto de los recursos de cómputo como de comunicación.

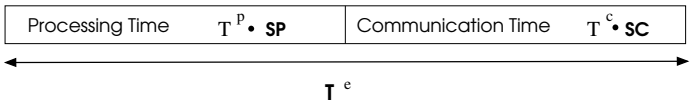


Figura 3.1: Modelado del tiempo de ejecución

3.3.1. Modelo del Tiempo de ejecución

El centro de nuestro interés es un Cluster C heterogéneo y no dedicado formado por un conjunto de β nodos $C = \{N^1 ... N^\beta\}$ y un conjunto de canales de comunicación $L = \{l^1 ... l^\beta\}$ internos al cluster, donde l^i representa el canal de comunicación entre el nodo N^i y el switch del cluster.

En el presente trabajo asumiremos un modelo de aplicación ampliamente utilizado en la literatura [BAG00, HSSY00, BBE03, BE03a, CC06, DA06]. En este modelo las aplicaciones paralelas se consideran no maleables y con asignación estática. Esto significa que el número de recursos que requiere una aplicación se mantiene inalterado durante toda su ejecución. Además, asumiremos que cada tarea se compone de un conjunto de iteraciones donde se alternan ciclos de cómputo y comunicación, que las tareas se ejecutan de forma separada y que los requisitos de cómputo y comunicación son similares entre todas las tareas que forman la aplicación.

Teniendo en cuenta estas consideraciones, definimos el tiempo de ejecución de una aplicación paralela (T^e) (Figura 3.1) mediante la ecuación 3.1.

$$T^e = T^p \cdot SP + T^c \cdot SC \tag{3.1}$$

donde T^p y T^c representan el tiempo de cómputo y comunicación obtenidos en un entorno dedicado y SP y SC el retardo sobre el tiempo de cómputo y comunicación producido por la compartición de los recursos con otros usuarios y su heterogeneidad.

En una situación ideal, donde los recursos son homogéneos y dedicados, ambos factores de retardo son iguales a 1 ($SP = SC = 1$) y el tiempo de ejecución T^e representa un mínimo ideal. En cambio, en una situación real donde los recursos son heterogéneos y están compartidos con otros usuarios, se produce un retraso en los tiempos de cómputo y comunicación que depende únicamente del grado la disponibilidad de los recursos.

A continuación se propone un método para aproximar el retardo en el cómputo (SP) y en las comunicaciones (SC) en función de la capacidad de los recursos.

3.3.1.1. Slowdown de Procesamiento (SP)

En un entorno heterogéneo y no dedicado, la capacidad de cómputo de un nodo depende de su potencia de cómputo y de su disponibilidad.

Definimos la ***Potencia relativa*** P^i de un nodo i, como la relación entre la potencia de cómputo del nodo i y el nodo más potente del Cluster. El rango de P^i es un valor entre $0 < P^i \leq 1$. $P^i = 1$ significa que el nodo i es el más potente del Cluster. Podemos medir la potencia de cómputo de un nodo promediando los tiempos de cómputo obtenidos para distintos tipos de benchmark, tal como se describe en [DZ97].

La ***Disponibilidad*** A^i de un nodo i, se define como el porcentaje de CPU disponible. Podemos aproximar la disponibilidad de la CPU relacionando el promedio de aplicaciones en el sistema con el uso de los ciclos de cómputo tal como se describe en [WSH00]. A^i toma valores entre $0 < A^i \leq 1$ para indicar el porcentaje de desocupación. $A^i \simeq 0$ significa que la CPU está ocupada al 100 %.

Definimos la ***Potencia efectiva*** de un nodo Γ^i, como el producto entre la ***Potencia relativa*** y la ***Disponibilidad*** del nodo i, tal como se expresa en la ecuación 3.2.

$$\Gamma^i = P^i \cdot A^i \tag{3.2}$$

donde Γ^i toma valores entre $0 \leq \Gamma^i \leq 1$. $\Gamma^i \simeq 0$ indica que el nodo i no dispone de suficientes recursos para ejecutar ninguna tarea y $\Gamma^i = 1$ cuando el nodo puede ejecutar tareas a la máxima potencia.

Una vez calculada la *Potencia efectiva* de cada nodo i podemos calcular los efectos sobre el retardo de cómputo de la aplicación paralela J. Sea T el conjunto de nodos del Cluster con tareas de la aplicación paralela J asignadas. El *Slowdown de procesamiento* $SP(J)^i$ que el nodo i del Cluster inflige sobre la aplicación paralela (J) se define como:

$$SP_J^i = \begin{cases} 0 & when\ N^i \notin T \\ (\Gamma^i)^{-1} & otherwise \end{cases} \tag{3.3}$$

Cuando un nodo no tiene asignada ninguna tarea de la aplicación paralela J, el efecto sobre el retardo de cómputo de la aplicación paralela es nulo sea cual sea la *potencia efectiva* del nodo. Cuando el nodo N^i alberga una tarea de la aplicación paralela, definimos el *Slowdown de procesamiento* SP^i sobre la tarea como la inversa de la *Potencia efectiva* del nodo i.

Finalmente, dado que asumimos que las tareas de una aplicación paralela se ejecutan de forma separada, definimos el ***Slowdown*** *de procesamiento de una aplicación*

paralela como el máximo *slowdown de procesamiento* de todo los nodos del Cluster:

$$SP_J = max\left\{SP_J{}^i,\ 1 \leq i \leq \beta\right\} \tag{3.4}$$

donde β es el número de nodos del Cluster.

3.3.1.2. Slowdown de Comunicación (SC)

La caracterización de la comunicación se basa en el modelo descrito por Jones en [JLPS05] para entornos homogéneos y dedicados. Jones no consideró la actividad del usuario local en la interacción de las comunicaciones ni la posibilidad de disponer de recursos con distintas capacidades. El ***Slowdown de comunicación*** de una aplicación paralela (SC_J) se define como la relación entre el bandwidth requerido por la aplicación (BW_J) y el bandwidth finalmente asignado (BW_J^{as}) por el sistema de comunicaciones como consecuencia de los reajustes de bandwidth realizados en los canales de comunicación más saturados. Formalmente:

$$SC_J = \left(\frac{BW_J}{BW_J^{as}}\right) \tag{3.5}$$

Cuando se produce la saturación de un canal de comunicación, el bandwidth asignado BW_J^{as} a las aplicaciones que comparten el canal es una fracción del bandwidth requerido BW_J, produciéndose un retraso en el tiempo de comunicación de la aplicación $(SC_J > 1)$. En cambio, cuando la aplicación recibe el bandwidth requerido $(SC_J \leq 1)$, las tareas de la aplicación se podrán comunicar a máxima velocidad.

En el *Algoritmo 3.1* se describe el proceso de obtención del bandwidth asignado a una aplicación paralela (BW_J^{as}). La idea principal es buscar los canales de comunicación saturados y reajustar el bandwidth de todas las aplicaciones que comparten dichos canales. El algoritmo finaliza cuando no existe saturación en ningún canal de comunicación del Cluster.

El *grado de saturación* de un canal (BW_i^{sat}) se calcula mediante la ecuación 3.6 y se define como la relación entre el bandwidth máximo del canal (BW_i^{max}) y el bandwidth asignado a todas las aplicaciones que comparten el canal. $BW_i^{sat} < 1$ significa que el canal i está saturado y que el bandwidth asignado a las aplicaciones que comparten el canal se debe reajustar de forma directamente proporcional al grado de saturación BW_i^{sat}.

$$BW_i^{sat} = \left(\frac{BW_i^{max}}{\sum BW_J^{as}}\right) \tag{3.6}$$

Algorithm 3.1 Cálculo del bandwidth asignado a una aplicación

1: **forall** job J, let $BW_J^{as} = BW_J$.
2: Let J_i be the set of jobs sharing link i.
3: **forall** ($i \in L$) Calculate BW_i^{sat}.
4: **while** ($\exists\ BW_i^{sat} < 1$) **do**
5: Identify the most saturated link i. (min$\{BW_i^{sat}\}$).
6: Reduce BW_J^{as} of every job $j \in J_i$ by a factor of BW_i^{sat}.
7: **forall** ($i \in L$) Calculate the new BW_i^{sat}.
8: **end while**

Dado que los canales de comunicación pueden estar compartidos con usuarios locales enviando información al exterior, el reajuste del bandwidth en un canal de comunicación no implica la eliminación de la saturación en el resto de canales utilizados por la aplicación paralela. Por este motivo se debe repetir el proceso de reajuste hasta eliminar la saturación de todos los canales del cluster.

Una vez reajustado el bandwidth de las aplicaciones paralelas se calcula el slowdown de comunicación de la aplicación paralela SC_J mediante la ecuación 3.5.

3.3.2. Estimación del Tiempo de ejecución remanente

Una vez modelado el tiempo de ejecución y el retardo provocado por la capacidad de los recursos, podemos estimar el tiempo de ejecución remanente de una aplicación paralela J, mediante la ecuación siguiente:

$$T_{rem}^{e}(J) = T^{e} \cdot Sd \tag{3.7}$$

donde T^e representa el tiempo de ejecución de la aplicación paralela J en un entorno dedicado y Sd el slowdown producido por la ocupación de los recursos.

El problema de la utilización directa de esta aproximación es que suponemos que la ocupación de los recursos no varía durante la ejecución de la aplicación J, y por tanto el slowdown Sd se mantiene constante. La realidad es muy diferente ya que compartimos los recursos con otras aplicaciones y éstas pueden finalizar durante el proceso de ejecución de la aplicación J dejando recursos libres y por tanto modificando la disponibilidad de los recursos.

El procedimiento de simulación descrito con detalle en la sección 3.4.2, permite capturar el comportamiento variable del sistema detectando los cambios en la utilización de los recursos y recalculando sus efectos en los tiempos de ejecución de las aplicaciones paralelas en cada nuevo escenario. Este procedimiento permite una

mayor aproximación a la realidad dinámica del sistema obteniendo mayor precisión en las estimaciones.

Cada vez que se detecta un cambio de estado, el simulador necesita estimar el tiempo de ejecución remanente de todas la aplicaciones que se encuentran en ejecución para conocer la siguiente aplicación en finalizar. Para calcular el tiempo de ejecución remanente de las aplicaciones paralelas necesitamos estimar el slowdown causado por el nuevo estado del sistema.

En el presente trabajo se propone dos métodos distintos para la estimación del slowdown de la aplicación paralela, el método *SDPN* y el método *Hybrid*. El método *SDPN* consiste en estimar el slowdown de la aplicación paralela basándose en el *modelo analítico* presentado en la sección 3.3.1. El método *Hybrid* utiliza el *método estadístico* IBL, presentado en la sección 3.2.2, para estimar el tiempo de ejecución de una aplicación paralela en función del estado de los recursos, y poder determinar así su slowdown.

A continuación se describen ambos métodos para el cálculo del tiempo de ejecución remanente. En la sección 3.5 se analiza su comportamiento bajo distintas condiciones de carga y se comparan los resultados obtenidos en cuanto a la precisión en la predicción.

3.3.2.1. Método SDPN para el cálculo del Tiempo de ejecución remanente

En cada nuevo paso de simulación, el núcleo de simulación estima el tiempo de ejecución remanente de todas la aplicaciones que se encuentran en ejecución, mediante la ecuación siguiente:

$$T^e_{rem}(J) = T^p_{rem} \cdot SP + T^c_{rem} \cdot SC, \tag{3.8}$$

Sea t el instante de tiempo en que se produce un cambio de estado. El *tiempo de ejecución remanente* T^e_{rem} de una aplicación paralela J en t se estima, relacionando los tiempo de procesamiento T^p_{rem} y comunicación T^c_{rem} remanentes en el instante t, con los slowdowns de procesamiento SP y comunicación SC producidos por la capacidad de los recursos en el instante t, según la ecuación 3.8.

Cuando una aplicación paralela en ejecución finaliza en el período $t+\Delta_t$ se recalcula el tiempo de ejecución remanente del resto de aplicaciones en ejecución mediante las ecuaciones 3.8 y 3.9, considerando la nueva situación de los recursos.

$$T^p_{rem} = T'^p_{rem} - \frac{T^p_{\Delta t}}{SP'}, \quad T^c_{rem} = T'^c_{rem} - \frac{T^c_{\Delta t}}{SC'} \tag{3.9}$$

donde el tiempo de procesamiento remanente T^p_{rem} en el instante $t+\Delta_t$, se calcula considerando la diferencia del tiempo remanente en el instante t, denotado por T'^p_{rem}, y el tiempo de procesamiento consumido durante el período $t+\Delta_t$ ponderado por el *slowdown de procesamiento* aplicado durante ese período, SP'. El procedimiento es igualmente válido para el cálculo del tiempo de comunicación remanente T^c_{rem}.

3.3.2.2. Método Hybrid para el cálculo del Tiempo de ejecución remanente

La estimación del tiempo de ejecución remanente de una aplicación paralela durante el proceso de simulación también se puede llevar a cabo con un método estadístico como los descritos en la sección 3.2. La principal diferencia con respecto al modelo analítico es que el *Slowdown* no se aproxima calculando de forma analítica la disponibilidad de los recursos de cómputo y comunicación sino que se calcula basándose en una estimación del tiempo de ejecución obtenida mediante un *modelo estadístico*.

El tiempo de ejecución remanente de una aplicación paralela recién llegada al sistema se determina mediante la ecuación siguiente:

$$T^e_{rem}(J) = T^e \cdot Sd \tag{3.10}$$

donde T^e representa el tiempo de ejecución de la aplicación paralela en un entorno dedicado, y Sd representa el retardo provocado por el estado actual de los recursos. Para poder estimar el retardo del tiempo de ejecución Sd en función del estado actual de los recursos proponemos realizar una estimación del tiempo de ejecución en las condiciones actuales y ponderarlo por el tiempo de ejecución de referencia T^e tal como se indica en la ecuación 3.11.

$$Sd = \frac{T^e_{estim}}{T^e} \tag{3.11}$$

donde T^e_{estim} representa el tiempo de ejecución estimado mediante un método estadístico considerando el estado actual del sistema.

Cuando una aplicación paralela en ejecución finaliza, se produce un cambio de estado y se debe recalcular el tiempo de ejecución remanente del resto de aplicaciones en ejecución considerando el nuevo estado de los recursos, mediante las ecuaciones 3.12 y 3.11.

$$T^e_{rem}(J) = \left(\frac{T'^e_{rem} - \Delta_t}{Sd'} \right) \cdot Sd \tag{3.12}$$

donde Δ_t es el tiempo transcurrido desde el último cambio de estado. T'^{e}_{rem} representa el tiempo de ejecución remanente calculado en el anterior estado, y Sd' el retardo producido por el anterior estado de los recursos.

Teniendo esto en cuenta, el término entre paréntesis representa el tiempo de ejecución remanente en el instante en que se produce el cambio de estado, y Sd es el retardo del tiempo de ejecución estimado según la nueva disponibilidad de los recursos, mediante la ecuación 3.11.

Considerar la disponibilidad de los recursos en cada paso de simulación es imprescindible para poder capturar el retardo en el tiempo de ejecución de las aplicaciones paralelas, permitiendo obtener mayor precisión en las estimaciones. Por este motivo, es conveniente utilizar un método estadístico que considere cierta información sobre el estado del sistema y por ello utilizamos el modelo ***IBL***, propuesto en el presente trabajo, para obtener las estimaciones del tiempo de ejecución T^{e}_{estim} utilizando el grado de paralelismo de los recursos como información sobre el estado del sistema.

3.4. Motor de Predicción

El motor de predicción (***P**rediction **E**ngine*) recibe peticiones tanto del Meta-Planificador como del Planificador local. Recopila toda la información del estado actual del sistema y estima el tiempo de *turnaround* de las aplicaciones paralelas, es decir, la suma de los tiempos de *espera* en las colas y de *ejecución* de las aplicaciones paralelas.

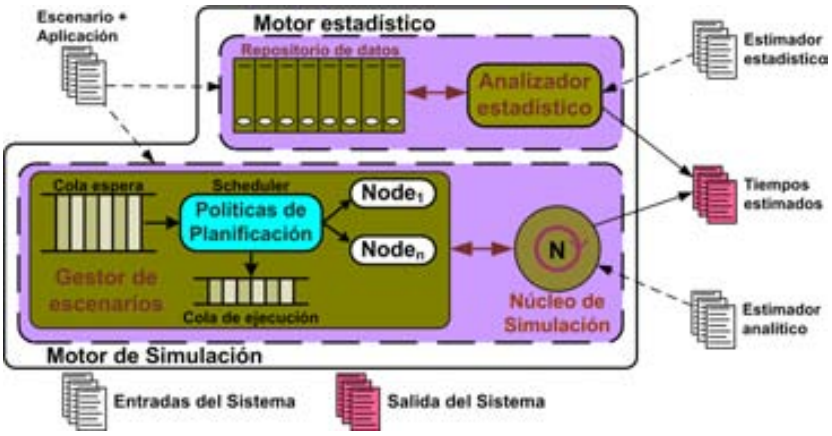


Figura 3.2: Motor de predicción

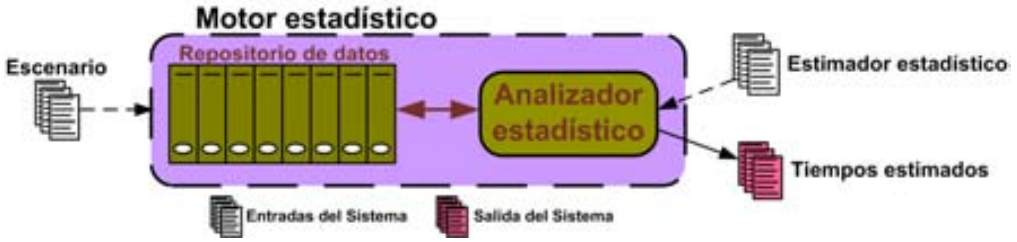


Figura 3.3: Motor estadístico.

En el presente trabajo proponemos dos nuevos métodos de estimación. Un método estadístico basado en el *aprendizaje de casos* (*IBL*) y un *modelo analític*o. Ambos métodos llevan a cabo las estimaciones utilizando procedimientos y mecanismos distintos. El método *IBL* requiere de una base de datos históricos y un motor estadístico para llevar a cabo distintos procesos estadísticos. El método analítico requiere de un simulador gestionado por eventos capaz de simular la planificación de procesos en un entorno cluster.

Para poder llevar a cabo el proceso de estimación hemos dotado al ***motor de predicción*** (Figura 3.2) de dos módulos distintos, el ***motor estadístico*** y el ***motor de simulación***. Cuando llega una nueva petición de estimación el motor de predicción decide utilizar el motor estadístico, el de simulación o una combinación de ambos en función del *estimador* que se haya configurado. El *estimador* configura los distintos métodos de predicción y decide como utilizarlos. En el presente capítulo analizaremos los resultados en la precisión para distintos estimadores.

En esta sección se presentan los elementos básicos de cada motor de estimación, la información que necesitan para llevar a cabo las estimaciones y como incorporan los métodos de estimación propuestos y otros de la literatura para poder realizar una comparativa.

3.4.1. Motor estadístico

El motor estadístico (Figura 3.3) se compone de dos elementos básicos, el ***repositorio de datos*** y el ***analizador estadístico***. El *repositorio de datos* almacena información sobre las aplicaciones paralelas y sobre su ejecución. En el apéndice B se puede consultar con detalle la información almacenada en el repositorio (base de datos históricos). El *analizador estadístico* se encarga de consultar la información del repositorio de datos y analizarla estadísticamente según el método estadístico que se desea aplicar.

Aplicación Paralela (Atributos) • Usuario • Nombre de la aplicación y Parámetros • Nodos requeridos • Cantidad de jiffies	
Cola de espera (Información de la cola donde se ubica la aplicación) • Nombre de la cola • Lista de aplicaciones en la cola	
Estado de los nodos (Información proporcionada por el *Monitoring Toolkit)* • Grado de paralelismo en cada nodo	

Tabla 3.1: Escenario de predicción en el motor estadístico

El motor estadístico recibe como información de entrada el ***escenario de predicción*** y la ***configuración del analizador***. El *escenario de predicción* se compone de información sobre el estado actual del sistema en el momento de la predicción. En la tabla 3.1 se describe la información que compone el *escenario de predicción*.

La *configuración del analizador* estadístico determina el método de estimación que deseamos aplicar, categorización o aprendizaje por casos y la función estadística (f_a). En el presente trabajo proponemos un método de estimación basado en el aprendizaje de casos (*IBL*). Con el fin de comparar los resultados con otros métodos de la literatura se ha implementado varios métodos de categorización que utilizan distintos patrones de clasificación de las experiencias pasadas. La función estadística utilizada en ambos casos es la media $(f_a = media)$.

3.4.2. Motor de simulación

El *motor de simulación* permite estimar tanto el tiempo de ejecución como el tiempo de espera de una aplicación paralela, simulando la planificación y ejecución de todas las aplicaciones paralelas que hay en el sistema. Los componentes principales del *motor de simulación* (Figura 3.4) son el ***gestor de escenarios*** y el ***núcleo de simulación***.

Para llevar a cabo el proceso de simulación sin interferir en el proceso de planificación real, el ***gestor de escenarios*** dispone de una reproducción de las unidades funcionales básicas del planificador real, el sistema de colas de trabajos en espe-

Figura 3.4: Esquema del motor de simulación.

ra, la cola de trabajos en ejecución, el planificador (Scheduler) y los recursos del sistema. Cuando llega una nueva aplicación al *motor de simulación*, el *gestor de escenarios* realiza una captura del *escenario de predicción* y lo reproduce en el *gestor de escenarios*. En la tabla 3.2 se muestra con detalle la información que compone el *escenario de predicción*.

El ***núcleo de simulación*** simula la planificación y ejecución de las aplicaciones paralelas en un determinado *escenario de predicción*, mediante el algoritmo presentado en la sección 3.4.2.1. Durante el proceso de simulación, el núcleo necesita estimar el tiempo de ejecución de las aplicaciones paralelas. El núcleo de simulación permite configurar el método de estimación que se desea utilizar.

En el presente trabajo proponemos dos métodos distintos para estimar el tiempo de ejecución de una aplicación paralela a cada paso de simulación, *SDPN* y *Hybrid*. El método *SDPN*, consiste en predecir el tiempo de ejecución utilizando el modelo analítico propuesto en la sección 3.3.2.1, donde se estima el tiempo de ejecución considerando las características y la ocupación de los recursos tanto de cómputo como de comunicación. El método *Hybrid*, consiste en estimar el tiempo de ejecución utilizando el método estadístico IBL propuesto en la sección 3.3.2.2.

Con el fin de comparar nuestras propuestas con otros métodos de la literatura, se ha implementado un método de estimación basado en el modelo analítico presentado por Hanzich en [HGH$^+$05a], que denominaremos *Hanzich*.

3.4.2.1. Algoritmo de simulación

Cuando llega una nueva aplicación paralela (*Target*), al motor de simulación (Figura 3.4), éste captura el escenario de predicción a partir del estado actual del sistema.

Aplicación Paralela (Atributos) • Aplicación y parámetros • Nodos requeridos • Requisitos de memória (MBytes) • Requisitos de comunicación (Mbps) • Tiempo estimado de ejecución y jiffies • Ratio de cómputo
Cola de espera (Información de la cola donde se ubica la aplicación) • Nombre de la cola • Lista de aplicaciones en la cola
Estado de los nodos (Información proporcionada por el *Monitoring Toolkit)* • Potencia Efectiva • Ocupación de Memoria (MBytes) • Disponibilidad de los recursos de red (Kbps) • Grado de paralelismo en cada nodo

Tabla 3.2: Escenario de predicción en el motor de simulación

A continuación activa el *núcleo de simulación* que simula la ejecución de todas las aplicaciones en las colas del sistema hasta obtener una estimación del tiempo de espera y ejecución de la aplicación paralela *Target*.

El proceso de simulación se describe en el *Algoritmo 3.2*. En primer lugar, el motor de predicción obtiene el escenario de predicción capturando la información necesaria sobre el estado del sistema: información de la aplicación paralela (*Target*), las aplicaciones en espera (*Low-level Queue*), el estado de las aplicaciones en ejecución (*Running List*) y el estado de los recursos. En cuanto al estado de los recursos, a través del *Monitoring Toolkit*, se obtiene la *Potencia Efectiva* (Γ^i) y el grado de paralelismo de cada nodo así como la disponibilidad de los canales de comunicación.

En cada nuevo paso de simulación, el núcleo de simulación estima el tiempo de ejecución remanente de todas la aplicaciones que se encuentran en la *Running List* (RL) en función de la disponibilidad actual de los recursos (línea 4). Para llevar a cabo la estimación del tiempo remanente podemos utilizar un método analítico o bien un método estadístico. El método de estimación que se desea utilizar se especifica durante la fase de configuración del núcleo de simulación.

Algorithm 3.2 Simulación guiada por el modelo analítico

1: Input arguments: Low-level Queue (*LQ*), Running List (*RL*, List of executing jobs), *Target*, Γ^i, Cluster (*C*), *Job_Selection_Policy* and *Node_Selection_Policy*.
2: Set t to the simulation start-time. $J = \phi$.
3: **while** $((RL \neq \{\phi\})$ and $(J \neq Target))$ **do**
4: **forall** $(j \in RL)$ Estimate the execution remaining time $T^e_{rem}(j)$, at time t.
5: Select the next job $J = min\{T^e_{rem}(j)\}$ to finish. Let $(t + \Delta t)$ the end time of J.
6: Remove J from *RL*.
7: **forall** $(j \in RL)$ Obtain its T^e_{rem} at time $(t + \Delta t)$.
8: **forall** $(j \in LQ)$ Increase the estimated $Waiting\_Time(j)$ in Δt time units.
9: **while** $((LQ \neq \{\phi\})$ and (available resources in *C*)) **do**
10: Select the next job $k \in LQ$ by using the *Job_Selection_Policy*.
11: Select the *C* nodes to map k by using the *Node_Selection_Policy*.
12: Remove k from *LQ*. Insert k into *RL*.
13: **end while**
14: Set t to $t + \Delta t$.
15: **end while**
16: **return** $T^e_{rem}(J)$ // $J = Target$

Una vez estimados los tiempos remanentes de todas las aplicaciones en la *RL*, se selecciona la aplicación paralela con el mínimo tiempo de ejecución remanente estimado $(min\{T^e_{rem}(J)\})$ (línea 5). Esta aplicación paralela J representa la próxima aplicación paralela en finalizar. Sea $\Delta_t = min\{T^e_{rem}(J)\}$, entonces $t + \Delta_t$ representa el instante de tiempo en que se estima que la aplicación J finalizará su ejecución.

Una vez localizada la siguiente aplicación paralela en finalizar, se elimina de la *RL* (línea 6) y se recalcula el tiempo de ejecución remanente (T^e_{rem}) del resto de aplicaciones en el instante $(t + \Delta_t)$ considerando los cambios en la disponibilidad de los recursos (línea 7). El tiempo de ejecución remanente se calcula mediante el modelo analítico *SDPN* o estadístico *Hybrid*, presentados en las secciones 3.3.2.1 y 3.3.2.2 respectivamente.

Una vez actualizados los tiempos remanentes de los trabajos en la *RL* el algoritmo de simulación actualiza el tiempo de espera de todos los trabajos en la *Low-level Queue* (línea 8). A continuación, intenta planificar el siguiente trabajo de la cola de espera según las políticas de planificación configuradas (*Job_Selection_Policy y Node_selection_Policy*) y avanza el período de simulación $t = t + \Delta_t$ (líneas 9..14).

El algoritmo finaliza cuando la aplicación *Target* devuelve su tiempo de espera y ejecución. El *Prediction Engine* calcula entonces el tiempo de *turnaround* sumando

Estimador	Método de estimación	Características	Consideraciones
Smith	Categorización	<u,e,n>	Aplicación
Li	Categorización	<u,c,e,n>	Aplicación
IBL	Aprendizaje basado en casos	<u,c,e,n,state>	Aplicación Cola de espera
SDPN	Analítico	$T^e = T^p SP + T^c SC$	Cómputo Comunicación
Hanzich	Analítico	$T^e = T^e \cdot MPL$	Cómputo (*MPL*)
Hybrid	Analítico + estadístico	$T^e = T^e \cdot Sd$ <u,c,e,n,state>	Slowdown estimado

Tabla 3.3: Estimadores implementados en el motor de predicción

ambos valores y devuelve la estimación de los tres tiempos, espera, ejecución y turnaround.

3.5. Análisis de los métodos predictivos

En la presente sección se analiza la desviación en la predicción de los tiempos de espera y ejecución bajo distintas condiciones de carga y utilizando la política de planificación *FCFS*. Escogemos esta política porque es una de las más utilizadas en entornos multicluster y porque no altera el orden de los trabajos en la cola de espera, resultando muy útil para poder comparar la precisión de los distintos métodos de predicción.

El objetivo principal de este análisis es detectar los métodos de predicción que obtienen mejores resultados de precisión y bajo que condiciones. Una vez analizados los resultados proponemos un método de predicción que utilizando una combinación de los métodos que ofrecen los mejores resultados, nos permite obtener la mejor precisión en la estimación del tiempo de *turnaround*.

Con el fin de implementar las propuestas presentadas y poderlas comparar con otros métodos de la literatura, hemos desarrollado varios *estimadores*. Cada *estimador* determina el método de predicción y la información que utiliza para llevar a cabo las estimaciones. En la tabla 3.3 se resumen los *estimadores* implementados.

En primer lugar, nuestra propuesta se centra en un estimador estadístico basado en un método de aprendizaje de casos, que denominaremos ***IBL*** (sección 3.2.2), y un

estimador analítico basado en un nuevo modelo analítico que considera la disponibilidad tanto de los recursos de cómputo como de comunicación y que denominaremos ***SDPN*** (sección 3.3.2.1).

Para comparar nuestras propuestas con otros métodos de la literatura hemos desarrollado varios estimadores implementando métodos de otros autores. ***Smith*** y ***Li*** son estimadores estadísticos basados en el método de categorización propuestos en [STF99] y [LGTW04] respectivamente. La única diferencia radica en la información que utilizan para categorizar las experiencias. ***Hanzich*** es un estimador basado en un modelo analítico propuesto en [HGH+05a] que estima el tiempo de ejecución considerando únicamente la ocupación de los recursos de cómputo. Además, hemos implementado un estimador que combina el uso de un modelo analítico y un método estadístico de aprendizaje basado en casos para estimar el tiempo de ejecución a cada nuevo paso de simulación, tal como se describe en la sección 3.3.2.2, y que denominaremos ***Hybrid***. *Hybrid* utiliza como estimador estadístico un modelo *IBL* donde el estado viene dado por el grado de paralelismo de los nodos.

El análisis comparativo se ha desarrollado desde el punto de vista de la desviación en la predicción y el coste algorítmico del proceso de estimación.

3.5.1. Entorno de experimentación

El entorno en que se ha llevado a cabo el presente análisis es un cluster real no dedicado formado por 16 nodos uniprocesador PIV a 3GHz con una capacidad de memoria de 1GB unidos a través de una red de comunicaciones Giga-Ethernet.

El objetivo de este análisis es evaluar la precisión en la predicción obtenida por los distintos estimadores bajo distintas condiciones de carga. Para poder llevar a cabo este análisis es necesario definir la carga paralela y la carga local que deseamos utilizar. La carga local es introducida por el sistema a través de un benchmark denominado *local_bench,* desarrollado en nuestro grupo de investigación. *local_bench* permite emular la utilización de CPU, Memoria y tráfico de red mediante valores parametrizables. La utilización de estos recursos ha sido caracterizada mediante la monitorización durante dos semanas de un aula de ordenadores de la universidad de Lleida, obteniendo los resultados descritos en el apéndice D.1.2.

De acuerdo con los resultados obtenidos hemos configurado *local_bench* con el perfil de un usuario con fuertes requisitos de interactividad y red. Este es un perfil de usuario muy común que maneja un entorno gráfico con aplicaciones de edición de documentos y que navega por internet realizando determinadas descargas. El perfil de usuario escogido que denominaremos *XWindows+Internet* ha sido modelado con los parámetros de uso siguientes, un 15 % de uso de CPU, 35 % de Memoria y 0,5MB/sec.

Las condiciones de carga que se han considerado son un *entorno dedicado*, un *entorno compartido sin carga local* y un *entorno compartido con carga local alta*. En un *entorno dedicado* las aplicaciones paralelas no comparten los recursos ni con otras aplicaciones paralelas ni con el usuario local. Esto significa que en cada recurso de cómputo solo puede haber una tarea paralela (MPL=1). En un *entorno compartido sin carga local*, una aplicación paralela no puede compartir los recursos con un usuario local pero si con otras aplicaciones paralelas hasta un máximo de cuatro ($MPL \leq 4$). En un *entorno compartido con carga local alta*, significa que las aplicaciones paralelas pueden compartir los recursos tanto con aplicaciones paralelas como con usuarios locales. En este entorno, se asigna de forma aleatoria el perfil de usuario local *XWindows+Internet* al 50 % de los nodos del cluster.

El conjunto de aplicaciones paralelas se ha constituido a partir de dos benchmarks de la NAS bien conocidos, el IS y el MG. En total se han construido 18 aplicaciones paralelas, 8 orientadas a la comunicación intensiva (IS) y 8 al cómputo intensivo (MG). Cada una de ellas con distintos requerimientos de tareas (2, 4 y 8) y con distinta clase de utilización de recursos (A, B y AB), donde cada clase especifica un grado de utilización de los recursos de cómputo, memoria y tráfico de red distinto. En el apéndice D.1.1 se describe con detalle la caracterización de cada aplicación paralela.

Una vez definidas las aplicaciones paralelas y con el fin de poder comparar bajo las mismas condiciones los distintos estimadores necesitamos definir un *workload paralelo* para ser utilizado en todas la pruebas. El *workload paralelo* está formado por un total de 700 aplicaciones escogidas de forma aleatoria entre el conjunto de aplicaciones paralelas, y que llegan al sistema bajo una distribución de Poison con un promedio de llegada $\mu = 4\,sec$.

La política de selección de trabajos escogida para realizar la comparativa es la más utilizada en entornos cluster, FCFS. Escogemos esta política porque no altera el orden de los trabajos en la cola de espera, resultando muy útil para poder comparar la precisión de los distintos métodos de predicción. La política *FCFS* permite la ejecución de aplicaciones paralelas de forma compartida, con un grado de paralelismo máximo de 4 aplicaciones ($MPL = 4$) por nodo.

Con el fin de comparar los distintos estimadores en el entorno más favorable hemos definido también una política denominada *BASIC*, que ejecuta las aplicaciones de forma dedicada, es decir, que los recursos asignados no son compartidos con ninguna otra aplicación o usuario local ($MPL = 1$).

3.5.2. Métricas de comparación

Estamos interesados en medir la desviación en las predicciones de cada estimador y evaluar el coste algorítmico en la obtención de las estimaciones.

En el presente trabajo, se desea evaluar la ***Desviación*** tanto del tiempo de ejecución como del tiempo de espera. Sea cual sea el parámetro a estimar, la desviación en cada experimento se define como el promedio del error de predicción ponderado por el promedio de los tiempos obtenidos en la ejecución real, tal como se muestra en la ecuación siguiente:

$$Desviacion = \frac{\sum_{1..n} |estimado_i - real_i|}{\sum_{1..n} real_i} \tag{3.13}$$

donde $estimado_i$ representa el valor estimado de la aplicación i, $real_i$ representa el valor obtenido en la ejecución real de la aplicación i y n es el total de aplicaciones paralelas ejecutadas.

Para obtener el ***coste algorítmico*** (ct) de cada estimador calculamos el promedio del tiempo de respuesta en cada experimento, mediante la ecuación siguiente:

$$ct = \frac{\sum_{1..n} tr_i}{n} \tag{3.14}$$

donde tr_i representa el tiempo empleado en obtener una estimación para la aplicación i y n es el total de estimaciones realizadas.

Aunque estas métricas nos ofrecen una buena aproximación para comparar los distintos estimadores, no proporcionan suficiente información para realizar un análisis exhaustivo y contestar algunas preguntas importantes que se plantean. ¿Tienen realmente las aplicaciones un comportamiento predecible o en cambio es tan variable que no es viable intentar realizar una predicción? ¿Como de sensible es un estimador a las variaciones en las condiciones de carga? ¿Tiende un estimador a sobreestimar o subestimar la realidad?

Otra cuestión que debemos considerar es la capacidad del estimador de representar la realidad. Puede suceder que un estimador A represente muy bien la realidad de lo que sucede pero con una desviación mucho mayor que otro estimador B, que realiza estimaciones más cercanas a los valores reales pero con una mayor variabilidad en el error cometido. Aunque *A* representa mejor la realidad de lo que sucede en el sistema, las estimaciones están más alejadas de la realidad que utilizando el estimador B. Sin embargo, si el error de A es prácticamente el mismo entre estimaciones significa que corrigiendo el error sistemático cometido en A podemos obtener incluso mejores resultados de desviación que en *B*.

Estos comportamientos los podemos observar gráficamente mediante los ***histogramas*** y los ***gráficos de dispersión***. El *histograma* nos permite observar distintos fenómenos. En primer lugar, como de cerca se encuentran los valores estimados de la mediana. Esto nos permite conocer si el comportamiento de una aplicación es predecible o bien si los valores en cada muestra son muy dispersos y no es viable su predicción. En segundo lugar, si comparamos el histograma de los tiempos estimados y reales de cada aplicación podemos observar hasta qué punto se ajusta un estimador al comportamiento de la aplicación. Además el histograma nos permite observar si los tiempos reales entre distintas ejecuciones de una misma aplicación son parecidos o existen comportamientos muy variables. Los casos donde aparecen muchos valores extremos dificultan las predicciones.

Otro método gráfico que nos proporciona gran información son los *gráficos de dispersión*. El gráfico de dispersión nos permite observar si el error cometido presenta una gran dispersión o bien si es predecible y por tanto se puede corregir. También nos proporciona información de si se estima por encima o por debajo de la realidad y como de grande es el error en las estimaciones.

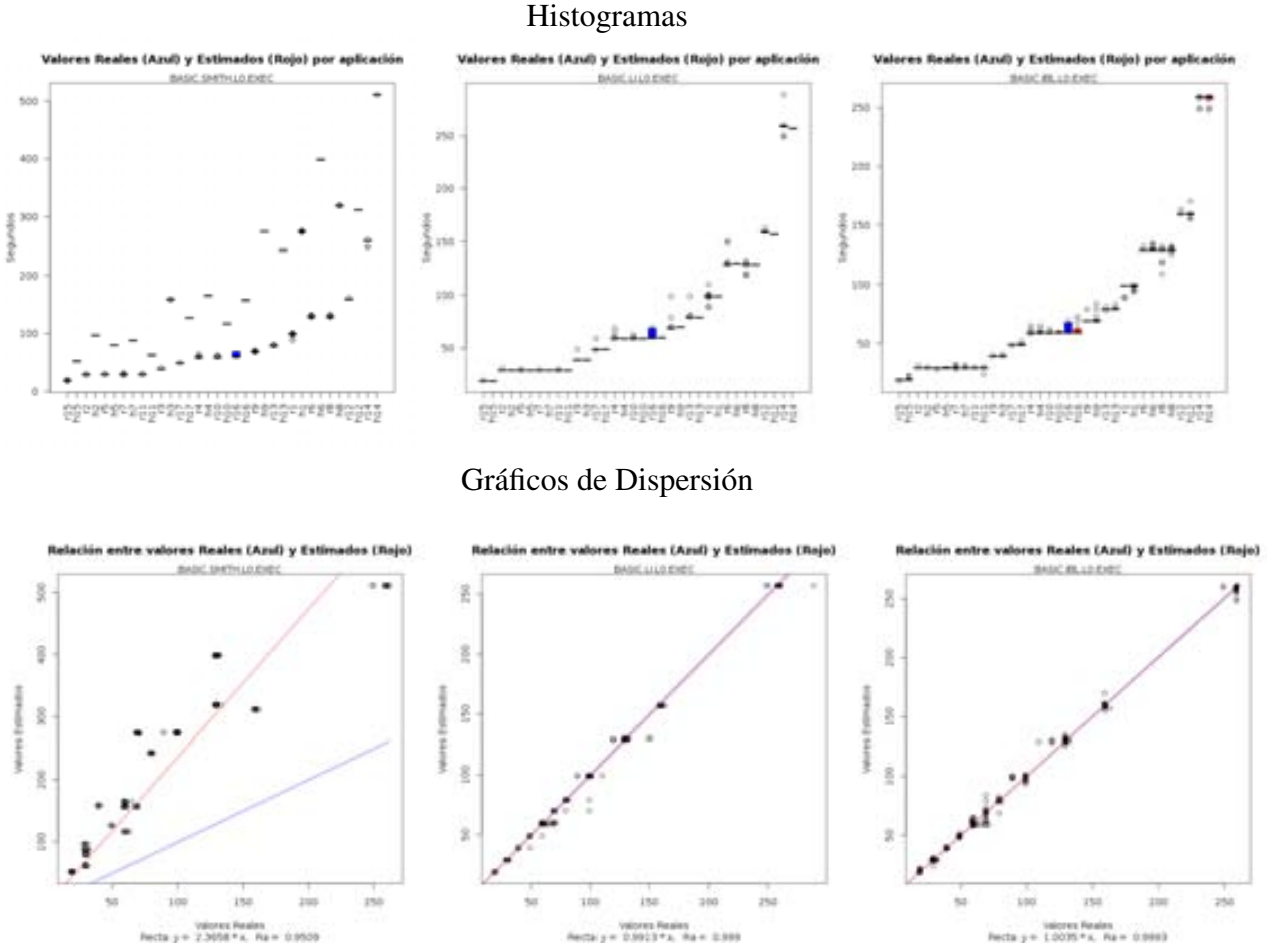
En el presente análisis representamos gráficamente, para cada aplicación, un histograma de valores reales y estimados. Además, esto se introduce en un único gráfico permitiendo comparar la naturaleza de las distintas aplicaciones.

Por otro lado, se presenta un gráfico de dispersión mostrando en el eje X los valores reales y en el eje Y los valores estimados. Cada gráfico de dispersión muestra además la información siguiente, la recta identidad (Azul), la recta de regresión (Roja), la pendiente de la recta de regresión y el coeficiente de correlación ajustado (Ra).

La recta identidad (Azul) se corresponde con la estimación esperada, es decir, aquella donde el error es cero. La recta de regresión (Roja) corresponde con la recta que mejor se ajusta al conjunto de estimaciones realizadas. La pendiente de la recta de regresión es el valor por el que hay que dividir el valor estimado para que coincida con el valor real. Este valor nos es útil para corregir las predicciones en los casos en que la dispersión del error no sea muy grande. El coeficiente de correlación ajustado (Ra) da una idea de la dispersión de la estimación, es decir, lo que se ajustan las estimaciones a la recta de regresión. $Ra \simeq 1$ significa que el error es el mismo en todas las estimaciones, sea cual sea el valor real. En estas situaciones es posible plantear la aplicación de un factor corrector que permita aproximar las estimaciones a su valor real y reducir significativamente la predicción en la desviación.

3.5.3. Predicción del tiempo de ejecución

En esta sección analizamos el comportamiento de los estimadores ante la estimación del tiempo de ejecución bajo distintas condiciones de carga. Las condiciones



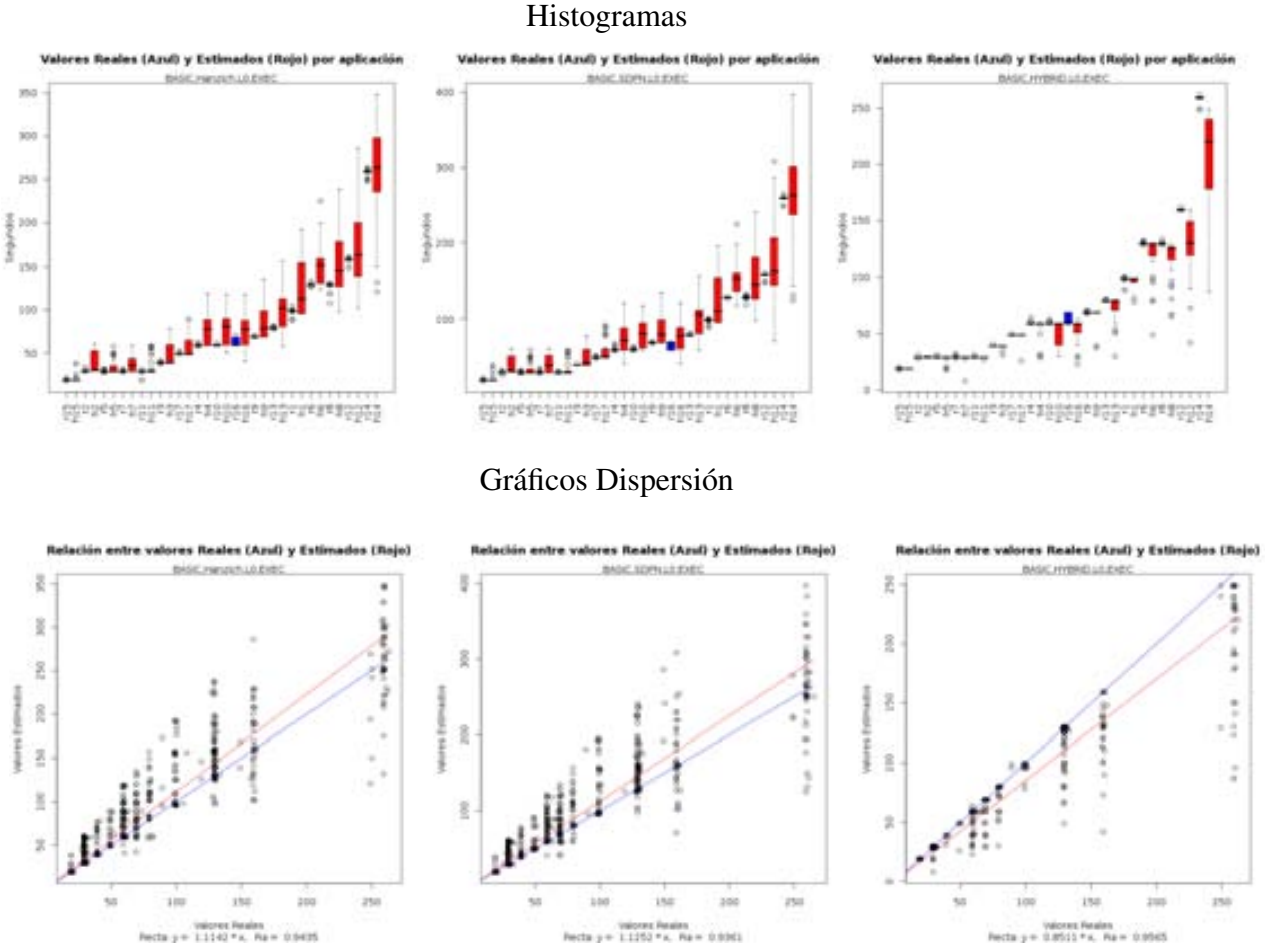
(a) Smith (b) LI (c) IBL

Figura 3.5: T. Ejecución. Entorno dedicado - Estadísticos

de carga que se han considerado son el *entorno dedicado*, *entorno compartido* y e*ntorno compartido con alta carga* local, definidos en la sección 3.5.1.

3.5.3.1. Entorno dedicado

La Figura 3.5 muestra los resultados obtenidos para la estimación del tiempo de ejecución de los estimadores estadísticos en un entorno dedicado. Los histogramas en la Figura 3.5 nos permiten observar que las aplicaciones presentan el mismo tiempo de ejecución entre distintas ejecuciones. Esto nos indica que el tiempo de



(a) Hanzich (b) SDPN (c) Hybrid

Figura 3.6: T. Ejecución. Entorno dedicado - Analíticos

ejecución es muy predecible en un entorno dedicado, lo cual es de esperar puesto que las aplicaciones no comparten recursos.

Los gráficos de dispersión (Figura 3.5) nos muestran que los estimadores estadísticos *Li* y *IBL* ajustan muy bien las predicciones a la realidad, no existiendo prácticamente ninguna diferencia entre ellos. Sin embargo, *Smith* tiende a sobreestimar los tiempos de ejecución. Esta desviación puede ser producida por la falta de información que tiene *Smith* respecto a los otros dos estimadores. Tanto *Li* como *IBL* consideran en las estimaciones información sobre la cola donde la aplicación ha sido planificada. Esta información es importante puesto que el tipo de cola define la política de planificación y en función de la política se pueden dar unos tiempos medios u otros.

Estadísticas Tiempo de Ejecución			
Estimador	Pendiente Recta	Ra	Desv (%)
Smith	2.3658	0.9509	1.6
Li	0.9913	0.999	0.01
IBL	1.0035	0.9993	0.01
Hanzich	1.1142	0.9435	0.23
SDPN	1.1252	0.9361	0.24
Hybrid	0.8511	0.9565	0.11

Tabla 3.4: Tiempos de ejecución en entorno dedicado

Los estimadores analíticos (Figura 3.6) tienden a una mayor dispersión de los valores estimados. Entre *Hanzich* y *SDPN* no existen prácticamente diferencias y ambos tienden a sobreestimar ligeramente el tiempo de ejecución. El comportamiento de *Hybrid* es muy parecido al de los modelos analíticos en la dispersión de los datos pero en cambio tiende a subestimar los tiempos reales. También se observa en *Hybrid*, que las aplicaciones con tiempos de ejecución pequeños no presentan dispersión en las estimaciones como sucede con los modelos *Hanzich* y *SDPN*. Este comportamiento es muy parecido al de los estimadores estadísticos. Su naturaleza híbrida hace que *Hybrid* presente un comportamiento similar al de los estimadores estadísticos y analíticos a la vez.

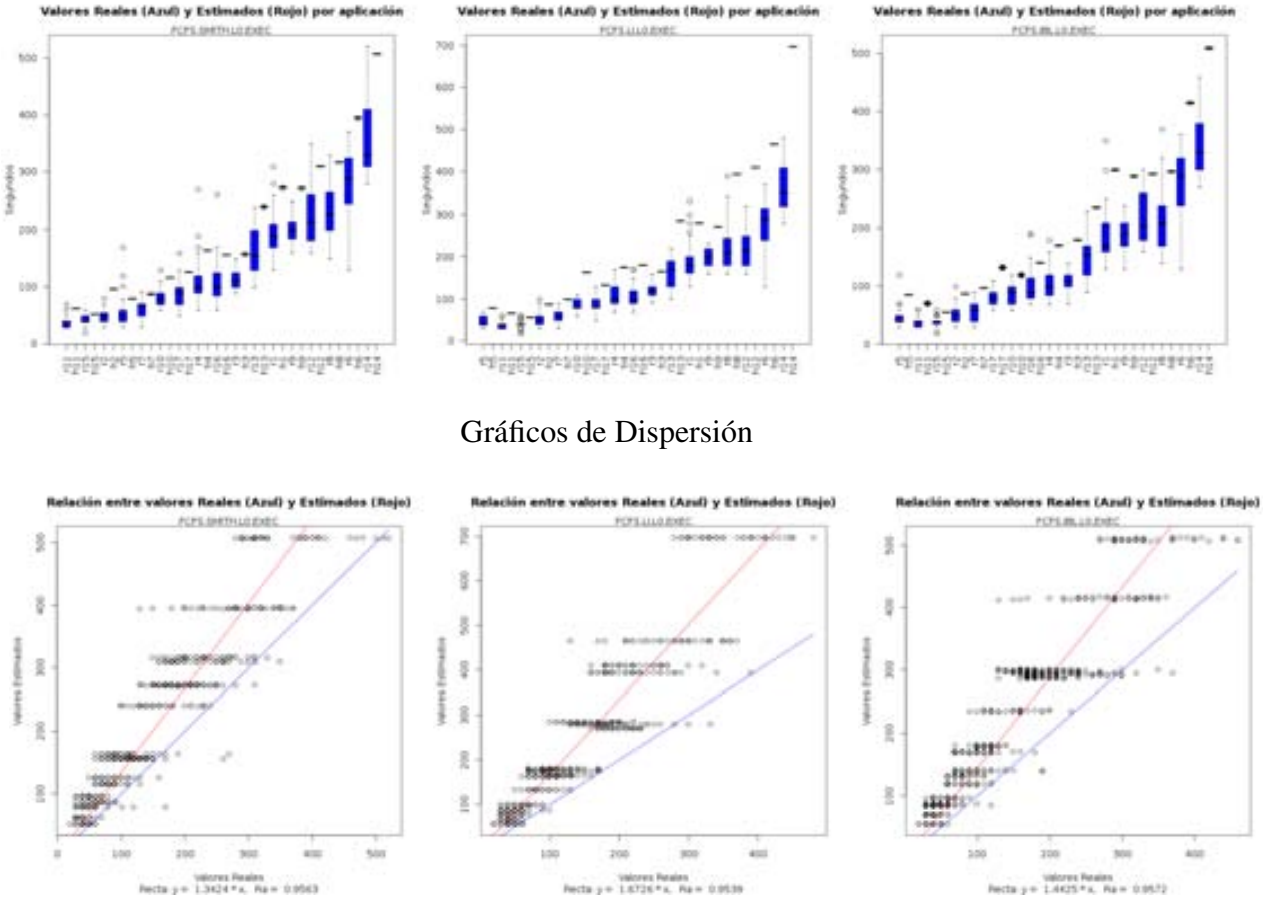
Como se observa en la tabla 3.4, tanto los estimadores estadísticos como analíticos obtienen desviaciones muy pequeñas, aunque la baja dispersión de los métodos estadísticos proporciona mejores resultados de precisión.

3.5.3.2. Entorno compartido sin carga local

Hasta el momento, los resultados son los esperados dado que no se comparten recursos con otras aplicaciones. La cuestión es como se comportan los estimadores cuando se aumenta el grado de paralelismo de las aplicaciones paralelas en los nodos del cluster ($MPL \leq 4$).

La Figura 3.7 muestra un comportamiento ligeramente distinto de los estimadores estadísticos respecto al entorno dedicado. En primer lugar, se producen variaciones en los tiempos de ejecución de una aplicación paralela que antes no se daban. En segundo lugar, estas variaciones no son capturadas por los estimadores estadísticos, produciéndose una mayor dispersión en las estimaciones y por tanto mayor desviación.

Histogramas

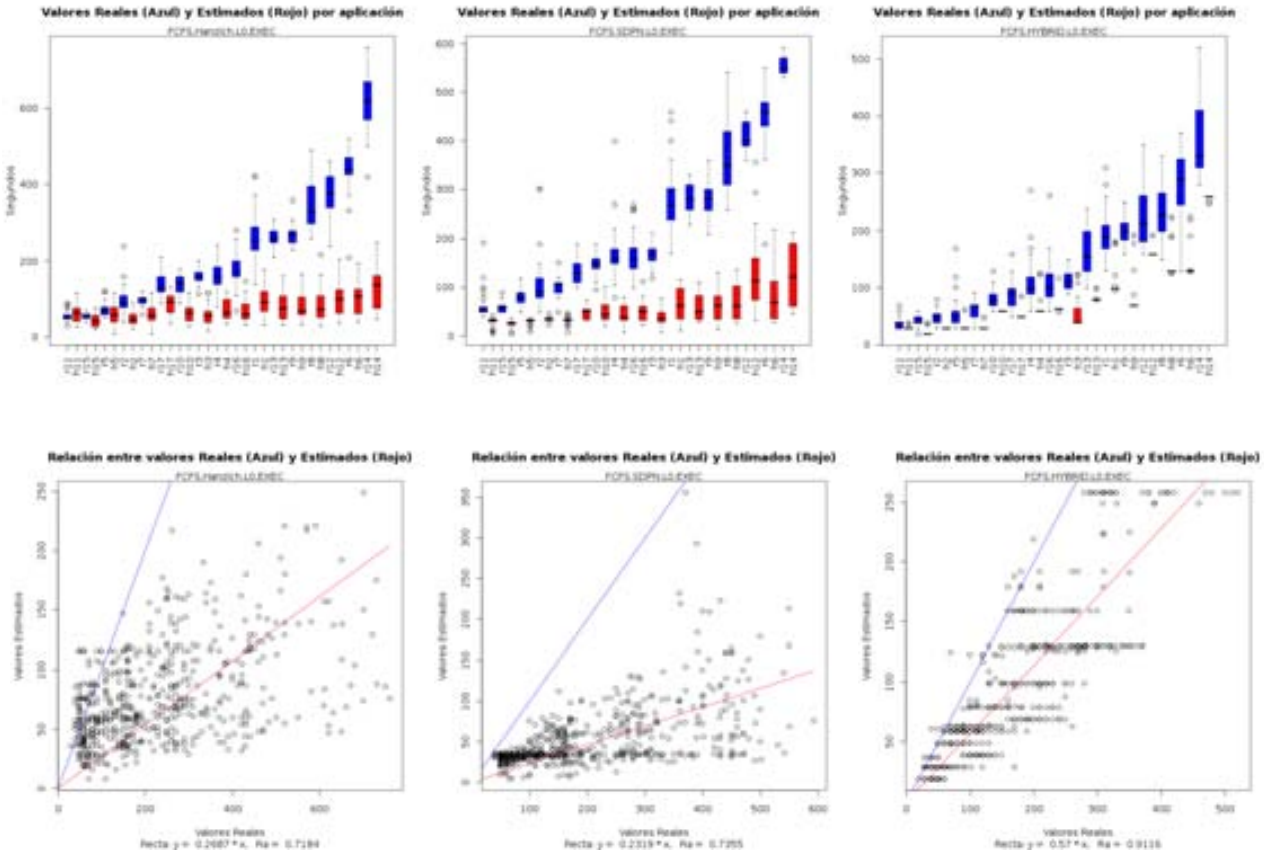


Gráficos de Dispersión

(a) Smith (b) LI (c) IBL

Figura 3.7: T. Ejecución. Entorno compartido sin carga local - Estadísticos

En la Tabla 3.5 se observa que no existen prácticamente diferencias entre los estimadores estadísticos, aunque *Li* obtiene peores resultados de desviación. Según el coeficiente de regresión ajustado (*Ra*), todos ellos se ajustan bastante bien a la recta de regresión y tienen un comportamiento lineal que refleja que a medida que aumenta el tiempo de ejecución real también aumenta el estimado. La distancia de la recta de regresión (Roja) a la recta identidad (Azul) es prácticamente la misma en los tres estimadores, aunque ligeramente superior para *Li*. Los tres estimadores estadísticos tienden a sobreestimar el tiempo de ejecución.



(a) Hanzich (b) SDPN (c) Hybrid

Figura 3.8: T. Ejecución. Entorno compartido sin carga local - Analíticos

Estadísticas Tiempo de Ejecución			
Estimador	Pendiente Recta	Ra	Desv (%)
Smith	1.3424	0.9563	0.43
Li	1.6726	0.9539	0.69
IBL	1.4425	0.9572	0.54
Hanzich	0.2687	0.7184	0.69
SDPN	0.2319	0.7355	0.62
Hybrid	0.57	0.9116	0.42

Tabla 3.5: Tiempos de ejecución en entorno compartido sin carga

Los estimadores analíticos (Figura 3.8) tienen el comportamiento contrario que los estadísticos. Así como los estadístico tienden a sobreestimar los tiempos de ejecución, los estimadores analíticos tienden a subestimar los tiempos. Los estimadores completamente analíticos *Hanzich* y *SDPN* muestran una gran dispersión en las estimaciones y un comportamiento muy similar entre ambos. En cambio *Hybrid*, aunque sigue subestimando como los métodos analíticos, ofrece unos resultados de desviación y dispersión semejantes a los estimadores estadísticos. En este caso concreto, *Hybrid* es el estimador que ofrece mejores resultados en la desviación.

Los métodos analíticos son muy optimistas en cuanto al efecto de la compartición de los recursos, y esto se debe tener en cuenta para obtener mejores predicciones del tiempo de ejecución. En este caso los estimadores estadísticos ofrecen mejores resultados que los estimadores analíticos.

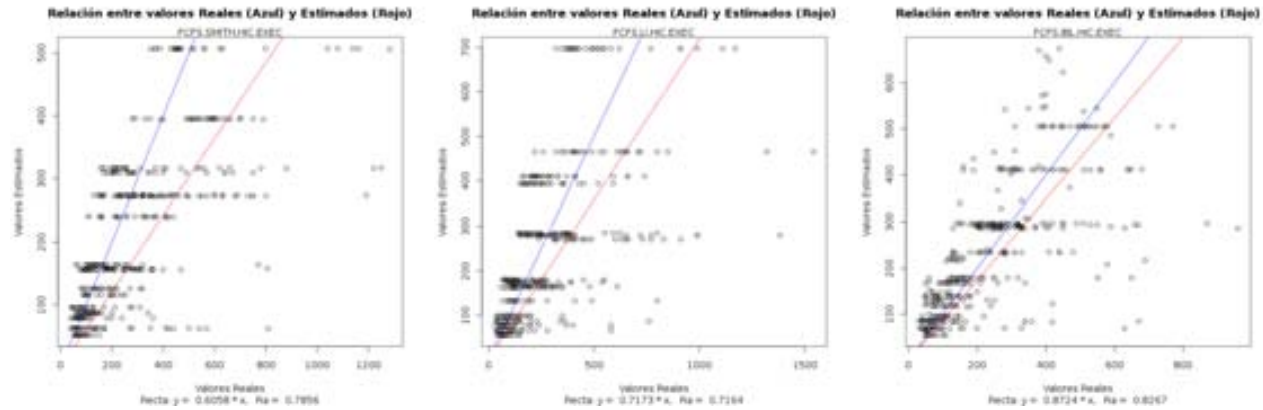
3.5.3.3. Entorno compartido con alta carga local

En los histogramas de las Figuras 3.9 y 3.10 se observa que las variaciones en el tiempo de ejecución en un entorno con mayor carga local aumenta significativamente, produciéndose un mayor número de casos que se alejan un poco del comportamiento habitual de la aplicación (mediana), añadiendo dificultad a la predicción.

Teniendo en cuenta este fenómeno y observando la Tabla 3.6, es significativo destacar que los estimadores estadísticos obtienen incluso mejores desviaciones que en el caso sin carga local. Entre los estimadores estadísticos el que ofrece mejores resultados en la desviación y en la dispersión es el modelo *IBL* propuesto en el presente trabajo. Esto nos lleva a pensar que la consideración del estado en las estimaciones toma importancia cuanto más cargado está el sistema y es entonces cuando empieza a dar buenos resultados. En situaciones que no hay mucha carga los métodos estadísticos no proporcionan grandes diferencias.

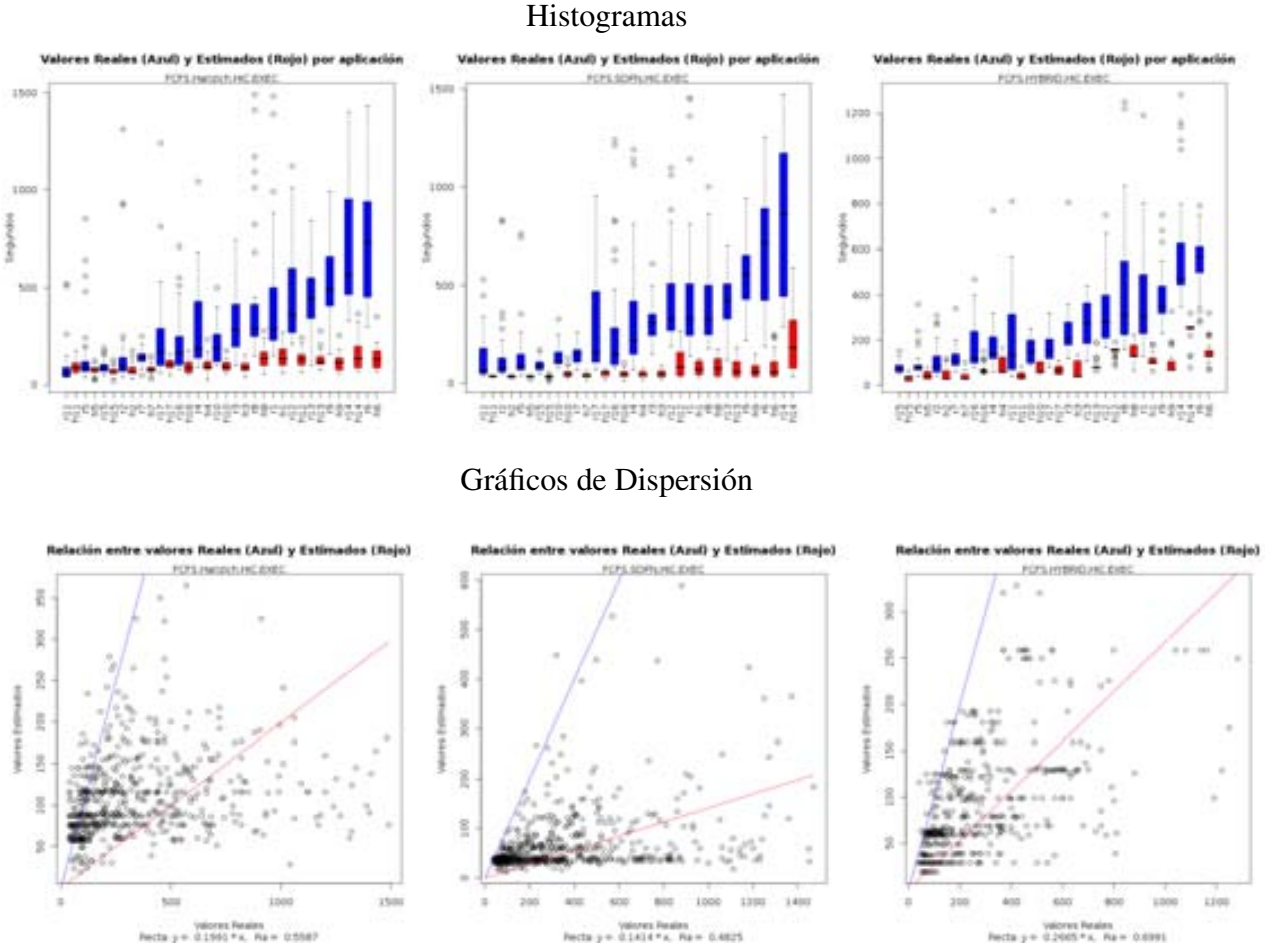
Histogramas

Gráficos de Dispersión



(a) Smith (b) LI (c) IBL

Figura 3.9: T. Ejecución. Entorno compartido carga local alta - Estadísticos



(a) Hanzich (b) SDPN (c) Hybrid

Figura 3.10: T. Ejecución. Entorno compartido carga local alta - Analíticos

Los estimadores analíticos (Figura 3.10) empeoran su comportamiento con el aumento de la carga local. En el caso analítico, se simula la ejecución de todas las aplicaciones del sistema antes de obtener una estimación del tiempo de ejecución. Un pequeño error en cada una de las estimaciones se propaga rápidamente con el número de tareas en la cola del cluster y el aumento de la carga local del sistema. Este fenómeno aún es más acentuado en el método *SDPN* ya que se tienen en cuenta un mayor número de parámetros del estado del sistema, y por tanto existe mayor probabilidad de cometer errores al aproximar los valores de estos parámetros.

Estadísticas Tiempo de Ejecución			
Estimador	Pendiente Recta	Ra	Desv (%)
Smith	0.6058	0.7856	0.4
Li	0.7173	0.7164	0.44
IBL	0.8724	0.8267	0.33
Hanzich	0.3139	0.6198	0.66
SDPN	0.1414	0.4825	0.79
Hybrid	0.2665	0.6991	0.59

Tabla 3.6: Tiempos de ejecución en entorno compartido con alta carga

El método analítico que mejores resultados obtiene es el *Hybrid.* Este método sigue mostrando cierta similitud con el comportamiento de los modelos estadísticos obteniendo así mejores resultados de desviación (hasta un 10 % menos de error con respecto a *Hanzich*).

3.5.4. Predicción del tiempo de espera

En esta sección analizamos el comportamiento de los estimadores respecto a la estimación del tiempo de espera bajo las mismas condiciones de carga utilizadas en el apartado anterior.

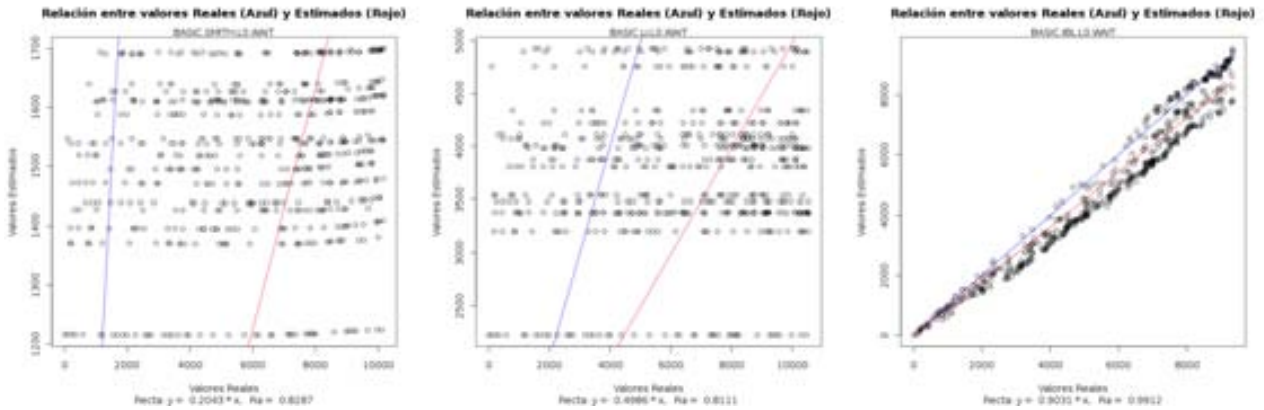
3.5.4.1. Entorno dedicado

En la Figura 3.11 se pueden observar grandes diferencias en la estimación del tiempo de espera con respecto a la estimación del tiempo de ejecución. En primer lugar, la dispersión de los tiempos de espera reales es mucho mayor que en los tiempos de ejecución (Figura 3.5). Esto es debido a que existen muchos parámetros que afectan al tiempo de espera de un aplicación en la cola de un cluster y esto provoca que el tiempo de espera sea mucho más difícil de predecir.

Los estimadores *Li* y *Smith* parecen tener comportamientos similares. Ambos métodos tienden a subestimar enormemente los tiempos reales. *Li* subestima los tiempos reales hasta un 50 % y obtiene mejores resultados en la desviación que *Smith*. El estimador *IBL* muestra un comportamiento totalmente diferente a los otros estimadores estadísticos obteniendo muy buenos resultados. El modelo *IBL* es capaz de capturar las variaciones presentadas por los tiempos reales, proporcionando muy poca dispersión ($Ra = 0{,}9912$) y ajustándose muy bien a la recta Identidad (Azul) ($Dev = 0{,}12$), obteniendo únicamente una desviación del 12 %.

Histogramas

Gráficos de Dispersión

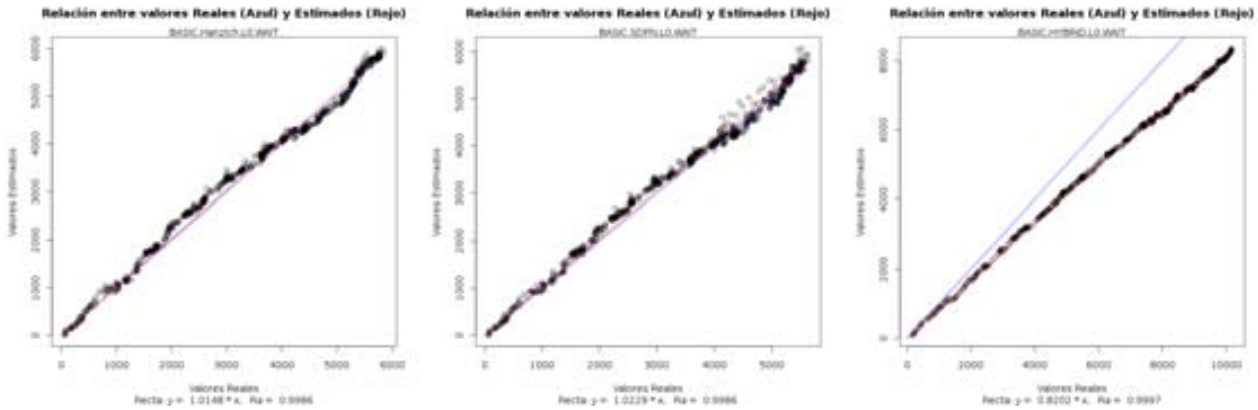


(a) Smith (b) LI (c) IBL

Figura 3.11: T. Espera. Entorno dedicado - Estadísticos

Histogramas

Gráficos de Dispersión



(a) Hanzich (b) SDPN (c) Hybrid

Figura 3.12: T. Espera. Entorno dedicado - Analíticos

Esto es así porque el modelo IBL considera como estado del sistema el número de trabajos en la cola y el tiempo de CPU de estos trabajos, lo que proporciona información suficiente a la hora de estimar los tiempos de espera.

Los estimadores analíticos (Figura 3.12), *Hanzich* y *SDPN*, no presentan dispersión y se ajustan perfectamente a la recta identidad, obteniendo valores de desviación inferiores al 5 %. *Hybrid* ofrece un comportamiento similar a los métodos anteriores ajustándose perfectamente a la recta de regresión pero desviándose ligeramente de

Estadísticas Tiempo de Espera			
Estimador	Pendiente Recta	Ra	Desv (%)
Smith	0.2043	0.8287	0.77
Li	0.4986	0.8111	0.70
IBL	0.9031	0.9912	0.12
Hanzich	1.0148	0.9986	0.04
SDPN	1.0229	0.9986	0.04
Hybrid	0.8202	0.9997	0.18

Tabla 3.7: Tiempos de espera en entorno dedicado

la recta identidad. Esto es debido a la dificultad que tiene el modelo estadístico utilizado por *Hybrid* para estimar los tiempos de espera.

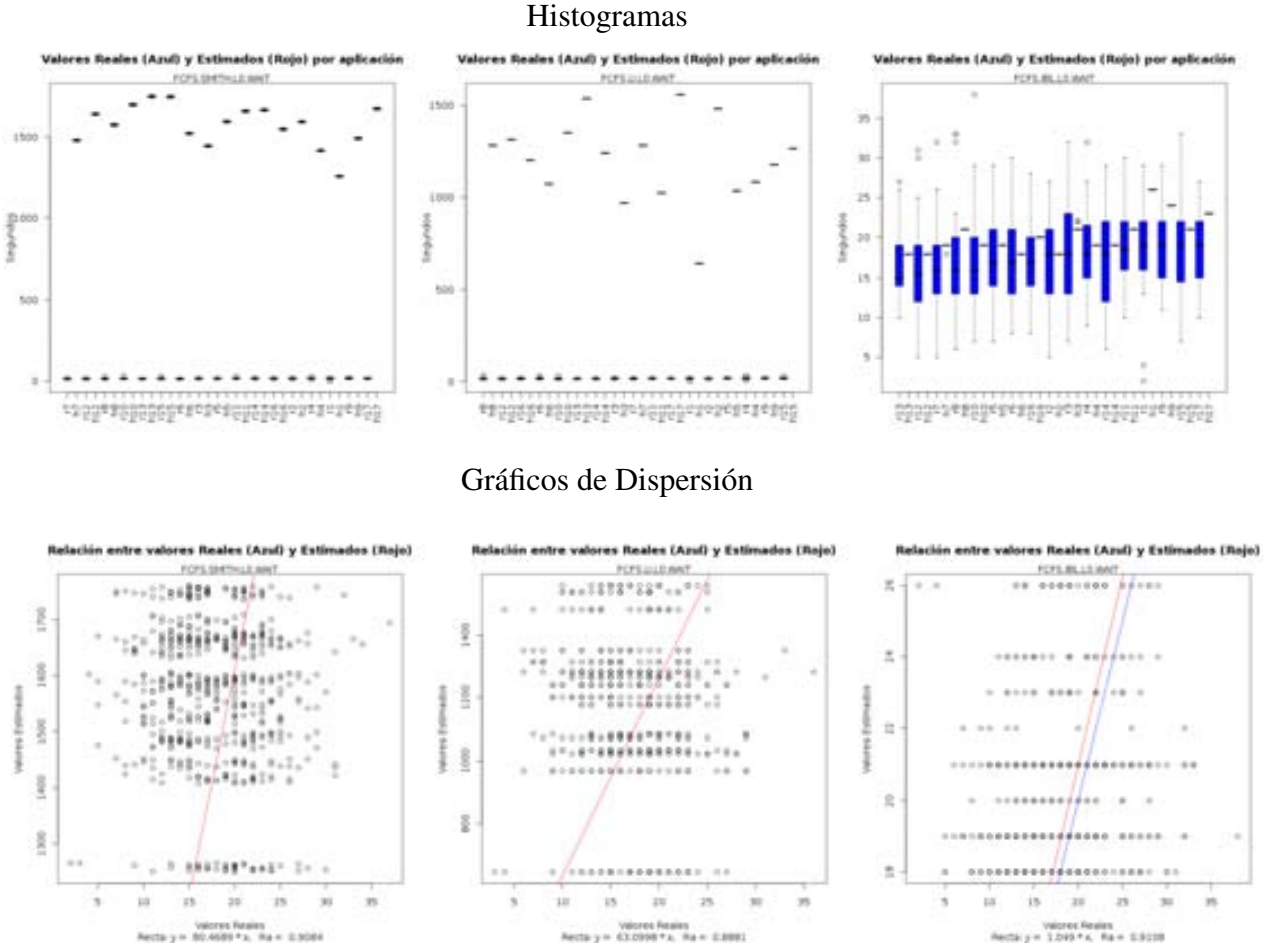
3.5.4.2. Entorno compartido sin carga local

En los entornos con carga compartida se hace aún más evidente la dificultad que tienen los estimadores estadísticos para estimar el tiempo de espera. En la Figura 3.13 se observa como los estimadores han pasado de subestimar el tiempo de espera respecto al entorno dedicado a sobreestimar enormemente este tiempo. Los estimadores *Smith* y *Li* ofrecen tal desviación de la estimación (Tabla 3.8) que no son útiles para estimar el tiempo de espera. El modelo *IBL* sorprendentemente mantiene una desviación por debajo del 30 % y con una gran agrupación de los valores entorno a la recta identidad.

Estadísticas Tiempo de Espera			
Estimador	Pendiente Recta	Ra	Desv (%)
Smith	80.4689	0.9084	86.79
Li	63.0998	0.8881	67.26
IBL	1.049	0.9108	0.28
Hanzich	0.3803	0.9661	0.61
SDPN	0.1508	0.9794	0.85
Hybrid	0.345	0.9342	0.75

Tabla 3.8: Tiempos de espera en entorno compartido sin carga

Los estimadores analíticos (Figura 3.14) en la estimación del tiempo de espera de un entorno compartido no ofrecen las mismas prestaciones que en un entorno dedicado, pero aporta resultados muy interesantes. En primer lugar, se sigue observando una

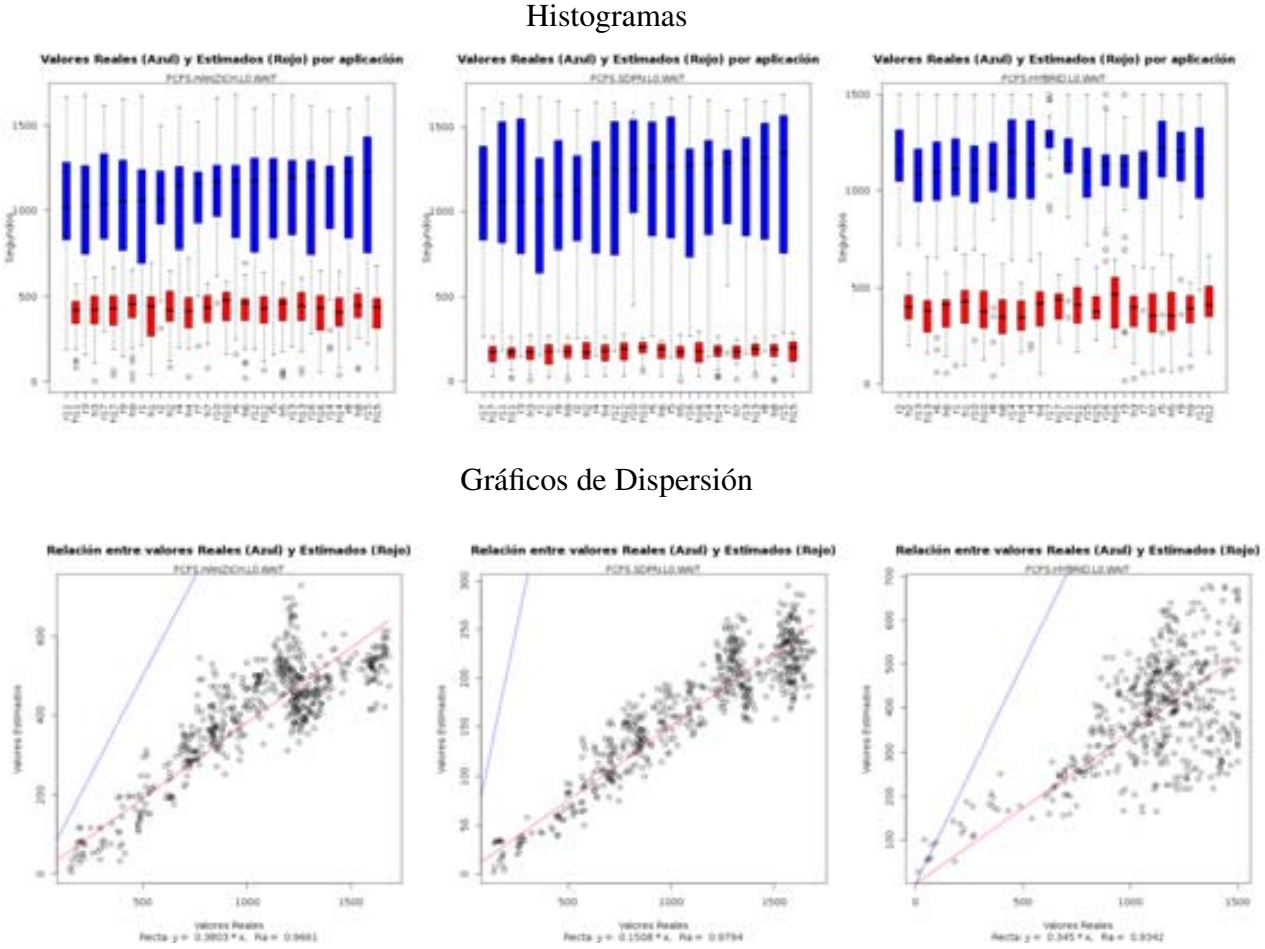


(a) Smith (b) LI (c) IBL

Figura 3.13: T. Espera. Entorno compartido sin carga local - Estadísticos

clara tendencia a subestimar el tiempo de espera al igual que sucedía con el tiempo de ejecución en la sección anterior. Esto tiene sentido, dado que el tiempo de espera de las aplicaciones paralelas en la cola depende del tiempo estimado de ejecución de las aplicaciones anteriores.

En segundo lugar, aunque existe una gran desviación con respecto a los valores reales, las estimaciones ofrecen baja dispersión ajustándose muy bien a la recta de regresión. Este resultado hace pensar que es posible la aplicación de un factor corrector que permita corregir el error sistemático que se comete durante el proceso de estimación.



(a) Hanzich (b) SDPN (c) Hybrid

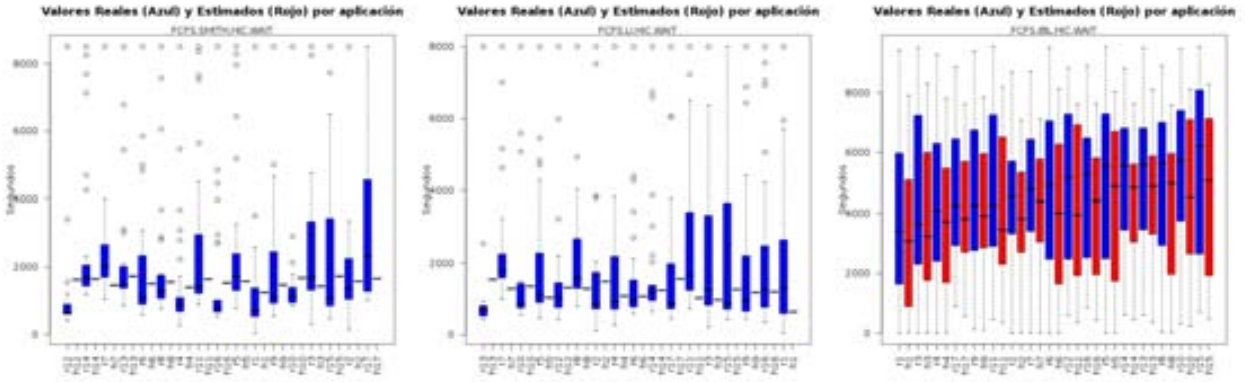
Figura 3.14: T. Espera. Entorno compartido sin carga local - Analíticos

En la tabla 3.8 se puede observar como *SDPN* obtiene una mayor desviación que *Hanzich* dado que se aleja más de la recta identidad pero en cambio se ajusta mejor a la recta de regresión. Esto da pié a pensar que corrigiendo la desviación es posible obtener mejores resultados de desviación, incluso mejores que con el modelo *IBL* dado que la dispersión de las estimaciones con este modelo es mayor.

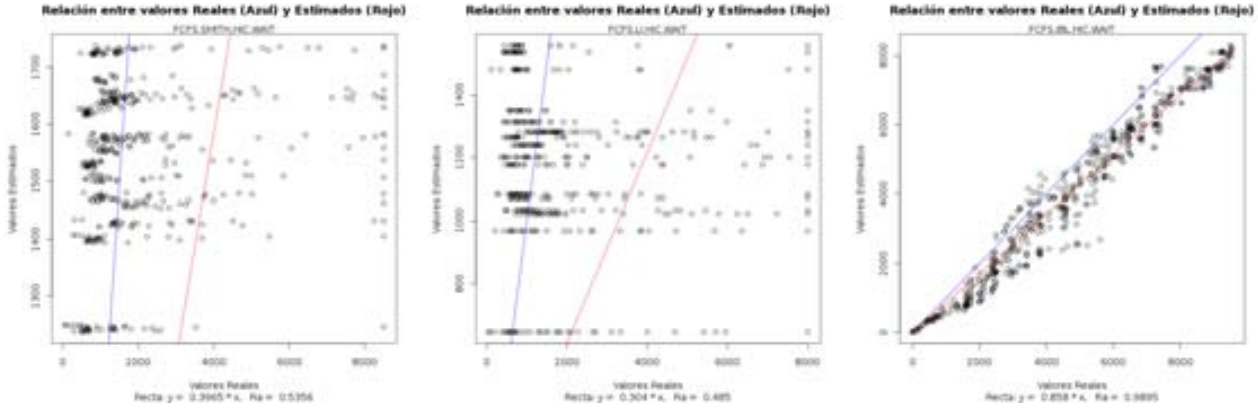
El método *Hybrid* (Tabla 3.8) al igual que el resto de estimadores analíticos presenta una gran desviación, con el inconveniente de que la dispersión es también mayor. Esto implica que en el caso de aplicar un factor corrector no se obtendrán mejores desviaciones que en otros métodos con una menor dispersión.

3.5.4.3. Entorno compartido con alta carga local

Histogramas



Gráficos de Dispersión



(a) Smith (b) LI (c) IBL

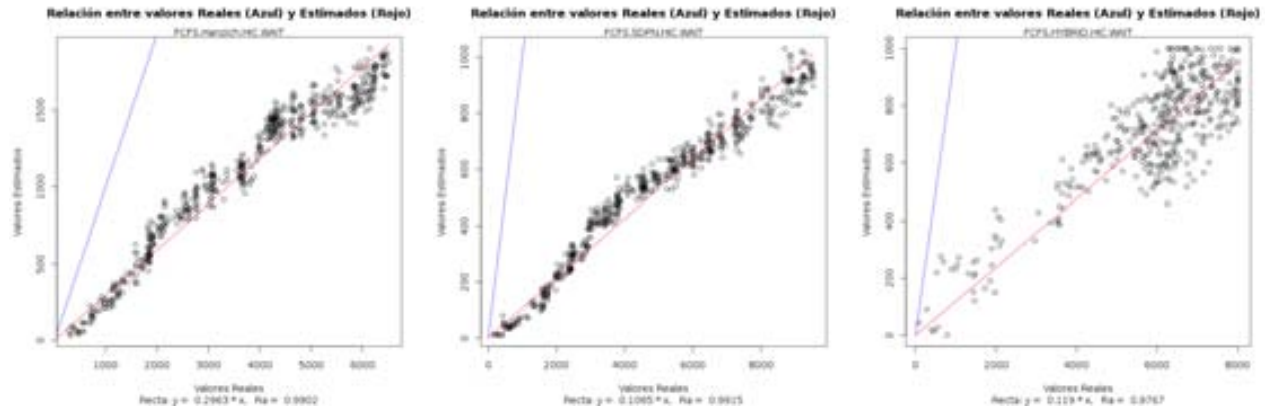
Figura 3.15: T. Espera. Entorno compartido carga local alta - Estadísticos

En un entorno compartido con alta carga local, los estimadores estadísticos (Figura 3.15) presentan un comportamiento similar al mostrado en un entorno compartido sin carga local.

En la Tabla 3.9 se observa que incluso se obtienen mejores resultados en las desviaciones. Entre todos los métodos estadísticos el que destaca con diferencia es el modelo *IBL* propuesto en el presente trabajo. Este modelo obtiene una desviación por debajo del 17 % en un entorno con alta carga. Además, ofrece una baja dispersión y una escasa distancia a la recta identidad.

Histogramas

Gráficos de Dispersión



(a) Hanzich (b) SDPN (c) Hybrid

Figura 3.16: T. Espera. Entorno compartido carga local alta - Analíticos

Los estimadores *Hanzich* y *SDPN* (Figura 3.16) muestran comportamientos muy similares. Una elevada desviación pero con muy baja dispersión. De nuevo podemos pensar en la posibilidad de aplicar un factor corrector que nos permita ajustar la recta de regresión a la recta identidad y mejorar significativamente los resultados de la desviación. En este caso el estimador que ofrezca la menor dispersión obtendrá la menor desviación.

El método *Hybrid* (Tabla 3.9) en circunstancias de alta carga obtiene una alta desviación y mayor dispersión que los otros métodos analíticos.

Estadísticas Tiempo de Ejecución			
Estimador	Pendiente Recta	Ra	Desv (%)
Smith	0.3965	0.5356	0.58
Li	0.304	0.485	0.63
IBL	0.858	0.9895	0.16
Hanzich	0.2963	0.9802	0.7
SDPN	0.1065	0.9915	0.89
Hybrid	0.119	0.9767	0.88

Tabla 3.9: Tiempos de espera en entorno compartido con alta carga

3.5.5. Conclusiones

De los resultados obtenidos en esta experimentación se desprenden interesantes conclusiones. En primer lugar, los estimadores se comportan de forma muy distinta en función del parámetro que se desea estimar, tiempo de ejecución o tiempo de espera. En general los estimadores estadísticos obtienen mejores resultados en la estimación del tiempo de ejecución que en la estimación del tiempo de espera. Esto es así porqué los estimadores estadísticos que no consideran ninguna información del sistema no disponen de suficiente información para poder determinar que sucede con el tiempo de espera. El estimador *IBL,* obtiene los mejores resultados en la estimación del tiempo de ejecución en prácticamente todos los casos, especialmente en situaciones de alta carga paralela y local.

Los estimadores analíticos presentan un mejor comportamiento en la estimación de los tiempos de espera. En general, aunque las desviaciones son grandes, las estimaciones presentan una baja dispersión y se ajustan muy bien a la recta de regresión. Esto significa que mediante el uso de un mecanismo de corrección se puede ajustar la recta de regresión a la recta identidad reduciendo enormemente la desviación y obteniendo mejores resultados que otros estimadores, que aún estando más cerca de la identidad disponen de una mayor dispersión y por tanto una mayor desviación.

El modelo Hybrid ofrece resultados muy similares a los ofrecidos por *Hanzich* y *SDPN* pero con una mayor dispersión. Los estimadores *Hanzich* y *SDPN* ofrecen resultados similares en todos los escenarios de carga. Esto es así porque el entorno experimental definido para llevar a cabo este estudio es un cluster homogéneo con un gran ancho de banda de red (Giga-Ethernet). Esto significa que los recursos de comunicación y la heterogeneidad de los recursos no influyen sobre el tiempo de ejecución, de modo que el estimador *SDPN* solo podrá decidir en función de la capacidad de cómputo del mismo modo que hace *Hanzich*, obteniendo así comportamientos similares.

Lo mismo sucede con el método *IBL*. Sorprendentemente presenta muy buenos resultados en la predicción del tiempo de espera comparado con el resto de estimadores estadísticos. Sin embargo, la información que considera del estado del cluster es la cantidad de trabajos en la colas y los ciclos de CPU que consumen, no teniendo en cuenta la heterogeneidad de los recursos.

Es necesario un entorno experimental más complejo para poder analizar con mayor detalle el comportamiento del estimador estadístico *IBL* y de los analíticos Hanzich y SDPN, bajo distintas condiciones de carga y características de los recursos.

En la siguiente sección, proponemos un mecanismo de corrección que nos permita ajustar las estimaciones en los casos en que sea posible con el fin de obtener una menor desviación y analizamos sus efectos sobre un caso práctico.

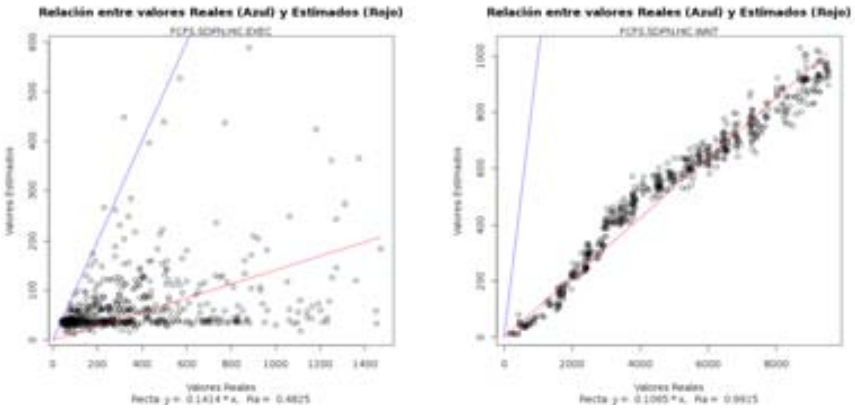
3.5.6. Mecanismos de corrección de la predicción

Supongamos que obtenemos dos predicciones como las mostradas en la figura 3.17. En el *caso A* disponemos de una gran dispersión ($Ra = 0{,}4825$) de los datos además de un gran distancia (Pendiente de la Recta: $Pe = 0{,}1414$) a la recta Identidad (Azul) que se traduce en una fuerte desviación ($Dev = 0{,}79$). Es evidente que podemos ajustar ambas rectas, la de Regresión y la Identidad. El problema radica en que continuaremos teniendo una gran dispersión sobre la recta Identidad (Azul), y valores de la desviación que aunque pueden haberse acercado a la Identidad, no proporcionan estimaciones adecuadas.

El caso B, aunque la distancia de la recta de regresión (Roja) a la recta Identidad (Azul) es incluso mayor que en el *caso A* (Pendiente de la Recta: $Pe = 0{,}0828$) se cumplen dos condiciones fundamentales. La primera es que la dispersión es muy pequeña ($Ra = 0{,}9915$) y la segunda es que el tiempo estimado crece linealmente con el tiempo real. Esto significa que aunque cometamos un gran error subestimando los valores reales, este error es siempre el mismo, de modo que el estimador se ajusta muy bien a lo que sucede en la realidad, aunque lo exprese en otra escala de valores.

Si intentamos en el *caso B* aproximar la recta de regresión a la recta Identidad, va a suceder que los valores que se ajustan bien a la recta de regresión, lo harán también a la recta Identidad, obteniendo muy buena aproximación a los valores reales.

Proponemos aplicar un factor corrector (f_c) para mejorar las estimaciones. El objetivo del factor corrector (f_c) es mover los valores de la *recta de regresión* (Roja) sobre la *recta Identidad* (Azul). Podemos expresar las ecuaciones de la recta mediante el sistema de ecuaciones 3.15, donde y_1 representa el valor estimado que



caso A caso B

Figura 3.17: Corrección en la predicción. Estudio de casos

queremos corregir, a representa la pendiente de la *recta de Regresión*, x representa el valor real que queremos estimar, y_2 es un punto sobre la recta Identidad que representa la corrección del valor y_1 y donde la pendiente de la recta identidad es 1.

$$\begin{cases} y_1 = a \cdot x & rectaRegresion \\ y_2 = 1 \cdot x & rectaIdentidad \end{cases} \tag{3.15}$$

Resolviendo el sistema de ecuaciones obtenemos que $y_2 = \frac{y_1}{a}$, de modo que para obtener la corrección de un valor estimado y_1 es suficiente con dividirlo por la pendiente de la recta de regresión a, de modo que podemos expresar el factor corrector (f_c) mediante la ecuación siguiente:

$$f(y_1) = \frac{y_1}{a} \tag{3.16}$$

La siguiente cuestión que debemos resolver es como podemos aplicar la corrección durante el proceso de estimación. En la sección siguiente se propone un método para poder aplicar las correcciones a cada nueva predicción.

3.5.6.1. Propuesta para el método de auto-corrección de las estimaciones

Para poder implementar este mecanismo necesitamos resolver dos cuestiones: cuando se debe aplicar el factor corrector, y como calcular la pendiente de la recta de regresión mientras se realizan nuevas estimaciones.

La idea es que el estimador debe almacenar información sobre las predicciones anteriores, más concretamente, la pendiente de la recta (a), el factor de dispersión (Ra) y el valor un promedio del error de predicción $avg(estim - real)$.

Para poder aplicar el factor de corrección se deben dar dos condiciones, que exista muy poca dispersión $(Ra \simeq 1)$ y que el promedio del error de predicción se mantenga prácticamente constante a lo largo de las predicciones. Cuando el promedio del error se mantiene invariable en las distintas estimaciones, significa que se comete un error sistemático y que es útil aplicar la corrección. Para tener una visión de lo variable que es el error podemos almacenar una ventana (acotada por un cierto intervalo) de los errores cometidos anteriormente y estudiar su variabilidad.

Una vez se decide aplicar el factor corrector utilizamos la pendiente de la recta (a) calculada en las estimaciones anteriores. A medida que las aplicaciones van finalizando, el sistema de predicción debe recalcular los errores cometidos y la nueva pendiente de la recta para las siguientes estimaciones.

En el caso que no existan experiencias pasadas de una determinada aplicación o situación, se deberá esperar a tener un conjunto suficiente de valores para poder estudiar su recta de regresión. Esta ventana de valores se puede determinar como parámetro de configuración del estimador.

3.5.6.2. Resultados

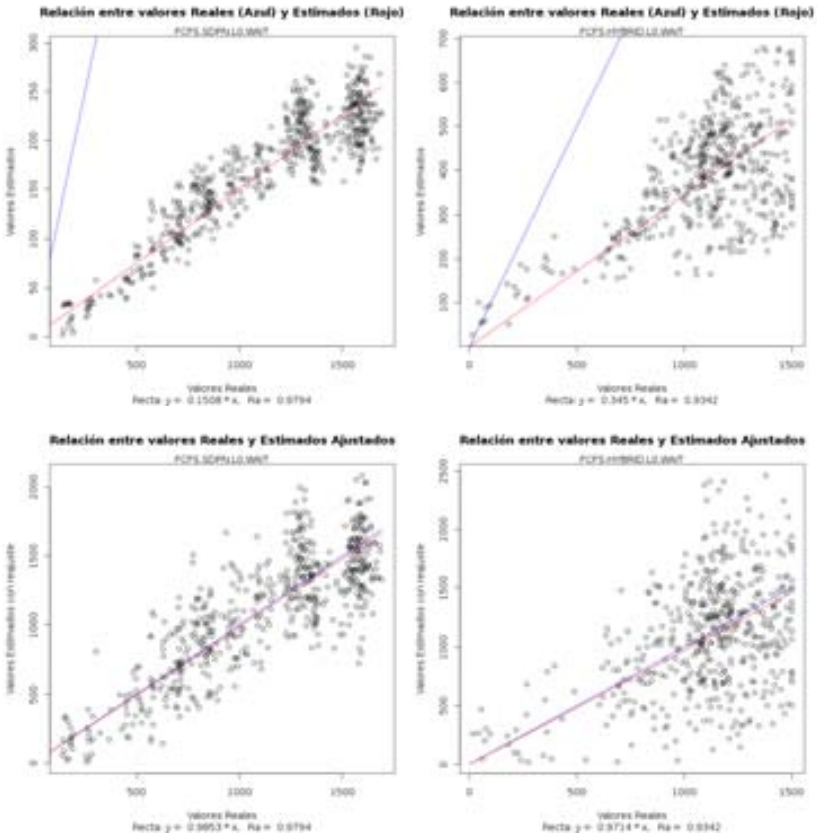
Una vez definido el método de cálculo del factor de corrección se han realizado varios experimentos para comprobar la ganancia obtenida en la predicción con la utilización de este método.

La aplicación del factor corrector se ha aplicado en los estimadores *SPDN* y *Hybrid* para la estimación de los tiempos de espera en entornos compartidos sin carga y con carga local. Las Figuras 3.18 y 3.19 muestran los resultados de aplicar el factor corrector a ambos estimadores en un entorno compartido sin carga local y un entorno con carga local alta respectivamente.

Como podemos observar en la tabla 3.10, las rectas se ajustan a la recta Identidad $(Pe \simeq 1)$. Las dispersiones se mantienen una vez aplicado el factor corrector pero las desviaciones disminuyen drásticamente. En un entorno compartido sin carga local obtenemos una ganancia en la desviación del 80 % en el estimador *SDPN* y un 50 % en el *Hybrid*. En el caso de un entorno con carga local alta, las ganancias son incluso mayores.

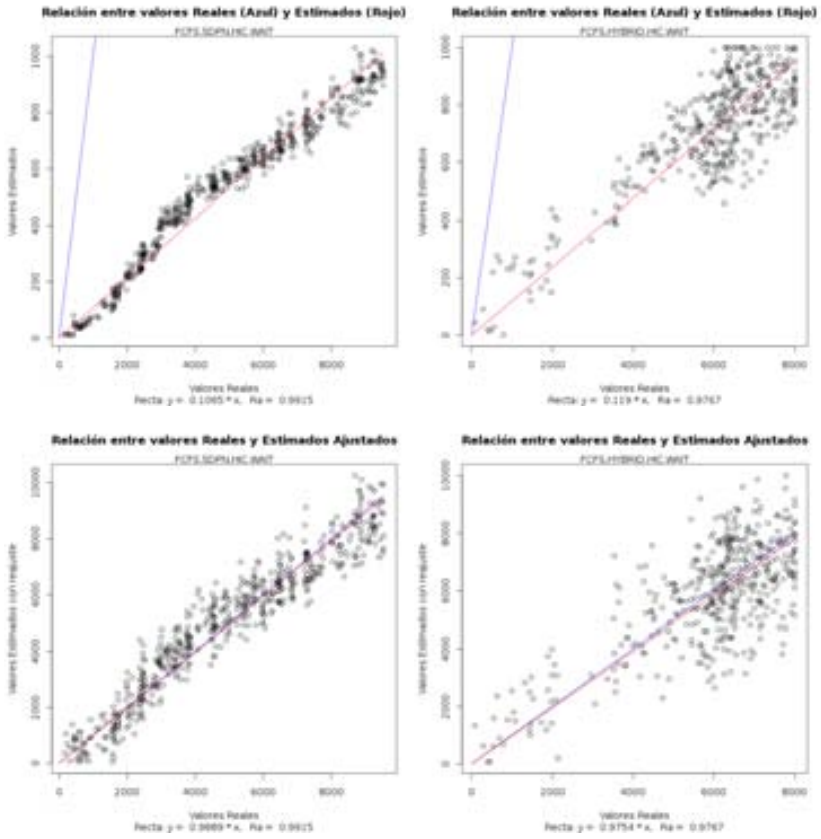
Estadísticas Tiempo de Espera							
		Estimación no corregida			Estimación corregida		
Entorno	Estimador	Pe	Ra	Desv (%)	Pe	Ra	Desv (%)
Compartido	SDPN	0.1508	0.9794	0.85	0.9853	0.9794	0.18
	Hybrid	0.345	0.9342	0.65	0.9714	0.9342	0.31
Alta Carga	SDPN	0.1065	0.9915	0.89	0.9889	0.9915	0.12
	Hybrid	0.119	0.9767	0.88	0.9754	0.9767	0.18

Tabla 3.10: Corrección en la predicción del tiempo de espera



SDPN Hybrid

Figura 3.18: Corrección del T. Espera - Entorno compartido sin carga local



SDPN Hybrid

Figura 3.19: Corrección del T. Espera - Entorno compartido con carga local alta

Estos resultados nos demuestran que no solo es una buena práctica sino necesario aplicar el factor de corrección si queremos obtener predicciones que puedan ser útiles para la planificación tanto a nivel de cluster como de Multicluster.

3.5.7. Análisis del coste algorítmico de la predicción

En la presente sección analizamos el coste algorítmico del proceso de predicción para cada estimador. El coste algorítmico de cada estimador se calcula haciendo el promedio del tiempo de respuesta del motor de predicción en todas sus estimaciones. La Tabla 3.11 muestra el coste de predicción obtenido para cada estimador.

	Estimadores					
Entorno	Smith	Li	***IBL***	Hanzich	***SDPN***	Hybrid
Dedicado	0.36	0.33	0.8	3.16	3.48	7.2
Compartido	0.4	0.37	1.05	8,85	8,95	9.7
Alta Carga	0.4	0.37	1.24	8,88	9,05	9.92

Tabla 3.11: Costes del proceso de predicción (*segundos*)

Podemos observar que el coste de predicción es mucho menor para los métodos estadísticos que para los métodos analíticos. Esto es debido a la complejidad de los algoritmos de predicción en cada caso. Los métodos estadísticos tienen una complejidad lineal $O(N_{HistDB})$, siendo N_{HistDB} el número de registros en la base de datos históricos que cumplen con los requisitos de la consulta realizada por el motor de predicción.

Los métodos analíticos calculan a cada paso de simulación el tiempo remanente de ejecución de todas las aplicaciones en la cola de ejecución (*Running List*) y esto lo hacen hasta que finalizan todos los trabajos de la cola de espera (*Low-level Queue*). Podemos expresar la complejidad de los métodos analíticos como $O(N_{LQ} \cdot N_{RL})$ siendo N_{LQ} el número de trabajos en la cola de espera y N_{RL} el número de trabajos en ejecución.

El coste de la predicción en los métodos estadísticos crece lentamente con la cantidad de registros que cumplen con los requisitos de la consulta. Los tiempos obtenidos en esta experimentación corresponden a consultas con un número máximo de 10^3 registros, obtenidos de una base de datos históricos con un volumen del orden de 5×10^4 experiencias. Se observa que los tiempos son prácticamente iguales para los métodos *Smith* y *Li* independientemente del estado de carga del sistema. Para el método *IBL*, en cambio, el coste es superior y aumenta ligeramente en función de la carga del sistema. Esto es así, porqué el modelo *IBL* utiliza información sobre el estado del sistema para realizar la predicción. Cuanto mayor es la carga del sistema mayor es el coste en obtener esta información.

El coste para los métodos analíticos es prácticamente el mismo excepto para el método *Hybrid*. Los métodos *Hanzich* y *SDPN* obtienen costes muy similares. Aunque los costes del método *SDPN* son ligeramente superiores por el cálculo del slowdown de comunicaciones la diferencia es poco significativa. El coste de predicción en un entorno dedicado es significativamente inferior que en un entorno compartido dado que al no disponer de otras aplicaciones paralelas ni locales en el sistema se simplifica mucho el cálculo del slowdown de comunicación. Para entornos compartidos los costes son prácticamente los mismos independientemente de la carga dado que el método de cálculo es exactamente el mismo.

Hybrid es el método que obtiene un mayor coste. La diferencia del método *Hybrid* con respecto al resto de métodos analíticos es que en cada paso de simulación realiza consultas a la base de datos históricos para obtener una estimación del tiempo de ejecución. Por este motivo, la complejidad de este método es la suma de las complejidades de los métodos puramente analíticos y estadísticos $O(N_{LQ} \cdot N_{RL} + N_{HistDB})$, obteniendo tiempos ligeramente superiores con respecto al resto de métodos puramente analíticos.

Aunque es importante trabajar en obtener una reducción de los tiempos de predicción de los métodos analíticos, el tiempo empleado es muy inferior a los tiempos de *turnaround* utilizados por la mayoría de aplicaciones paralelas en entornos reales [SW02, LGTW04, TEF07].

3.6. Propuesta para la predicción del tiempo de Turnaround

Analizando los resultados de precisión obtenidos por los distintos estimadores llegamos a la conclusión que ningún método predice mejor que los otros en todos los casos. Se observa que los métodos estadísticos se comportan muy bien cuando aquello que se intenta predecir tiene un comportamiento similar bajo circunstancias parecidas, como sucede por ejemplo con los tiempos de ejecución de las aplicaciones paralelas. Con el tiempo se puede almacenar un gran número de escenarios donde las aplicaciones se comportan de forma análoga y esto se refleja en los buenos resultados obtenidos en la precisión de las estimaciones.

No sucede lo mismo cuando lo que se intenta predecir ofrece comportamientos muy dispares bajo circunstancias que en principio parecen similares. Esto es lo que sucede con los tiempos de espera, donde circunstancias que se han definido como similares no son representativas en muchos casos del comportamiento del sistema y eso se refleja directamente en los errores en la predicción. Son muchos los motivos que pueden producir este efecto, en primer lugar la política de planificación puede realizar asignaciones muy distintas para situaciones aparentemente similares. Otro problema todavía más importante es como decidir que dos circunstancias son similares. El conjunto de parámetros que definen el estado del sistema es muy grande y la selección de aquellos que sean más representativos ha sido el centro de interés de un gran número de estudios. La conclusión que se extrae de la mayoría de estos trabajos es que estos valores dependen directamente del entorno de producción, por lo que se deben hacer reajustes para cada caso particular.

Los métodos de simulación, por otro lado, estiman el tiempo de ejecución reproduciendo la evolución de las aplicaciones y la disponibilidad de los recursos. Como

hemos podido comprobar, este método ofrece peores resultados en la estimación del tiempo de ejecución que los métodos estadísticos. Uno de los motivos es que en cada paso de simulación se aproxima la disponibilidad de los recursos basándonos en valores aproximados de ocupación. Estas aproximaciones degeneran a medida que avanza la simulación y esto se ve reflejado en una menor precisión en las estimaciones. Otro motivo es que el método de simulación no considera la llegada en un futuro de aplicaciones que compartirán los recursos con la aplicación objeto de la predicción y que perturban los tiempos de ejecución estimados. En cambio, no sucede lo mismo con los tiempos de espera que aunque también generan grandes desviaciones, éstas guardan una fuerte correlación lineal. Aplicando correctamente métodos correctores podemos obtener una aproximación suficientemente razonable a los tiempos de predicción reales.

Sin tener en cuenta las posibles mejoras que se pueden introducir en los distintos métodos de predicción disminuyendo así la desviación en la predicción, es evidente que cada método ofrece ventajas dependiendo de la métrica que se estima. Esto nos lleva a considerar la posibilidad de utilizar un esquema combinado de los estimadores que obtienen los mejores resultados en cada una de las métricas.

A continuación se propone un método de predicción del tiempo de *turnaround* que utiliza una combinación de los estimadores que obtienen los mejores resultados en la precisión de la predicción en cada una de la métricas a predecir, el tiempo de espera y tiempo de ejecución de las aplicaciones paralelas.

3.6.1. Esquema de predicción combinado

La idea básica es predecir el tiempo de *turnaround* de una aplicación paralela que llega al motor de predicción utilizando un estimador para la predicción del tiempo de espera y otro estimador para la predicción del tiempo de ejecución.

Tal como puede verse en la Figura 3.20, nuestra propuesta consiste en incorporar al motor de predicción un nuevo módulo ***Decisor***. Este nuevo módulo es el encargado de decidir el estimador más adecuado mediante un análisis previo de la precisión obtenida por cada estimador en las predicciones anteriores del tiempo de espera y ejecución de una aplicación paralela.

El módulo ***decisor***, a la llegada de una nueva aplicación paralela comprueba la desviación en la predicción del tiempo de espera y/o ejecución de los distintos estimadores, seleccionando aquellos que obtienen mayor precisión. A partir de las estimaciones del tiempo de espera y/o ejecución obtenidas por los estimadores seleccionados, el ***decisor*** genera una nueva estimación para el tiempo de *turnaround* de la aplicación paralela con la mínima desviación posible.

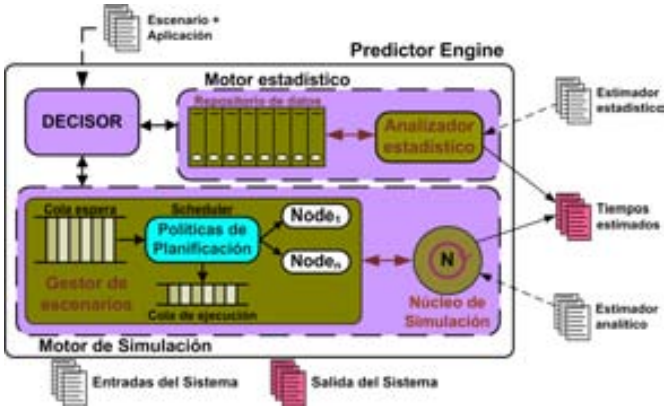


Figura 3.20: Predictor Engine con Decisor

La desviación en la predicción de cada estimador se calcula promediando un subconjunto ζ de estimaciones anteriores. El tamaño máximo del subconjunto de ζ estimaciones anteriores se define como una parámetro auto-ajustable denominado α.

El ***decisor*** puede auto-ajustar el tamaño del subconjunto ζ de estimaciones previas a considerar en el cálculo de la desviación, de modo que no se supere la *cota de desviación aceptable*. El valor de la *cota de desviación* aceptable es un valor que define el administrador del sistema en función de los valores de precisión deseados.

Capítulo 4

Sistema Meta-planificador

La tarea de meta-planificación consiste en decidir qué aplicación será ejecutada a continuación y en qué recursos. Aunque la selección de las aplicaciones y recursos ha sido ampliamente estudiada en los sistemas paralelos y distribuidos tradicionales [Fei97], sistemas multiprocesador simétricos (SMPs), computadores paralelos masivos (MPPs) y clusters (NOW), hay ciertas consideraciones que no son válidas en entornos Multicluster no dedicados, donde existe una gran cantidad de recursos de naturaleza muy distinta distribuidos en clusters y compartidos entre varios usuarios. En un entorno de estas características, se hace evidente la necesidad de disponer de un sistema de meta-planificación encargado de gestionar gran cantidad de recursos distintos y seleccionar los más adecuados para cada aplicación paralela.

La tarea de meta-planificación es una tarea compleja formada por un conjunto de subtareas importantes que definen el propósito de nuestro sistema. Algunas de estas tareas son la selección de recursos, la selección de trabajos a ejecutar, establecer el límite de tamaño adecuado en las colas de cada cluster, la gestión de la co-asignación, etc. Proponer una solución a la meta-planificación significa proponer soluciones en las distintas fases de la tarea de meta-planificación. El objetivo de la meta-planificación en el presente trabajo es obtener el máximo rendimiento de las aplicaciones paralelas sin perjudicar al usuario local. En el presente capítulo proponemos soluciones a las distintas fases de la tarea de meta-planificación con el fin de cumplir ese objetivo.

En la sección 4.1 presentamos algunas consideraciones previas que nos ayudan a contextualizar nuestras propuestas de meta-planificación dentro de la literatura. En la sección 4.2 se presentan los principales pasos de la tarea de meta-planificación y se sitúan las distintas propuestas realizadas. En la sección 4.3 se propone una solución para la *selección de recursos* basada en predicción. En la sección 4.4 se propone un refinamiento para la *selección de trabajos* que permite la re-asignación

de aplicaciones paralelas en función de los cambios de estado del sistema. En la sección 4.5 se propone un método para establecer el *límite de tamaño adecuado de la cola* de cada cluster en función de su nivel de carga. En la sección 4.6 se propone un método para la asignación de recursos a aplicaciones que requieren *co-asignación* y finalmente en la sección 4.7 se presenta un refinamiento a la *tarea de meta-planificación* que amplía el rango de aplicaciones co-asignadas bajo determinados criterios con el fin de obtener un mayor rendimiento.

4.1. Consideraciones de la meta-planificación

En el presente trabajo hemos optado por una ***organización jerárquica*** del sistema de planificación. Esto está motivado por el hecho que disponemos de distintos clusters, cada uno de ellos con su propio sistema de planificación y necesitamos un nivel superior de planificación (*Meta-Planificador*) que mediante una visión global de todo el Multicluster permita aprovechar al máximo los recursos disponibles y obtener un mayor rendimiento del sistema y de las aplicaciones paralelas. En [LSG+06] demostramos los buenos resultados de rendimiento obtenidos con el uso de una arquitectura jerárquica con respecto a una arquitectura totalmente centralizada. Otros muchos estudios de la literatura avalan esta decisión [TRA+05, EHS+02, BE00, BE02, SCJG00].

Otra decisión que se debe tomar es si la planificación de las aplicaciones se debe restringir al cluster donde se generan, meta-planificando únicamente aquellas que requieren más recursos de los disponibles en el cluster donde han sido generadas, o bien es preferible permitir la ***meta-planificación de todas las aplicaciones paralelas*** que llegan a cualquiera de los clusters.

Bucur afirma en [BE02] que se obtienen mejores resultados de rendimiento permitiendo la ejecución de las aplicaciones paralelas en cualquier cluster y no restringiendo su ejecución al cluster donde han sido generadas. En el presente trabajo consideramos que todas las aplicaciones paralelas que se lanzan en el sistema, llegan a la cola del meta-planificador para obtener la mejor planificación posible. En la sección 4.2 se describen los pasos principales de la *tarea de meta-planificación*.

La ***tarea de meta-planificación*** consiste principalmente en decidir qué aplicación será ejecutada a continuación y en qué recursos. La *tarea de meta-planificación* se divide en dos subtareas, la *selección de recursos* y la *selección de aplicaciones*.

En cuanto a la tarea de ***selección de recursos,*** existe un gran número de propuestas clasificadas en dos grandes grupos, la *planificación a nivel de recursos* que trata de mejorar su utilización [ELvD+96, Xu01, RL04] y la *planificación a nivel de aplicación*, donde el objetivo principal es mejorar el tiempo de *turnaround* de las apli-

caciones paralelas [CFK+98, AFKT00, BE03b, HJSN04, CC06]. Otras propuestas intentan cumplir con ambos objetivos al mismo tiempo sin considerar la interactividad del usuario local [HSSY00, KNP01, JSK+06, KNOW08]. En la sección 4.3, se propone una técnica de ***selección de recursos*** que considera tanto la estimación del tiempo de respuesta de las aplicaciones paralelas como su efecto sobre el usuario local.

En cuanto a la tarea de ***selección de aplicaciones*** nos hemos centrado en la utilización de las políticas más comunes en entornos Multicluster, *FCFS* (***F**irst **C**ome **F**irst **S**erved*) y *FPFS* (***F**it **P**rocessors **F**irst **S**erved*), ambas introducidas en el capítulo 2. Sin embargo, en un entorno de planificación jerárquico con una cola a nivel global y colas a nivel de cluster debemos resolver varias cuestiones, cual es el límite óptimo de las colas locales y como se debe proceder cuando una aplicación es asignada a un cluster con la cola de espera llena. En la sección 4.4, se propone un refinamiento de ambas políticas que permite definir cuando una aplicación paralela debe permanecer en la cola de espera del *Meta-Planificador* y como puede ser desbloqueada y asignada a un cluster para su ejecución. En la sección 4.5 se propone un método para determinar el tamaño de la colas de cada cluster de forma dinámica según el estado de ocupación del cluster.

En un entorno Multicluster existe un particionamiento natural de los recursos en distintos clusters, donde cada uno de ellos puede disponer de recursos de distinta naturaleza y de su propia política de planificación local. El *Meta-Planificador* debe considerar las características particulares de cada cluster a la hora de decidir dónde asignar la siguiente aplicación paralela. Aunque este es el modelo utilizado en la mayoría de sistemas Multicluster [Xu01, SvANS00, AD03, JAA06], existe un gran número de estudios [SCJG00, BE02, EHS+02, BE00, BE01, JLPS05] que demuestran que compartiendo recursos entre distintas particiones (*clusters*) se puede aumentar la utilización de los recursos y mejorar el rendimiento de las aplicaciones paralelas y del sistema en general.

En ocasiones pueden llegar al sistema aplicaciones con fuertes requerimientos de recursos, de modo que no existe ningún cluster en el sistema con suficientes recursos para ejecutar la aplicación de forma individual. En estos casos una posible solución es la compartición de recursos entre distintos clusters. Esta técnica se conoce con el nombre de *co-asignación*. Se ha demostrado que la compartición de recursos permite ejecutar aplicaciones con grandes requerimientos aumentando al mismo tiempo la utilización de los recursos del sistema.

Para obtener un buen rendimiento con la utilización de la *co-asignación* necesitamos disponer de una técnica de *selección de recursos* que considere las características y el estado de los canales de comunicación entre clusters. La mayoría de estudios previos [BAG00, Wol03, HHL+06] realizan la tarea de selección conside-

rando la capacidad de los recursos de cómputo sin prestar atención a los recursos de comunicación. Otros muchos trabajos [DZ97, BE03b, JA05] proponen modelos de comunicación que permiten valorar mediante simulación el rendimiento del sistema asumiendo valores predeterminados de tamaño de mensaje y latencias. Estos métodos no son útiles para tomar decisiones on-line sobre la *selección de recursos* considerando las características reales de cada aplicación y el estado actual del sistema. En la sección 4.6 se propone un método de la *selección de recursos* en entornos Multicluster, basado en la utilización de un ***modelo de programación entera*** (*Mixed Integer Programming*) que permite obtener una aproximación de la mejor asignación posible considerando el estado de los recursos tanto de cómputo como de comunicación de los distintos clusters.

En [BE02, JLPS05] se demuestra que los canales de comunicación entre clusters pueden formar un cuello de botella y perjudicar enormemente el rendimiento del sistema si no se limita la *co-asignación* a aplicaciones con bajos requisitos de comunicación o situaciones con baja probabilidad de saturación de los canales de comunicación. En la sección 4.7 se propone un refinamiento a la *meta-planificación* donde la co-asignación no se limita únicamente a las aplicaciones con grandes requerimientos, sino que se aplica también a las aplicaciones a la espera de recursos libres en algún planificador local (*cluster*) y que pueden ser co-asignadas sin saturar los canales de comunicación.

4.2. Mecanismo de meta-planificación

El objetivo principal de la tarea de meta-planificación es seleccionar para cada aplicación paralela que llega a la cola de *MetaLoRaS* el conjunto de recursos más adecuado para llevar a cabo su ejecución. Esto significa seleccionar el conjunto de recursos que permita obtener el máximo rendimiento de las aplicaciones paralelas sin perjudicar la interactividad del usuario local.

El *Algoritmo 4.1* detalla los principales pasos de la tarea de meta-planificación que lleva a cabo *MetaLoRaS*. Las tareas se insertan en la *Upper-Level Queue* según un estricto orden de llegada. La tarea de meta-planificación se activa a la llegada de una nueva aplicación, cuando una nueva aplicación se asigna a un cluster para ser ejecutada o cuando ésta termina.

El meta-planificador selecciona el siguiente trabajo (J) de la *Upper-Level Queue* según el mecanismo de *selección de trabajos* **(paso 1)**. Los mecanismos de *selección de trabajos* utilizados en el presente trabajo se basan en las políticas *FCFS* y *FPFS*. En la sección 4.4 se propone un refinamiento de ambas políticas.

Algorithm 4.1 Meta-Scheduling algorithm

1: Choose the next job (J) from the *Upper-level Queue.*
2: Obtain a list of Resources that satisfies the job J software requirements.
3: If the list is empty, reject the job J and **Go to** Step 1
4: Obtain a list of Clusters that have enough resources to execute the job J.
5: If the list is empty, **co-allocation** and **Go to** Step 7
6: Obtain the Cluster C to map job J according to the **Prediction Engine**.
7: Dispatch Job J.
8: **Go to** Step 1

Una vez seleccionada la aplicación paralela (J), el meta-planificador consulta al *Admision System*, preguntando si los requisitos software (SO, tipología de los recursos, librerías, etc) de la nueva aplicación pueden ser satisfechos por los recursos del Multicluster **(paso 2)**. En caso contrario, la aplicación es eliminada del sistema y se vuelve a seleccionar una nueva aplicación **(paso 3)**.

Cuando la aplicación paralela (J) no es rechazada por el *Admision System*, el Meta-Planificador obtiene a través del *Monitoring Toolkit* una lista de clusters con suficientes nodos para ejecutar la nueva aplicación (J) **(paso 4)**. Si la lista está vacía, significa que la aplicación paralela (J) no puede ser ejecutada sin la compartición de recursos entre distintos clusters. Entonces, aplicamos la ***co-asignación*** intentando ejecutar la aplicación (J) aprovechando los recursos libres de los distintos clusters sin saturar los canales de comunicación entre clusters **(paso 5)**, tal como se describe en la sección 4.6.

En aquellos casos en que existan varios clusters con suficientes nodos para ejecutar la aplicación (J), el Meta-Planificador deberá seleccionar el cluster más adecuado para su ejecución en función de las características de la aplicación y del estado de los recursos **(paso 6)**. Para tomar esta decisión nos basamos en las estimaciones del tiempo de *turnaround* obtenidas a través del ***Prediction Engine*** de los distintos clusters candidatos, tal como se describe en la sección 4.3.

Una vez seleccionado el cluster encargado de la ejecución de J, el Meta-Planificador le entrega la aplicación **(paso 7)** y busca en la *Upper-Level Queue* un nuevo trabajo para ser planificado **(Paso 8)**. En los casos en que se aplica *co-asignación*, son varios los clusters encargados de ejecutar la aplicación J.

Para llevar a cabo la co-asignación, se propone la creación de un cluster virtual formado por los recursos libres de cada cluster, encargado de ejecutar las aplicaciones paralelas co-asignadas en recursos de distintos clusters. El cluster virtual nos permite gestionar la co-asignación de un forma flexible, ocupándose del mantenimiento de los recursos disponibles, la ejecución de las aplicaciones co-asignadas y la búsqueda de recursos disponibles.

4.3. Propuesta de Selección de clusters basada en predicción

Muchas de las heurísticas utilizadas hasta el momento [DA06] para la selección de clusters presentan algunos inconvenientes. En muchos casos solo funcionan correctamente cuando los recursos son homogéneos y el número de recursos en cada cluster es similar [HSSY00, EHYS02]. Además no se considera la evolución en la utilización de los recursos en un futuro inmediato. Esto significa que la toma de este tipo de decisiones en entornos heterogéneos y no dedicados requiere de mecanismos más sofisticados que consideren tanto la heterogeneidad como la posible evolución en el estado de los recursos.

Subramani en [SKSS02] presenta un técnica mediante la cual se distribuyen varias copias de un mismo trabajo a distintas colas locales. Cuando un trabajo se inicia en uno de los cluster envía una notificación para cancelar la ejecución del resto de copias del trabajo. El objetivo de esta técnica es mejorar la utilización de los recursos del sistema y disminuir el tiempo de espera de las aplicaciones en las colas, pero no están orientadas a mejorar el rendimiento de las aplicaciones paralelas ni tienen en cuenta la carga de los usuarios locales. Otros autores [SCBG03] mantienen varias copias de una misma aplicación ejecutándose al mismo tiempo hasta que la primera de ellas finaliza. Estas técnicas tampoco estiman el rendimiento a la hora de realizar las réplicas de los trabajos.

Otros muchos estudios [Wol03, LS05, HGH$^+$06] presentan modelos analíticos que estiman el rendimiento basándose únicamente en la evolución de los recursos de cómputo no prestando ninguna atención a las comunicaciones, cuya influencia no puede ser despreciada cuando tratamos de sistemas distribuidos posiblemente con recursos de comunicación heterogéneos [BE01, EHYS02, JLPS05].

En el capítulo 3 presentamos un motor de predicción que estima el tiempo de *turnaround* de una aplicación paralela en un cluster en función de las características y la ocupación de los recursos tanto de cómputo como de comunicación.

En la presente sección proponemos dos mecanismos de selección de clusters distintos, el *Turnaround Time* (*TAT*) y *Ponderated Turnaround Time* (*PTT*), ambos basados en la estimación del tiempo de *turnaround* que cada cluster proporciona al meta-planificador. El primer mecanismo *(TAT)* se enmarca en el conjunto de técnicas de planificación centradas en la aplicación, ya que se centra únicamente en mejorar el rendimiento de las aplicaciones paralelas seleccionando el cluster que devuelve el tiempo mínimo de *turnaround*. El segundo mecanismo *(PTT)* en cambio, incorpora información sobre la cantidad de usuarios locales que hay en los clusters, seleccionando el cluster con el mínimo tiempo de *turnaround* ponderado

por la cantidad de carga local. Este mecanismo permite buscar un equilibrio entre el rendimiento de la aplicación paralela y la interactividad que puede requerir el usuario local. El valor de ponderación es un valor configurable por el administrador del sistema para cada cluster por separado y que puede ser ajustado en función de las necesidades del usuario local en cada caso.

Es importante destacar que, en este estudio, ambas políticas recorren todo el espacio de posibles soluciones (Clusters) en busca de la estimación del tiempo mínimo de *turnaround*. Esta opción es razonable cuando se considera un Multicluster compuesto por un conjunto no muy grande de clusters. Cuando el espacio de posibles soluciones crece, el tipo de búsqueda de la mejor solución puede influir negativamente en el rendimiento del sistema.

4.3.1. Planificación a nivel de aplicación (TAT)

Nuestra primera propuesta es un método orientado exclusivamente a mejorar el rendimiento de las aplicaciones paralelas. El mecanismo de *Turnaround Time (TAT)* selecciona el cluster que devuelve el tiempo estimado de *turnaround* mínimo. En el *Algoritmo 4.2* se detallan los pasos que se llevan a cabo mediante el uso de *TAT*.

El mecanismo de selección de clusters (*TAT*) recibe de la tarea de meta-planificación una lista de los clusters que cumplen con los requisitos software de la aplicación J que se desea ejecutar y que disponen de suficientes nodos. A continuación, se selecciona el cluster donde se estima el menor tiempo en finalizar la aplicación. Para ello se recorre la lista de posibles candidatos solicitando a *Prediction Engine* de cada uno de ellos la estimación del tiempo de *turnaround* para la aplicación J (paso 1).

Algorithm 4.2 *TAT* Cluster Selection Algorithm.

Require: List of clusters that satifies the job J software requirements and have enough nodes.

1: MetaScheduler request the *turnaround time* of Job J to each cluster k where the job can be executed ($TR_k(J)$).
2: Metascheduler selects the cluster C where $TR(J) = min_{k=1}^{n} \{TR_k(J)\}$, is reached.

El tiempo de *turnaround* contempla tanto el tiempo de ejecución como el tiempo de espera a que haya suficientes recursos disponibles en el cluster correspondiente. Una vez obtenida la lista de todas las estimaciones, el mecanismo de selección escoge el cluster que devuelve la menor estimación del menor tiempo de *turnaround* (paso 2).

En este punto se hace evidente que la precisión de los mecanismos de predicción es de vital importancia para llevar a cabo decisiones que permitan aumentar el rendimiento de las aplicaciones paralelas y del sistema en general. En este sentido es importante seguir trabajando en la mejora de la precisión en la predicción de modo que los estimadores sean capaces de representar con mayor precisión la realidad del entorno donde se llevan a cabo las predicciones, obteniendo así una mayor precisión en las predicciones y por tanto un mayor rendimiento de las aplicaciones y del sistema.

4.3.2. Planificación a nivel de recursos (PTT)

En algunos casos puede suceder que el usuario local tenga unos requisitos muy exigentes de interactividad y ocupación de los recursos, aplicaciones multimedia, Vídeo, Juegos etc. Aunque cada cluster reserva unos recursos mínimos para el usuario local, esto puede no ser suficiente cuando los requisitos son tan elevados. En estos casos necesitamos disponer de mecanismos de selección que tengan en cuenta la cantidad de carga local a la hora de repartir las aplicaciones paralelas.

En estos casos el mecanismo *TAT* presenta algunos inconvenientes ya que se centra únicamente en buscar el cluster que proporciona el menor tiempo de *turnaround* sin considerar la cantidad de carga local. El objetivo principal de esta nueva propuesta es realizar un reparto equilibrado de las aplicaciones paralelas considerando la cantidad de carga local en los clusters.

Para cumplir este objetivo el mecanismo de selección propuesto en esta sección, denominado *Ponderated Turnaround Time* (*PTT*), considera tanto la estimación del tiempo de *turnaround* como la cantidad de nodos con carga local de cada cluster. En el *Algoritmo 4.3* se detallan los pasos que se llevan a cabo mediante el uso de *PTT* para la obtención del cluster donde se asignará el trabajo paralelo J.

Algorithm 4.3 *PTT* Cluster Selection Algorithm.

Require: List of clusters that satifies the job J software requirements and have enough nodes.

1: Metascheduler request the *turnaround time* of Job J in each cluster k where the job can be executed ($TR_k(J)$) and the local resource occupancy ($LT_k, \forall k = 1..n$).
2: Metascheduler computes $PTT_k(J), \forall k = 1..n$ (see equation 4.1).
3: Metascheduler selects the cluster C where $PTT(J) = min_{k=1}^{n} \{PTT_k(J)\}$, is reached.

La principal diferencia entre TAT y PTT la encontramos en el **paso 1**, donde además de proporcionar la estimación sobre el *turnaround time* se proporciona información sobre la cantidad de nodos que disponen de carga local. Esta métrica trata de clasificar los clusters teniendo en cuenta de forma simultanea, el tiempo estimado de *turnaround* y la cantidad de usuarios locales que pueden verse perturbados. El parámetro w permite configurar la prioridad que el planificador debe dar a cada uno de los criterios.

A partir de esta información, mediante la ecuación 4.1, se realiza el cálculo del *Tiempo de Turnaround Ponderado* (*Ponderated Turnaround Time* $PTT(J)$) para la aplicación J. Sea C un Multicluster compuesto de n clusters. El *Tiempo de Turnaround Ponderado* ($PTT_k(J)$) para un cluster $k \leq n$ y un trabajo J, se define como:

$$PTT_k(J) = TR_k(J)\left(w\frac{TR_k(J)}{MaxTR(J)} + (1-w)\frac{LT_k}{MaxLT}\right), \tag{4.1}$$

donde $TR_k(J)$ es el tiempo de turnaround estimado en el cluster k, $MaxTR(J)$ es el tiempo máximo de *turnaround* estimado en los clusters candidatos, LT_k es el número de nodos asignados al trabajo J en el cluster k con actividad local y finalmente, $MaxLT$ es el máximo número de nodos asignados al trabajo J con actividad local en el conjunto de clusters candidatos. Para evitar una división por 0, $MaxLT$ se pone a 1 cuando no hay actividad local ($MaxLT$=0). w (un valor real entre [0..1]) es el peso asignado en la fórmula al tiempo de *turnaround*. $(1-w)$ es por tanto el peso asignado a la ocupación de los recursos por las aplicaciones locales.

Una vez obtenidos los $PTT_k(J)$ para todos los clusters candidatos, el mecanismo de selección escoge el cluster con el mínimo tiempo de turnaround ponderado, tal como se muestra en la ecuación 4.2.

$$PTT(J) = min_{k=1}^{n}\{PTT_k(J)\} \tag{4.2}$$

Con la utilización de este mecanismo conseguimos que la tarea de meta-planificación asigne el trabajo J al cluster que proporcione el mínimo tiempo de *turnaround* ralentizando lo menos posible a las aplicaciones del usuario local. Por otro lado, variando w podemos obtener distintos resultados en la planificación. Si se desea que los trabajos paralelos no perturben en lo más mínimo las aplicaciones del usuario local, definimos $w \simeq 0$. Por el contrario, si es más importante el rendimiento de las aplicaciones paralelas definimos $w \simeq 1$. Para valores de $w \simeq 0{,}5$, indicamos que deseamos un equilibrio entre la intrusión a las aplicaciones locales y el tiempo de turnaround de los trabajos paralelos.

Este mecanismo permite al administrador del sistema definir en cada cluster valores distintos de w en función de las necesidades del usuario local. Además, es posible

la implantación de mecanismos que ajusten de forma automática los valores de *w* en función del uso de los recursos por parte del usuario local.

4.3.3. Experimentación

En esta sección comparamos los mecanismos de selección de clusters propuestos *TAT* y *PTT* con otros mecanismos de selección utilizados comúnmente en la literatura, *FFIT* y *RR*. *FFIT* (*Firts Fit*) selecciona el primer cluster con suficientes nodos para ejecutar la aplicación paralela, sin tener en cuenta ni la carga de los recursos ni la cantidad de usuarios locales que se encuentran en cada cluster. *RR* (*Round Robin*) selecciona de forma secuencial los diferentes clusters siempre y cuando el cluster tenga suficientes nodos para ejecutar la aplicación. *TAT* (*Turnaround Time*) selecciona el cluster que devuelve la menor estimación del tiempo de *turnaround* y *PTT* (*Ponderated Turnaround Time*) selecciona el cluster considerando la estimación del *Turnaround Time* y la cantidad de recursos con carga local de cluster, según la ecuación 4.1. El parámetro *w* en la presente experimentación se ha configurado con valores de [0,0.5,1]

En los experimentos realizados se evalúa el rendimiento de las aplicaciones paralelas, la intrusión al usuario local y el rendimiento global del sistema bajo distintas condiciones de carga local y distribución de nodos en los clusters.

La carga local es introducida en el sistema a través de un benchmark desarrollado en nuestro grupo de investigación, denominado *local_bench*. Este benchmarck emula la utilización de CPU, Memoria y tráfico de red mediante valores parametrizables, tal como se describe en el apéndice D.1.2. El perfil de usuario local escogido que denominaremos *XWindows+Internet* ha sido modelado con los parámetros de uso siguientes, un 15 % de uso de CPU, 35 % de Memoria y un Bandwidth de 0,5MB/sec. Los experimentos han sido realizados para distintos niveles de carga local (0 %, 30 %, 60 %, 100 %) distribuida uniformemente entre todos los nodos del multicluster o de forma agrupada en determinados clusters.

El conjunto de aplicaciones paralelas se ha constituido a partir de dos benchmarks de la NAS bien conocidos, el IS y el MG. En total se han construido 18 aplicaciones paralelas, 8 orientadas a la comunicación intensiva (IS) y 8 al cómputo intensivo (MG), cada una de ellas con distintos requerimientos de tareas (2, 4 y 8) y con distinta clase de utilización de recursos (A, B y AB), donde cada clase especifica distinto grado de utilización de los recursos de cómputo, memoria y tráfico de red. En el apéndice D.1.1 se describe con detalle las características de cada aplicación.

El *workload paralelo* está formado por un total de 60 aplicaciones escogidas de forma aleatoria entre el conjunto de aplicaciones paralelas definido anteriormente,

que llegan al sistema bajo una distribución de Poison con un promedio de llegada, $\mu = 6sec$. El nivel de paralelismo alcanzado en los nodos de cada cluster dependerá del estado de carga del sistema pero está limitado a un máximo de 4 aplicaciones paralelas $(MPL \leq 4)$, con el fin de preservar la utilización de los recursos por parte del usuario local.

Otro parámetro importante que se ha variado es la configuración de los clusters que componen el Multicluster. Disponemos de 16 nodos con las mismas características, donde se han realizado dos tipos de configuración distintas. La primera, denominada 555, está compuesta de 3 clusters con 5 nodos cada uno. La segunda, denominada 844, se compone de 3 clusters, uno con 8 nodos y 2 clusters de 4 nodos.

Métricas

La comparación de los distintos métodos de selección de clusters se ha realizado utilizando 3 métricas distintas, el *Turnaround*, el *Makespan* y el *Slowdown*.

El tiempo de ***Turnaround*** mide el tiempo que transcurre desde que la aplicación es entregada a un cluster hasta que finaliza su ejecución. Esta métrica permite evaluar los mecanismos de selección de clusters desde el punto de vista del rendimiento de las aplicaciones paralelas.

El ***Makespan*** proporciona una medida del tiempo transcurrido desde que llega la primera aplicación paralela al Multicluster hasta la finalización de la última. Esta métrica nos permite evaluar los mecanismos de selección desde el punto de vista del rendimiento global del sistema.

Otro de los parámetros significativos que deseamos evaluar es la intrusión de la carga paralela sobre el usuario local. Para medir esta intrusión utilizamos el mismo benchmark utilizado para introducir la carga local en el sistema, el *local_bench*. Este benchmark se encarga de tomar muestras de la latencia (tiempo de respuesta) media durante su ejecución.

El ***Slowdown*** (Sl) se define, según la ecuación *4.3*, como la relación entre la latencia obtenida durante la ejecución de la carga paralela $AvLat_{non-dedicated}$ y la latencia obtenida en un entorno con solo carga local $AvLat_{dedicated}$.

$$Sl = \frac{AvLat_{non-dedicated}}{AvLat_{dedicated}} \tag{4.3}$$

Un valor de Sl próximo a 1 significa que las técnicas utilizadas no aumentan los valores de latencia y por lo tanto no afectan negativamente la interactividad del usuario local. Valores de Sl mayores que 1 significa que las técnicas de planificación utilizadas realizan una gestión de los recursos que provoca ralentización de las

aplicaciones locales, y que por tanto afectan negativamente a la interactividad del usuario local.

Análisis del tiempo de *Turnaround*

La métrica de *Turnaround* nos proporciona el promedio del tiempo de *turnaround* de todas las aplicaciones paralelas ejecutadas. Cuanto más bajo sea este promedio mejor será el rendimiento de las aplicaciones paralelas y en consecuencia el rendimiento general del sistema.

La figura 4.1 muestra los resultados obtenidos cuando la cantidad de nodos es la misma para todos los clusters del Multicluster. La Figura 4.2 muestra los resultados obtenidos cuando la cantidad de nodos no es la misma para todos los clusters. En cada figura, a la derecha se muestran los resultados cuando la carga local se distribuye uniformemente entre los clusters, y a la izquierda se observan los resultados cuando la carga local es agrupada en determinados clusters. Para cada gráfica se modifica la cantidad de carga local y el mecanismo de selección utilizado.

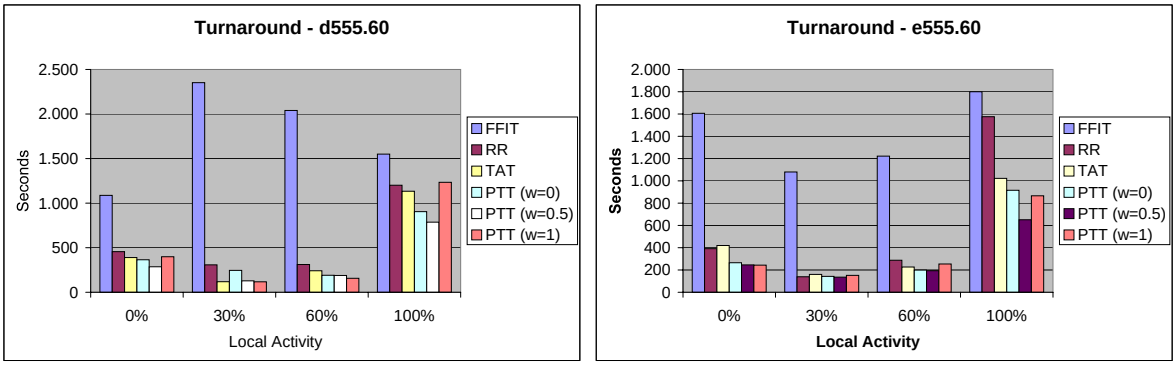


Figura 4.1: Evaluación del *Turnaround Time* (mismo número de nodos x cluster)

En la figura 4.1 se observa como las políticas basadas en predicción se comportan mejor en todos los casos que las políticas clásicas, obteniendo un beneficio promedio de un 43.3 % respecto la política *RR*, que es la política clásica que mejores resultados obtiene. En la figura 4.2, se observa que los beneficios no son tan buenos pero aún así se consigue un beneficio respecto a la política *RR* de un 29.9 %.

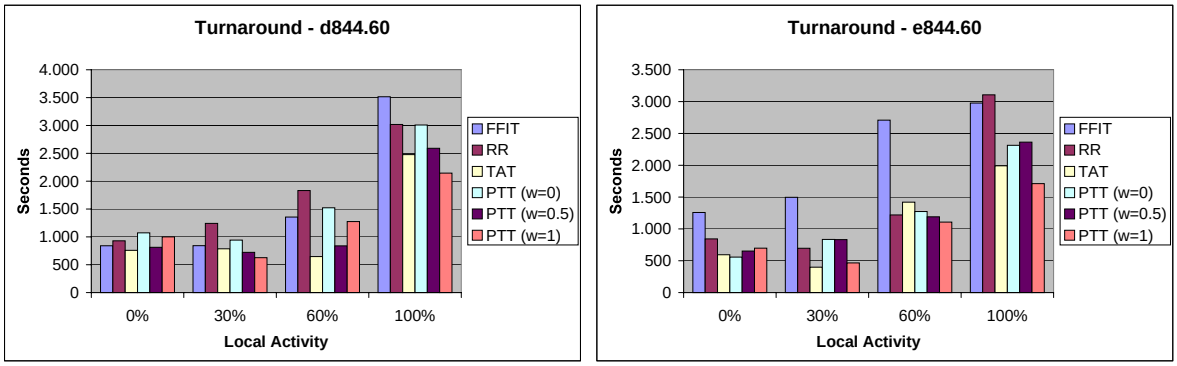


Figura 4.2: Evaluación del *Turnaround Time* (distinto número de nodos x cluster)

Esto nos indica que las políticas de predicción se comportan mejor cuando la cantidad de recursos que forman los distintos clusters está equilibrada (Figura 4.1). Cuando los recursos están repartidos de forma desigual entre los diferentes clusters (Figura 4.2), se puede dar el caso que existan aplicaciones que solo se puedan ejecutar en un determinado cluster. En estos casos el sistema impone restricciones que impiden a las políticas de predicción tomar decisiones que permitan mejorar el rendimiento.

Otro de los comportamientos que se observan mirando las figuras 4.1 y 4.2 es que en los sistemas con un conjunto de recursos similar, el comportamiento de los mecanismo basados en predicción *TAT* y *PTT* es parecido independientemente de la carga local del sistema. En cambio en sistemas donde la cantidad de recursos en cada cluster es diferente y con una distribución desequilibrada de la carga local, el mecanismo de selección *PTT* obtiene mejores resultados que el mecanismo *TAT*. Esto nos lleva a pensar que en determinado tipo de configuraciones el factor de ponderación w de la política *PTT* puede ayudar a conseguir mejores resultados.

Efecto del factor de ponderación (w) de *PTT* sobre el rendimiento

La Figura 4.3 muestra los valores medios del tiempo de *turnaround* obtenidos para cada mecanismo de selección *TAT* y *PTT*. El mecanismo *PTT* (*Ponderated Turnaround Time*) se diferencia del *TAT* (*Turnaround Time*) en que para seleccionar un cluster considera, además del tiempo estimado de *turnaround* el grado de utilización de los recursos por parte de los usuarios locales, según la ecuación 4.1.

La influencia del grado de utilización de los recursos por parte del usuario local con respecto a la estimación del tiempo de turnaround se mide mediante el parámetro

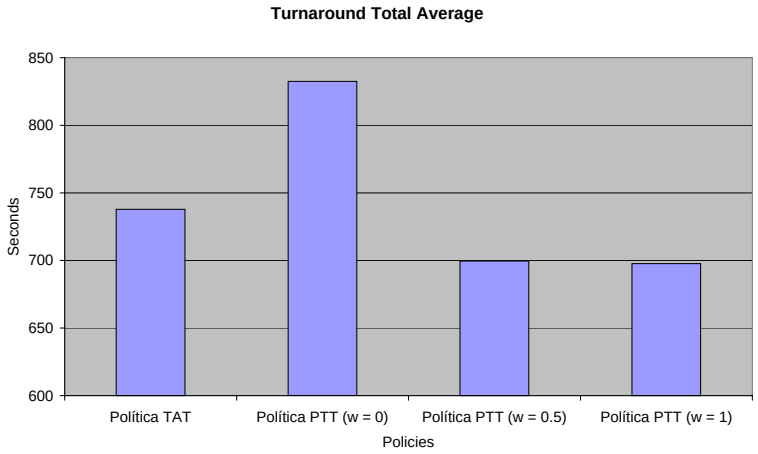


Figura 4.3: *Turnaround time* promedio según mecanismo de selección de recursos

de ponderación w. Un valor de $w = 0$ significa que la utilización de los recursos por parte del usuario local es muy importante. Un valor de $w = 1$ significa que se le da más importancia a la estimación del tiempo de *turnaround*.

El mecanismo de selección que ofrece el mejor rendimiento para las aplicaciones paralelas es la política *PTT*. Concretamente, para valores de $C = 0{,}5$ y $w = 1$, sin prácticamente ninguna diferencia entre ambas. Por otro lado, *PTT* con $w = 0$ es la que obtiene peores resultados. Este resultado es lógico si tenemos en cuenta que $PTT_{w=0}$ busca entre todas las posibles asignaciones aquella donde el número de recursos con carga local es menor, sin tener en cuenta el tiempo de retorno estimado.

Este mecanismo tiene la ventaja que las decisiones tomadas respetan al máximo la utilización de los recursos por parte del usuario local. Por otro lado, puede pasar que estas decisiones no sean las que ofrecen el mejor rendimiento para la aplicación paralela, disminuyendo así el rendimiento global ofrecido por esta política.

Un inconveniente de $PTT_{w=0}$, es que a la hora de valorar las distintas soluciones, sólo se tiene en cuenta la cantidad de nodos que tienen carga local y no el uso de recursos que esta carga realiza. Pueden existir asignaciones con menor número de nodos con carga local pero con una utilización muy superior a otras con un mayor número de nodos con carga local y con una pobre utilización de recursos.

Análisis de la intrusión al usuario local

En esta sección se analiza como influyen las aplicaciones paralelas en la interactividad del usuario local. Medir como afectan las aplicaciones paralelas a la interacción del usuario local no es una tarea fácil.

En este trabajo se ha propuesto un método para medir la intrusión, basado en calcular el promedio de los tiempos de latencia de un conjunto de llamadas al sistema durante la ejecución de las aplicaciones paralelas. Para poder establecer una medida comparativa del efecto de las aplicaciones paralelas sobre el sistema, se ponderan los tiempos de latencia obtenidos en cada experimento con los obtenidos sin la ejecución de ninguna carga paralela.

A esta métrica la hemos denominado *Slowdown*, y nos va a permitir medir las diferencias en los tiempos de latencia cuando se introducen cargas paralelas en el sistema. En la figura 4.4, se muestra el promedio del *Slowdown* obtenido por cada política en las pruebas realizadas.

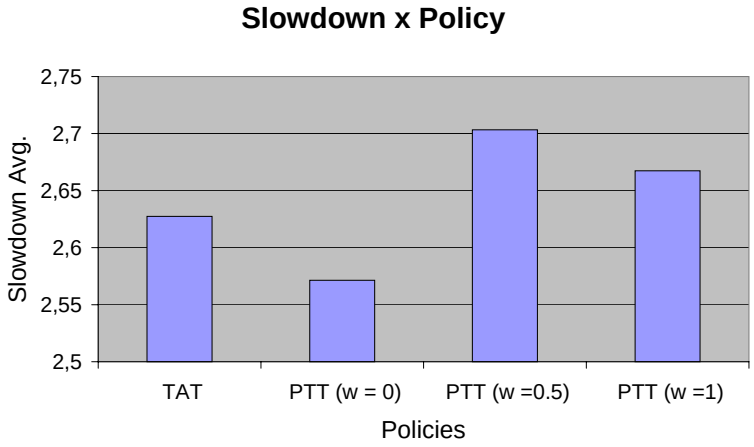


Figura 4.4: *Slowdown* promedio según mecanísmo de selección de recursos

Como se puede observar, el mecanismo de selección que proporciona el *Slowdown* más bajo es $PTT_{w=0}$. Este mecanismo obtiene un beneficio del 2,13 % respecto del uso de la política *TAT*. Esto indica que $PTT_{w=0}$ consigue los objetivos para los que se ha diseñado, que es respetar al máximo los recursos disponibles para el usuario local, aunque esto implique una disminución del rendimiento de las aplicaciones paralelas, tal como hemos podido comprobar en la figura 4.3.

Por otro lado, $PTT_{w=0,5}$ representa la otra cara de la moneda, ya que nos ofrece el *Slowdown* más alto pero también el mejor promedio en el tiempo de retorno.

Analizando detalladamente los resultados, observamos que $PTT_{w=0}$ obtiene un beneficio en el *Slowdown* de un 2,13 % mientras que pierde un 11,37 % en el promedio del tiempo de retorno respecto a la política *TAT*. El mecanismo $PTT_{w=0,5}$, nos ofrece una pérdida del 2,8 % en el *Slowdown* y un beneficio en el tiempo de retorno del 5,18 % respecto al mecanismo *TAT*.

El mecanismo $PTT_{w=1}$, aunque se comporta de forma similar a $PTT_{w=0,5}$, obtiene una pérdida en el *Slowdown* de un 1,49 %, y un un beneficio del 5.4 % en el promedio del tiempo de retorno de la aplicación paralela respecto del mecanismo TAT.

La mejor elección cuando el objetivo es preservar al máximo la interactividad del usuario local es $PTT_{w=0}$. Cuando no existen usuario locales o no tienen requisitos de interactividad importantes, la mejor elección es la utilización de $PTT_{w=1}$ que ofrece los mejores resultados en el rendimiento de las aplicaciones pero con una menor intrusión que $PTT_{w=0,5}$. La política mejor equilibrada es TAT, ya que obtiene buenos resultados de rendimiento y una menor intrusión que la política PTT con $w = [0{,}5\,,\,1]$.

La opción $PTT_{w=0,5}$, nos proporciona rendimientos de aplicación similares a los de la política $PTT_{w=1}$ pero con la intrusión ligeramente superior debido a que reparte la carga paralela dando la misma importancia a la cantidad de carga local que a la estimación del tiempo de turnaround. Este reparto de las aplicaciones paralelas puede ser útil para mejorar la utilización de los recursos a costa de introducir algo más de intrusión al usuario local.

Análisis del MakeSpan

El *Makespan* mide el tiempo que tarda el sistema en ejecutar el *workload paralelo*. Esta métrica nos proporciona información sobre el rendimiento global del sistema.

Una característica importante a tener en cuenta del *Makespan* es que los resultados están influenciados por el instante de tiempo en que entra la última aplicación al sistema. Supongamos un sistema en que queremos probar distintas técnicas de planificación. Si todas las técnicas ofrecen un buen rendimiento a las aplicaciones paralelas, o bien el instante de tiempo en que entra la última aplicación está muy alejado del resto, puede pasar que cuando llegue la última aplicación, el resto de tareas ya hayan finalizado. En este caso, el *Makespan* será el mismo para todas las técnicas y no va a permitir extraer conclusiones del rendimiento de las distintas técnicas.

En la figura 4.5 se muestran los valores del *Makespan* promedio obtenido de todas las pruebas realizadas para cada una de las políticas que se han implementado en nuestro sistema.

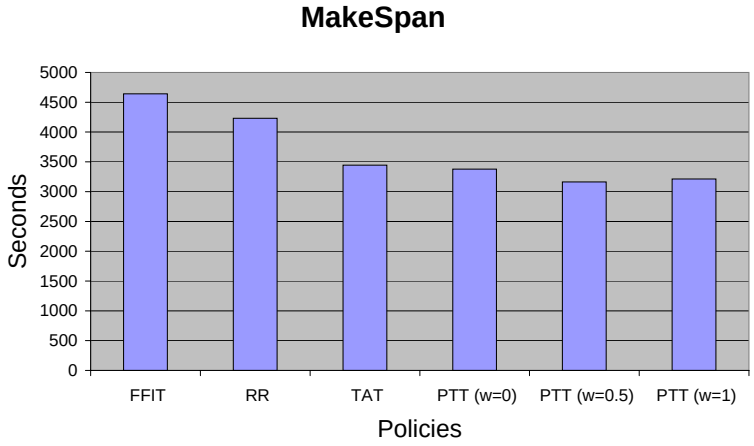


Figura 4.5: *Makespan* según mecanísmo de selección de recursos

Se puede observar como el rendimiento global del sistema mejora notablemente para los mecanismos de selección que utilizan la predicción. Entre los mecanismos de predicción, PTT ofrece resultados ligeramente mejores que TAT. Los mecanismos que obtienen un mejor rendimiento global son $PTT_{w=0,5}$ y $PTT_{w=1}$.

Esto apoya los resultados obtenidos en la sección 4.3.3, donde analizamos los tiempos de turnaround. En ellos podíamos ver que los mecanismo que proporcionaban un mejor rendimiento a las actividades paralelas son $PTT_{w=0,5}$ y $PTT_{w=1}$, y esto se ve reflejado en el rendimiento global del sistema (*Makespan*). Aunque en ambos casos, estos mecanismos proporcionan una mayor intrusión al usuario local.

4.3.4. Conclusiones

La utilización de los mecanismos basados en predicción *TAT* y *PTT* ofrecen mejores resultados de rendimiento en todos los ámbitos, rendimiento del sistema (*makespan*), rendimiento de las aplicaciones paralelas (*turnaround*) y confort del usuario local (*slowdown*), respecto a las políticas clásicas.

En el caso de $PTT_{w=0}$ se obtiene la mayor penalización del tiempo de *turnaround* promedio, consiguiendo una pérdida en el rendimiento de un 11 % con respecto al mecanismo *TAT* y de un 16 % con respecto al $PTT_{w=0,5}$. En cambio, es el mecanismo que menor intrusión produce sobre el usuario local, obteniendo un 3 % menos de intrusión que TAT y un 5 % menos que $PTT_{w=0,5}$. El mecanismo que mejores resultados obtiene en cuanto al rendimiento de las aplicaciones paralelas (*turna-*

round) es $PTT_{w=0,5}$, obteniendo una mejora del 5 % con respecto al rendimiento de las aplicaciones paralelas y un 5 % más de intrusión sobre el usuario local que con el mecanismo *TAT*. El mecanismo $PTT_{w=1}$ ofrece resultados similares al mecanismo $PTT_{w=0,5}$ en el rendimiento de las aplicaciones paralelas y un 1 % menos de intrusión sobre el usuario local.

De estos resultados se desprende que debe tenerse en cuenta la posibilidad de aplicar distintas configuraciones de la política *PTT*, para poder obtener los resultados más óptimos según el estado del sistema y los requisitos de la carga paralela. $PTT_{w=0}$ resulta muy útil cuando el interés principal es el confort del usuario local, mientras que $PTT_{w=0,5}$, es la mejor opción cuando se prima el rendimiento de las aplicaciones paralelas o cuando el tipo de carga local no tiene fuertes requisitos de interactividad. Por otro lado, *TAT* ofrece unos resultados razonables de rendimiento sin afectar la intrusión del usuario local tanto como lo hace $PTT_{w=0,5}$, siendo el mecanismo ideal cuando deseamos un equilibrio entre el efecto sobre el rendimiento de las aplicaciones paralelas y la intrusión al usuario local.

Los métodos de selección de recursos que se proponen *TAT* y *PTT* superan el rendimiento de los sistemas de selección clásicos más utilizados. Estos mecanismos además consideran tanto la carga del usuario local como el dinamismo de las aplicaciones paralelas durante su ejecución. Esto proporciona además de disponer de una estimación del tiempo de turnaround una toma de decisiones más respetuosa con el usuario local y más realista con la evolución del sistema en un futuro próximo y una reducción de los costes de redistribución de aplicaciones paralelas como sucede en [AD00] y de replicación de aplicaciones en distintos clusters como se presenta en [SKSS02].

4.4. Propuesta de mecanismo de Selección de trabajos

El mecanismo de selección de trabajos más común en la literatura en entornos Cluster [LLM88, Fei97, AKN98] y Multicluster [BE03a, BE03b, BBE03] es el *FCFS* (*First Come First Served*). El principal inconveniente de este mecanismo es que cuando un trabajo se bloquea en la cola de espera porque no existen suficientes recursos para su ejecución, el resto de trabajos con menores requisitos de recursos deben esperar en la cola aún existiendo suficientes recursos para su ejecución. Aumentando el tiempo de espera de los trabajos en la cola global disminuye el rendimiento de las aplicaciones paralelas y el rendimiento global del sistema.

En situaciones con elevada carga, cuando el mecanismo de selección de clusters ha seleccionado el cluster más adecuado para ejecutar una determinada aplicación

paralela basándose en la estimación del tiempo de *turnaround*, puede suceder que la aplicación sea rechazada por una limitación en la capacidad de la cola local del cluster.

En [SvANS00, BE01] se hace referencia a un límite máximo en las colas locales aunque no se establece ningún criterio para determinar dicho valor máximo. Por otro lado, cuanto mayor es el tamaño de las colas locales, se espera una mayor desviación en la estimación del tiempo de turnaround. En la sección 4.5 proponemos un método para determinar de forma dinámica el límite de trabajos en las colas locales.

Cuando una aplicación paralela es rechazada por el cluster donde ha sido asignada, debemos decidir si debe ser planificada en otro cluster o bien si debe permanecer en la cola de espera global y durante cuanto tiempo. Otra cuestión que se debe resolver es si durante el bloqueo de la aplicación en la cola de espera global es preferible adelantar otras aplicaciones con menores requisitos, o bien se debe esperar a que se planifique la primera de las aplicaciones.

Con el fin de poder resolver estas cuestiones en la presente sección proponemos distintos mecanismos de selección de trabajos que nos permiten definir cuando una aplicación paralela es bloqueada en la cola de espera global, cuando puede ser desbloqueada y si puede o no ser adelantada por otras aplicaciones con menores requisitos.

Para cada mecanismo de selección propuesto se evalúa el rendimiento de las aplicaciones paralelas con distintos tipos de carga local con el fin de determinar qué mecanismo se comporta mejor y bajo qué condiciones.

Por otro lado, se establecen distintos límites de ocupación para las colas locales de cada cluster y se evalúan sus efectos sobre el rendimiento de las aplicaciones paralelas y sobre la desviación en las estimaciones del tiempo de *turnaround*.

El objetivo principal es determinar si es o no útil limitar el tamaño de las colas locales, sus efectos sobre el rendimiento de las aplicaciones, la desviación sobre las estimaciones y por último determinar el mecanismo de selección que obtiene un mejor comportamiento en función del dinamismo de la carga local.

4.4.1. Selección de trabajos bloqueante

Nuestra primera propuesta, que denominaremos *bloqueante* (*B*), se basa en la política más común en este tipo de entornos la *FCFS* (*First Come First Served*). El *Algoritmo 4.4* describe los principales pasos del mecanismo de selección de trabajos.

En primer lugar, el mecanismo de selección *bloqueante* (B) selecciona para ser ejecutado el primer trabajo de la *Upper-Level Queue* (línea 2). De este modo cuando

un trabajo paralelo llega a la *Upper-Level Queue* deberá esperar la asignación de todos los trabajos previos antes de poder ser planificado.

A continuación, el meta-planificador activa el mecanismo de selección de clusters (línea 6). El cluster más adecuado para ejecutar la aplicación se selecciona en función de la estimación del tiempo de *turnaround*.

Cuando la cola del cluster escogido está llena, el trabajo paralelo se bloquea en la *Upper-Level Queue* (línea 10) y permanece en espera de que el cluster escogido libere algún trabajo paralelo de su cola de espera (líneas 1, 4, 15). Una vez se libera espacio en la cola del cluster escogido (línea 8), el trabajo paralelo es desbloqueado y asignando al cluster correspondiente (líneas 17 y 18).

A continuación, se comprueba de nuevo la *Upper-Level Queue* en busca del siguiente trabajo paralelo (línea 1).

4.4.2. Selección de trabajos no-bloqueante

El mecanismo de selección de trabajos *no-bloqueante* (*NB*) trata de reducir el tiempo de espera de los trabajos en la *Upper-Level Queue* producido por el bloqueo de la primera aplicación de la cola. La idea principal es que mientras el primero de los trabajos permanece bloqueado, otros trabajos pueden ser asignados al resto de clusters que no han alcanzado el tamaño máximo de su cola de espera. Reduciendo el tiempo de espera se aumenta el rendimiento de las aplicaciones paralelas y el global del sistema.

El *Algoritmo 4.4* muestra los pasos del mecanismo de selección *no-bloqueante*. La diferencia con el mecanismo *bloqueante* (*B*) está en las lineas 12 y 13. Cuando la cola de espera del cluster seleccionado para la ejecución del trabajo paralelo está llena (línea 8), en lugar de bloquear el trabajo paralelo se elimina de la *Upper-Level Queue* (línea 12) eliminando también el cluster bloqueado de la cola de clusters disponibles (*MCluster*) (línea 13).

A continuación, se recorre la *Upper-Level Queue* en busca de otros trabajos que puedan ser asignados a los clusters disponibles hasta que finalmente no quede ningún trabajo en la cola global o ningún cluster disponible (línea 1). Cuando la cola de espera de alguno de los clusters bloqueados libera un trabajo paralelo, la lista de cluster disponibles (*MCluster*) se actualiza automáticamente.

Este mecanismo es muy similar a la técnica *FPFS* presentada en [SME04, BE03a, BE03b], aunque en estos trabajos no se utiliza un sistema jerárquico de colas y los trabajos se bloquean en la cola de espera por falta de recursos y no por limitaciones

Algorithm 4.4 Job Selection algorithm

Require:

***MCluster*:** List of clusters with their *Low-level queue* not full. $MCluster = \{C_i, i = 1..n\}$, where n is the number of Clusters making up the Multicluster.

UpperLQueue: List of the waiting jobs in the *Upper-level Queue*. $UpperLQueue = \{J_i, i = 1..k\}$, where k is the number of jobs in the *Upper-level Q*ueue.

1: **While** ($UpperLQueue \neq \{\phi\}$ **and** $MCluster \neq \{\phi\}$) **do**
2: Select the first job from $UpperLQueue$.
3: **If** (J has a previous estimation of the turnaround time) **then**
4: $C =$ Cluster selected in the previous estimation of the job J.
5: **else**
6: Select Cluster $C = C_i \in MCluster$, with minimum estimated turnaround time for job J
7: **end if**
8: **If** (number of jobs in *Low-level Queue* of Cluster C is equal to its *Low-level Queue* limit) **then**
9: **if (Blocking)**
10: Block J in the *Upper-level queue* until the unblocking of Low-level queue of C
11: **else /* non-blocking */**
12: $UpperLQueue = UpperLQueue - J$. Delete the job of the Available jobs list.
13: $MCluster = MCluster - C$. Delete the Cluster of the Available Clusters list.
14: **end if**
15: **Continue** /* Go to the begin of the while loop*/
16: **end if**
17: Dispatch the job J into Cluster C
18: $UpperLQueue = UpperLQueue - J$
19: **end while**

en el número de trabajos de las colas locales. En estos casos las condiciones de espera y desbloqueo de un trabajo paralelo en la cola del Multicluster son ligeramente distintas.

4.4.3. Selección de trabajos con re-planificación

El inconveniente principal de los mecanismos presentados anteriormente es que no se adaptan a los cambios de estado de los recursos. La selección del cluster más adecuado para la ejecución de un trabajo paralelo se realiza una sola vez cuando el trabajo llega a la cabeza de la *Upper-Level Queue*. Si la cola de espera del cluster seleccionado está al límite de su ocupación, el trabajo se bloquea hasta que el cluster libera algún trabajo paralelo de su cola de espera. Durante todo ese tiempo el estado del Multicluster puede variar provocando que la estimación del tiempo de *turnaround* y la asignación previa sean erróneas.

El mecanismo de selección de trabajos con *re-planificación* (*RS*) trata de resolver este inconveniente según sea el método de bloqueo que adoptemos *bloqueante* (*B*) o *no-bloqueante* (*NB*). La idea es que cuando un trabajo paralelo es desbloqueado de la *Upper-Level Queue* se active de nuevo el mecanismo de selección de clusters, cuando se detecta que el estado del Multicluster es distinto con respecto a la asignación anterior. Esto puede causar una nueva asignación (*re-planificación*) del trabajo paralelo que se adapte mejor a las nuevas circunstancias del estado del sistema.

La *re-planificación* puede ser aplicada a ambos mecanismos de selección de trabajos explicados previamente, el bloqueante (*B*) y el no-bloqueante (*NB*) sustituyendo en el algoritmo 4.4 la línea 3 por la línea siguiente:

3: **If** (*J* has a previous estimation of the turnaround time **and** the Multicluster state has not changed with regard to the previous estimation) **then**

La principal aportación de este mecanismo es que nos permite reducir los efectos negativos producidos por el dinamismo de la carga local y aumentar el rendimiento con respecto a los mecanismos estáticos presentados anteriormente sin introducir un gran coste en el algoritmo de meta-planificación.

4.4.4. Experimentación

En la presente sección se analiza el comportamiento de los mecanismos de selección de trabajos propuestos, bajo distintas condiciones de carga y distintos límites en los tamaños de las colas locales. El objetivo principal es demostrar la utilidad de la

limitación de las colas de los clusters y qué mecanismo de selección de trabajos obtiene los mejores resultados de rendimiento.

Los mecanismos de selección de trabajos utilizados han sido el *bloqueante* (***B***), el *no-bloqueante* ***NB*** y el bloqueante y no-bloqueante con *re-planificación,* ***B.RS*** y ***NB.RS*** respectivamente. Los límites establecidos en el tamaño de las colas son de $1,2,3,4$ y UL (sin límites), donde todos los trabajos son asignados al cluster seleccionado sea cual sea el tamaño de la cola sin producirse espera ninguna en la *Upper-level Queue*.

El entorno en que se ha llevado a cabo el presente análisis es un Multicluster formado por 3 clusters idénticos y no dedicados. Cada cluster está formado por 8 nodos uniprocesador PIV a 3GHz con 1GB de memoria principal y una memoria cache de 2048KB unidos por una red de comunicación Giga-Ethernet.

El perfil de usuario local escogido es el *XWindows+Internet* modelado según los parámetros de uso siguientes, un 15 % de uso de CPU, 35 % de Memoria y un 0,5MB/sec. En la presente experimentación se han definido dos tipos de carga local distintos, la carga local estática que introduce un volumen de carga local (0 %, 30 %, 60 %, 100 %) distribuido de forma aleatoria entre los nodos del Multicluster y que permanece estático durante la ejecución de todas las aplicaciones paralelas, y la carga local dinámica donde los nodos que reciben el volumen de carga local varían durante la ejecución de las aplicaciones paralelas.

El *workload paralelo* está formado por un total de 60 aplicaciones escogidas de forma aleatoria entre el conjunto de benchmarks de la NAS (IS y MG) con distintos requerimientos en el número de tareas (2, 4 y 8) y en el consumo de recursos (clases A, B y AB). Estas aplicaciones llegan al sistema bajo una distribución de Poison con un promedio de llegada de $\mu = 6 sec$. El nivel de paralelismo máximo esta limitado a un máximo de 4 aplicaciones paralelas ($MPL \leq 4$) con el fin de preservar la utilización de los recursos por parte del usuario local.

En los experimentos realizados en esta sección se ha evaluado el *rendimiento* de los trabajos paralelos y la *desviación* en las predicciones. El rendimiento de las aplicaciones paralelas es evaluado mediante el promedio de los tiempos de *turnaround* de los trabajos ejecutados. La desviación en la predicción se calcula mediante la ecuación 4.4.

$$\frac{avg\,|estimated\,time - real\,time|}{avg\,(real\,time)} \tag{4.4}$$

donde *estimated time* es el tiempo de *turnaround* estimado y *real time* es el tiempo de *turnaround* que se ha obtenido en la ejecución real.

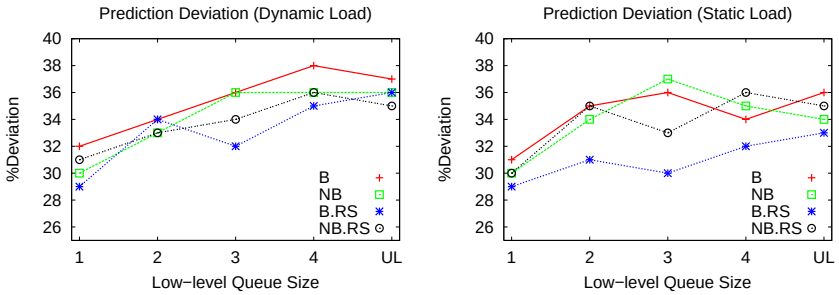


Figura 4.6: Desviación en la predicción según mecanísmo de selección de trabajos

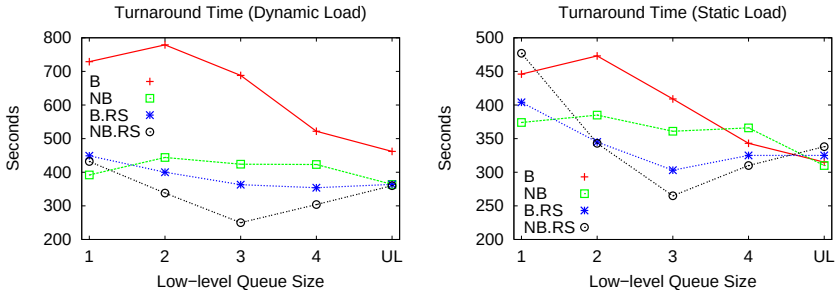


Figura 4.7: *Turnaround time* promedio según mecanísmo de selección de trabajos

Análisis de la Desviación en la Predicción

En la Figura 4.6 se muestra la desviación producida en la estimación de los tiempos de *turnaround* para los mecanismos de selección de trabajos propuestos con distintos tamaños de cola. Como se puede observar, la desviación en la predicción crece con el aumento del tamaño de las colas y se hace más pronunciada cuando la carga local es dinámica.

Esto nos indica que la desviación en la predicción es sensible a la variabilidad en la carga local y depende de la longitud de las colas. El error acumulado durante el proceso de predicción de un determinado trabajo aumenta con el número de trabajos en la cola de espera local.

Por otro lado, se observa que los mecanismos con re-planificación (*B.RS* y *NB.RS*) ofrecen mejor precisión en las estimaciones obteniendo una ganancia del 4 % con respecto a los mecanismos sin re-planificación (*B y* NB*).*

Análisis del Rendimiento

En la Figura 4.7 se observa el rendimiento obtenido por los distintos mecanismos con distintos límites en el tamaño de la cola local. En general, el incremento en

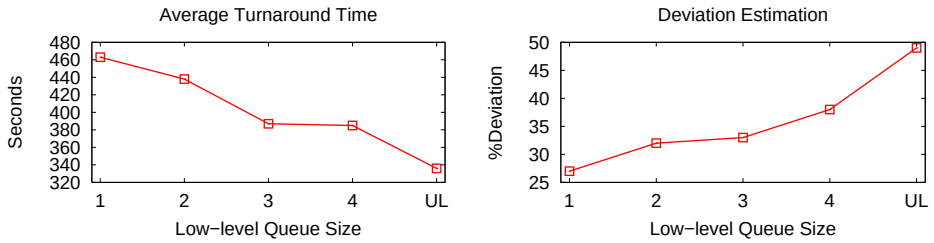


Figura 4.8: Efecto del tamaño de cola sobre el rendimiento y la desviación

el límite del tamaño de las colas proporciona mejores resultados en el rendimiento dado que se reduce los tiempos de espera de los trabajos en la *Upper-level queue*.

En cuanto a los mecanismos de selección de trabajos, los mecanismos con *re-planificación* proporcionan mejores resultados debido a que son capaces de capturar las variaciones en el estado de los recursos, especialmente en las situaciones con carga dinámica. En este caso el mejor rendimiento se obtiene con el mecanismo *no-bloqueante con re-planificación* (*NB.RS*) con una ganancia promedio de un 13 % con respecto al *bloqueante con re-planificación* (*B.RS*).

En general, las políticas bloqueantes obtienen peores resultados que las no-bloqueantes debido al incremento en el tiempo de espera de las aplicaciones paralelas producido por el bloqueo de las aplicaciones en la *Upper-level queue*.

Un comportamiento interesante que se observa en la Figura 4.8 es que con un valor para el límite del tamaño de las colas de 3 trabajos se obtiene un aumento significativo del rendimiento de las aplicaciones paralelas con una desviación razonable en la precisión de las estimaciones. A partir de este valor la desviación en la predicción aumenta de forma considerable comparado con la ganancia obtenida en el rendimiento de las aplicaciones paralelas.

Esto nos hace pensar que existe un valor óptimo para el tamaño máximo de la cola que depende directamente de la velocidad de ejecución de cada cluster y de la velocidad de llegada de los trabajos paralelos. En la sección 4.5 se propone un método que permite determinar para cada cluster en particular el tamaño máximo de las colas en función de la capacidad de producción y la tasa de llegada de las aplicaciones paralelas al cluster.

Para valores por encima del óptimo los resultados de rendimiento y desviación tienden a igualarse independientemente del tamaño máximo de la cola. Esto es así, porque cuando el tamaño máximo de las colas es grande o inexistente el tiempo de espera de los trabajos paralelos en la *Upper-level queue* es nulo provocando que los mecanismos bloqueantes y de re-planificación no puedan intervenir y no produzcan

ningún efecto sobre el rendimiento o la desviación, obteniendo resultados similares para todos los mecanismos.

4.4.5. Conclusiones

El tamaño máximo de longitud de las colas locales tiene un efecto directo sobre la precisión en las estimaciones del tiempo de turnaround y sobre el rendimiento de las aplicaciones paralelas. Se tiende a pensar que la mejor elección son aquellos mecanismos que proporcionan un mejor rendimiento de las aplicaciones sin considerar la desviación en la predicción. Sin embargo, la precisión en la predicción permite proporcionar a los usuarios del sistema información fiable sobre el comportamiento de las aplicaciones e incluso implementar mecanismos de QoS que en ocasiones son más importantes incluso que el rendimiento de las aplicaciones paralelas.

Otra observación importante es que existe un valor óptimo para el tamaño máximo de las colas donde la desviación en la predicción es razonable y se obtiene un aumento en el rendimiento de las aplicaciones paralelas. Este valor depende directamente de la velocidad de ejecución de cada cluster y de la velocidad de llegada de los trabajos paralelos. En la sección 4.5 se propone un método que permite determinar de forma dinámica en función de las condiciones particulares de cada cluster el valor óptimo para el tamaño máximo de cola local.

En general, los mecanismos con re-planificación obtienen los mejores resultados de precisión en la predicción y rendimiento de las aplicaciones paralelas, obteniendo una ganancia del 4 % en la precisión de las estimaciones y del 20 % en el rendimiento de las aplicaciones paralelas con respecto a los mecanismos sin re-planificación. Estos mecanismos son capaces de capturar los cambios producidos en la carga local permitiendo tomar mejores decisiones de meta-planificación, siendo especialmente útiles en sistemas no dedicados.

4.5. Propuesta al límite de tamaño de las colas locales

En un entorno de planificación con un sistema de colas de espera jerárquico como el que se propone en capítulo 2, con una cola global (*Upper-Level Queue*) en el meta-planificador (*MetaLoRaS*) y una cola local en cada cluster (*LoRaS*), una de las cuestiones que se plantean es si las colas locales de cada cluster deben tener una capacidad máxima y en caso afirmativo cual es el tamaño adecuado.

Santoso [SvANS00] y Bucur [BE01] hacen referencia a un tamaño máximo de las colas locales pero no se establece ningún criterio para determinar el mismo. Abawajy propone en [AD03] un sistema donde cada cluster solicita al planificador de

nivel superior trabajos paralelos cuando aumenta su capacidad de procesamiento no acumulando trabajos en las colas de espera.

En la experimentación realizada en la sección 4.4 se pone de manifiesto la necesidad de establecer un límite a la capacidad de las colas locales. Limitando las colas locales favorecemos que las aplicaciones paralelas que no puedan ser ejecutadas en un cluster local por la falta de recursos libres, se mantengan en espera en la cola global del meta-planificador proporcionando varias ventajas.

En primer lugar, aumenta la precisión en la estimación del tiempo de *turnaround* permitiendo realizar mejores asignaciones. Para estimar el tiempo de turnaround de un aplicación se necesita estimar el tiempo de ejecución de las aplicaciones anteriores en la cola de espera del cluster donde la aplicación ha sido asignada. Tal como se demuestra en la sección 4.4 el error acumulado durante el proceso de predicción de un determinado trabajo aumenta con el número de trabajos en la cola de espera local. En segundo lugar, la espera de las aplicaciones en la cola global permite tomar mejores decisiones de meta-planificación adaptándose a los cambios de estado en los recursos del multicluster y evitando los inconvenientes de una asignación estática.

Sin un límite en la capacidad de las colas locales, en períodos con una elevada tasa de llegada de las aplicaciones paralelas, se produce la acumulación de aplicaciones en las colas disminuyendo la precisión en las estimaciones. La disminución en la precisión de las estimaciones produce asignaciones erróneas disminuyendo el rendimiento de las aplicaciones paralelas y del sistema en general. En los casos en que se produzca un aumento drástico de la utilización de los recursos por parte del usuario local, la asignación estática de las aplicaciones a un determinado cluster produce un efecto dramático sobre el rendimiento del sistema que solo puede verse reducido pagando un elevado coste de re-asignación de aplicaciones a otros clusters.

En la presente sección proponemos un método de auto-ajuste del tamaño máximo de las colas locales que permite a cada cluster determinar de forma dinámica el tamaño adecuado de la cola de espera en función de su capacidad de procesamiento, con el fin de evitar los inconvenientes descritos anteriormente.

4.5.1. Método de reajuste dinámico del tamaño máximo de cola local

En la sección 4.4 se demuestra que existe un valor adecuado para el tamaño máximo de las colas con el que se obtienen mejores rendimientos de las aplicaciones paralelas y menor desviación en las estimaciones. Lo que sucede en realidad, es que al establecer un límite en el crecimiento de la cola se evita que en situaciones con

una elevada carga o bien con una llegada masiva de aplicaciones paralelas, la cola crezca indefinidamente provocando los efectos negativos que esto conlleva, perdida de precisión en las estimaciones y la re-asignación de aplicaciones.

El tamaño de la cola puede oscilar durante la ejecución de un workload paralelo, aumentando cuando los recursos están muy ocupados y disminuyendo cuando los recursos están libres y las aplicaciones se consumen con mayor velocidad. El número medio de trabajos en la cola según [Kle75], lo podemos expresar mediante la ecuación 4.5, donde ρ representa el grado de ocupación de la cola.

$$\bar{N} = \frac{\rho}{1-\rho} \tag{4.5}$$

Lo que hay que intentar es que en situaciones críticas, cuando los nodos consumen aplicaciones muy despacio debido a una elevada carga y/o se produce un aumento en la tasa de llegada de las aplicaciones paralelas, la cola de espera de los cluster crezca rápidamente produciendo los efectos negativos mencionados.

Por este motivo, necesitamos un sistema que permita limitar el tamaño máximo de trabajos en las colas locales en función del estado de los recursos, detectando la posible saturación de los recursos y limitando el acceso de más trabajos cuando se producen estas situaciones. Este mecanismo puede ser aún más interesante si esta detección se hace con tiempo suficiente para que el cluster no acepte trabajos que luego deban permanecer largos periodos de tiempo en la cola esperando a que desaparezca la saturación de los recursos.

La idea consiste en no permitir la entrada de más aplicaciones paralelas en aquellos clusters que se encuentran en estado de saturación. Un cluster está saturado cuando recibe más aplicaciones de las que es capaz de consumir aumentando rápidamente el número de aplicaciones en la cola.

La Figura 4.9 representa el modelo jerárquico de colas con el que estamos trabajando. El grado de ocupación de la cola de espera del cluster *i* se puede definir mediante la ecuación 4.6.

$$\rho_i = \frac{\lambda_i}{\mu_i} \; \forall i \in 1..\alpha \tag{4.6}$$

donde λ_i es la tasa media de llegada de aplicaciones a la cola de espera del cluster y μ_i es la tasa media de servicio del cluster. Siendo $1/\mu_i$ el tiempo medio de ejecución de las aplicaciones en el cluster.

La condición de estabilidad de la cola de espera de cualquier cluster *i* viene dada por la ecuación 4.7.

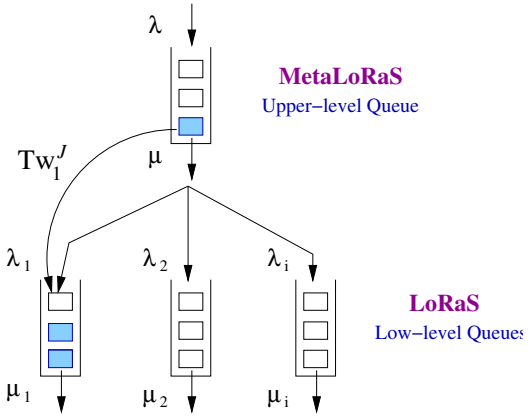


Figura 4.9: Modelo jerárquico de colas de espera en el Multicluster

$$\rho_i = \frac{\lambda_i}{\mu_i} < 1 \tag{4.7}$$

Cuando la ocupación del cluster $\rho_i \geq 1$ es mayor o igual que 1, significa que el cluster recibe aplicaciones más rápido de lo que es capaz de consumir encontrándose en estado de saturación.

Una vez el meta-planificador selecciona el cluster C_i más adecuado para ejecutar una determinada aplicación paralela según el mecanismos de selección de clusters, el meta-planificador asigna la aplicación al cluster correspondiente y el Cluster debe decidir si acepta o no la aplicación. En caso de no saturación del cluster $\rho_i < 1$, la aplicación paralela se introduce en la cola de espera y se procesa localmente. En caso de saturación, $\rho_i \geq 1$, el cluster rechaza la aplicación y el meta-planificador se encarga de tomar una decisión en función del mecanismo de selección de trabajos descrito en la sección 4.4.

Con este método permitimos que cada cluster autoajuste el tamaño máximo de la cola de espera en función de su capacidad de producción. La capacidad de producción viene determinada por las características de los recursos del cluster y por su ocupación que puede variar de forma dinámica en función de la carga local. De este modo cada cluster puede admitir un mayor número de aplicaciones paralelas a medida que su capacidad de procesamiento aumenta y disminuir su ratio de admisiones en caso de saturación.

4.5.2. Método de prevención de la saturación

Aunque el método de reajuste propuesto permite adaptarse de forma dinámica a la situación de ocupación de los recursos hay que tener en cuenta determinadas consideraciones a la hora de calcular la ocupación de las colas.

Los parámetros para determinar la ocupación de las colas, según la ecuación 4.6, son la tasa de llegada de las aplicaciones (λ) y la tasa de servició (μ). Un modo de mantener actualizados estos parámetros es recalcularlos a cada nuevo evento de llegada o ejecución de una aplicación paralela. Estos parámetros aunque reflejan el estado actual del sistema no consideran la naturaleza de las aplicaciones existentes en la cola y se pueden producir ciertos efectos negativos.

Supongamos que en un momento determinado se liberan gran cantidad de recursos en un cluster cargado, esto provoca la planificación de nuevas aplicaciones de la cola de espera aumentando la tasa de servicio (μ) y por tanto la admisión de nuevas aplicaciones a la cola de espera. Si se admiten nuevas aplicaciones sin considerar la naturaleza de las aplicaciones actuales en la cola de espera, puede suceder que alguna de las aplicaciones que estaba en espera sea una gran consumidora de recursos saturando nuevamente el cluster. En este caso, el cluster acepta nuevas aplicaciones en la cola de espera sin ser consciente que en breve el cluster estará de nuevo saturado.

Un modo de prevenir esta situación es considerar la naturaleza de las aplicaciones en la cola de espera antes de admitir nuevas aplicaciones paralelas. De este modo, cuando una nueva aplicación J solicita ser asignada a un Cluster i, el gestor de colas debería predecir la tasa de servicio (μ_i) considerando los trabajos actualmente en la cola de espera. Esto permitiría detectar la saturación antes de asignar nuevos trabajos y prevenir la asignación de aplicaciones ofreciendo al meta-planificador la posibilidad de tomar mejores decisiones que permitan aumentar el rendimiento del sistema.

La tasa de servicio de cualquier cluster μ_i puede estimarse, mediante la ecuación 4.8, a partir del tiempo de espera estimado de la aplicación paralela entrante J en el cluster i, que denotaremos Tw_i^J.

$$\mu_i = \frac{n_i}{Tw_i^J} \tag{4.8}$$

siendo n_i el número de procesos en la cola. Esto significa, que con una estimación del tiempo de espera Tw_i^J de la aplicación paralela J, podemos estimar la tasa de servicio μ_i producida por los trabajos en la cola de espera y prevenir posibles saturaciones.

Utilizando la estimación del tiempo de espera proporcionada por el motor de predicción del cluster y expresando la condición de estabilidad mediante la ecuación 4.9, obtenemos una estimación de la saturación del cluster *i* considerando las aplicaciones que se encuentran en su cola de espera.

$$\rho_i = \frac{\lambda_i \cdot Tw_i}{n_i} < 1 \tag{4.9}$$

Esta información proporciona información útil al meta-planificador para tomar mejores decisiones de planificación, evitando la asignación de aplicaciones a clusters que se prevé que en breve estén saturados.

4.6. Propuesta a la Co-asignación

La compartición de recursos entre distintos clusters durante la ejecución de una aplicación paralela se conoce con el nombre de ***co-asignación***. Existen un gran número de estudios [SCJG00, BE02, EHS+02, BE00, BE01, JLPS05] que demuestran que compartiendo recursos entre distintos *clusters* se puede aumentar la utilización de los recursos del Multicluster y mejorar el rendimiento de las aplicaciones paralelas y del sistema en general.

Sin embargo, los canales de comunicación entre clusters pueden saturarse muy rápidamente formando un cuello de botella que puede afectar tanto a las aplicaciones paralelas que comparten estos canales como a las aplicaciones locales con necesidades de comunicación hacia el exterior de los clusters (internet por ejemplo), perjudicando así su rendimiento y el global de todo el sistema.

Para obtener un buen rendimiento utilizando la *co-asignación* se deben considerar tanto la heterogeneidad y la capacidad de los recursos de cómputo como las características y el estado de los canales de comunicación entre clusters [EHS+02, BE01, JLPS05]. Sin embargo, la mayoría de estudios previos [BAG00, Wol03, HHL+06] no prestan atención a los recursos de comunicación.

En la sección 3.3.1 del capítulo 3, propusimos un modelo que considera las características y la ocupación de los recursos de cómputo y de los canales de comunicación internos a un cluster. En la presente sección proponemos una extensión de dicho modelo que considera además las características y la ocupación de los canales de comunicación entre clusters.

Una vez disponemos de un modelo que nos permite valorar el tiempo de ejecución de una aplicación paralela en función de las características y la ocupación de los recursos, necesitamos un mecanismo que nos permita obtener la mejor asignación de

la aplicación paralela teniendo en cuenta los requisitos de la misma. Para la búsqueda de la mejor asignación proponemos modelar el problema de la asignación como un problema de *programación entera binaria* (*Mixed Integer Programming*) que mediante una búsqueda exhaustiva del espacio de soluciones nos permite obtener una aproximación de la mejor asignación posible y evitar la saturación de los canales de comunicación. Dado que los valores utilizados en el modelado del estado de los recursos y de la caracterización de las aplicaciones son aproximados mediante estimaciones o funciones estadísticas, no podemos probar que las soluciones obtenidas sean soluciones óptimas, y entonces hablaremos de métodos de aproximación a las soluciones óptimas.

Algunos trabajos previos [BBC+04, NLYW05] han utilizado también técnicas de programación entera para resolver problemas de planificación. Sin embargo, Naik en [NLYW05] busca una solución óptima para el rendimiento del sistema asumiendo que el *workload paralelo* es conocido con anterioridad. Este procedimiento que resulta útil en sistemas *Batch* no puede ser aplicado en sistemas on-line donde no se conocen las cargas que pueden venir a continuación. Banino por otro lado presenta en [BBC+04] una solución a la planificación temporal de trabajos con dependencias de datos difícil de llevar a la práctica.

La búsqueda de la mejor asignación es un problema NP-Completo para el cual existen distintos mecanismos meta-heurísticos que permiten realizar búsquedas inteligentes por el espacio de soluciones obteniendo en un tiempo razonable una aproximación a la solución óptima. Algunos de los meta-heurísticos más utilizados [DA06] son los conocidos como *Genetic Algorithm* (*GA*), *Simulated Annealing* (*SA*) y *Tabu Search* (*TS*). Otros estudios proponen la utilización de algoritmos meta-heurísticos para la búsqueda de la soluciones óptimas considerando la optimización de múltiples criterios [KNP01, MWW04, KNOW08]. Existe la posibilidad de utilizar algoritmos de búsqueda que permiten obtener soluciones próximas al óptimo en un tiempo razonable considerando tanto las características de cómputo como de comunicación a partir del modelo propuesto en el presente trabajo.

En la presente sección se describe el modelo de *programación entera binaria* (*Mixed Integer Programming*) propuesto y se analiza su comportamiento bajo distintas configuraciones, composición del entorno multicluster, estado de los recursos y requisitos de las aplicaciones. El objetivo principal es determinar los parámetros críticos del modelo y estudiar su escalabilidad. Además, se proponen alternativas al modelo que pueden ser útiles para proporcionar soluciones válidas en situaciones con elevada carga o bien con objetivos de rendimiento distintos como la utilización de los recursos, etc.

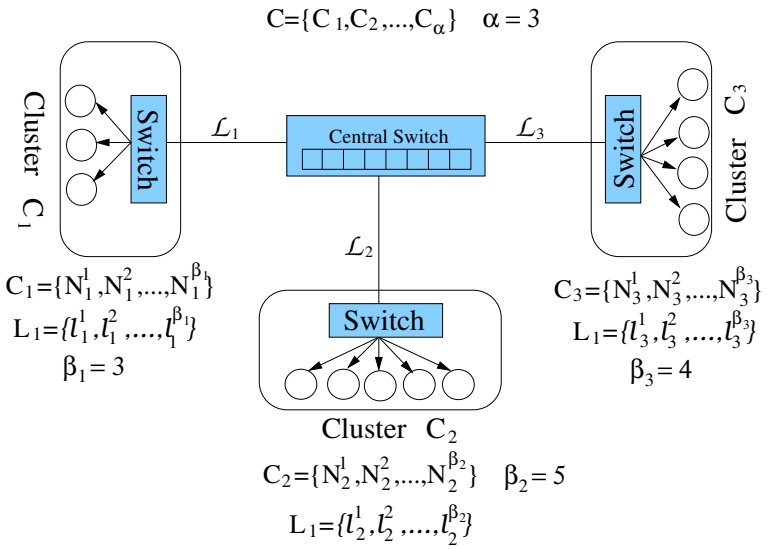


Figura 4.10: Topología Multicluster

4.6.1. Modelado del tiempo de ejecución

En la sección 3.3.1 se presenta un modelo analítico que determina el tiempo de ejecución de una aplicación paralela en función de las características y el estado de los recursos de cómputo y comunicación del cluster donde se ejecuta. En la presente sección extendemos el modelo a un entorno multicluster, considerando además el efecto que tienen las características y la ocupación de los canales de comunicación entre clusters sobre el tiempo de ejecución de las aplicaciones paralelas que comparten recursos entre clusters.

4.6.1.1. Topología del entorno Multicluster

La figura 4.10 representa un Multicluster formado por un conjunto C de clusters $C = \{C_1...C_\alpha\}$ y un conjunto $\mathscr{L}$ de canales de comunicación entre clusters $\mathscr{L} = \{\mathscr{L}_1...\mathscr{L}_\alpha\}$ donde cada $\mathscr{L}_k$ representa el canal de comunicación entre el cluster C_k y el switch central. Cada cluster C_k, siendo $k = 1..\alpha$, está compuesto por un conjunto arbitrario de nodos $C_k = \{N_k^1...N_k^{\beta_k}\}$ y un conjunto de canales de comunicación $L_k = \{l_k^1...l_k^{\beta_k}\}$ internos al cluster, donde l_k^i representa el canal de comunicación entre el nodo i del cluster k, N_k^i, y el switch del cluster k.

El tiempo de ejecución de una aplicación paralela se define del mismo modo que para un único cluster mediante la ecuación 4.10.

$$T^e = T^p \cdot SP + T^c \cdot SC \tag{4.10}$$

donde T^p y T^c representan el tiempo de cómputo y comunicación obtenidos en un entorno dedicado y *SP* y *SC* el retardo sobre el tiempo de cómputo y comunicación producido por la compartición de los recursos con otros usuarios y su heterogeneidad.

La diferencia está en que ahora para el cálculo de *SP* y *SC* se debe considerar los nodos de distintos clusters. Para el cálculo de *SC*, además, se debe considerar la ocupación y las características de los canales de comunicación entre clusters cuando una aplicación comparte recursos entre diferentes clusters.

4.6.1.2. Cálculo del Slowdown de procesamiento (SP)

El Slowdown de Procesamiento (*SP*) que inflinge cada nodo sobre la aplicación paralela j, se calcula del mismo modo que en un único cluster pero considerando ahora todos los nodos del multicluster, según la ecuación 4.11.

$$SP(j) = \begin{cases} 0 & when\ N_k^i \notin T_k \\ (\Gamma_k^i)^{-1} & otherwise \end{cases} \tag{4.11}$$

donde T_k representa el conjunto de nodos del cluster k con tareas de la aplicación paralela asignadas, y Γ_k^i la *Potencia efectiva* del nodo i del cluster k, $\forall i \in 1..\beta_k$.

Siendo el ***Slowdown de procesamiento de la aplicación paralela*** $SP(j)$ el máximo *slowdown de procesamiento* de todos los nodos del Multicluster, según la ecuación 4.12.

$$SP(j) = max\left\{SP(j)_k^i,\ 1 \leq i \leq \beta_k\ \forall k \in 1..\alpha\right\} \tag{4.12}$$

donde β_k es el número de nodos del cluster k.

4.6.1.3. Cálculo del Slowdown de Comunicación (SC)

El cálculo del Slowdown de comunicación (*SC*) es ligeramente distinto al utilizado en un solo cluster. En primer lugar debemos considerar las características y el estado de los canales de comunicación entre clusters con el fin de detectar y evitar los cuellos de botella. En segundo lugar, necesitamos simplificar el cálculo de la saturación para poder expresar el problema con un modelo de *programación entera binaria* (***M**ixed **I**nteger **P**rogramming)*. Para ello, buscamos una expresión para la

saturación sin necesidad de calcular el reajuste del ancho de banda de cada aplicación como hacíamos en el modelo utilizado por el motor de predicción (sección 3.3.1.2).

El retraso en las comunicaciones de una aplicación paralela viene dado por el canal mas lento (saturado) de todos los canales de comunicación entre clusters que la aplicación paralela utiliza. La idea es calcular el *Slowdown de comunicación* (*SC*) de los canales entre clusters y obtener el máximo de todos ellos, ecuaciones 4.15 y 4.16.

El *Slowdown de comunicación* de cada canal viene dado por la saturación del canal tras la asignación de la aplicación paralela j. El grado de saturación de un canal k se define como la relación entre el ancho de banda máximo y el total requerido, entendiendo por total requerido el ancho de banda actualmente en uso más el requerido por la aplicación j, tal como se define en la ecuación 4.13.

$$BW_k^{sat} = \frac{BW_k^{max}}{BW_k^{consumed} + BW_k^j} \quad \forall k \in 1..\alpha \tag{4.13}$$

donde BW_k^{max} representa el ancho de banda máximo del canal de comunicación entre el cluster k y el switch central del MultiCluster. $BW_k^{consumed}$ representa el ancho de banda actualmente en uso y BW_k^j representa el ancho de banda requerido por la aplicación paralela j. Cuando $BW_k^{sat} \geq 1$ significa que el canal k no está saturado, mientras que $BW_k^{sat} < 1$ significa que el canal k está saturado, retrasando las comunicaciones de todas la aplicaciones que comparten el canal.

En esta ecuación, la capacidad máxima del canal BW_k^{max} y el ancho de banda actualmente en uso $BW_k^{consumed}$ son valores proporcionados mediante la monitorización de los recursos, mientras que el ancho de banda requerido es una característica de la aplicación paralela j.

Con el fin de modelar el ancho de banda requerido BW_k^j por la aplicación paralela j en cada canal k, asumimos que las aplicaciones paralelas siguen un patrón de comunicaciones *all-to-all*, donde todas las tareas tienen el mismo requisito de ancho de banda sobre el canal de comunicaciones del nodo que ocupan, caracterizado por $PNBW^j$.

En los casos en que la aplicación paralela j comparta recursos entre distintos clusters, el ancho de banda utilizado en los canales de comunicación entre clusters depende de la cantidad de tareas asignadas a cada cluster y se calcula mediante la ecuación 4.14.

$$BW_k^j = \left(t_k^j \cdot PNBW^j\right) \cdot \left(\frac{n_T^j - t_k^j}{n_T^j - 1}\right), \quad \forall k \in 1..\alpha \tag{4.14}$$

donde n_T^j es la cantidad de tareas de la aplicación paralela j y t_k^j es la cantidad de tareas asignadas al cluster C_k. El primer factor entre paréntesis representa el volumen de comunicación requerido por las tareas asignadas al cluster C_k mientras que el segundo factor representa el porcentaje del volumen de comunicación que atravesará el canal entre clusters k.

El *Slowdown de comunicación* de cualquier canal del multicluster viene dado por la inversa del grado de saturación, tal como indica la ecuación 4.15.

$$SC_k = \left(BW_k^{sat}\right)^{-1} \quad \forall k \in 1..\alpha \tag{4.15}$$

Siendo el ***Slowdown de comunicación de la aplicación paralela*** $SC(j)$ el máximo *slowdown de comunicación* de todos los canales del Multicluster, según la ecuación 4.16.

$$SC(j) = max_k\{SC_k, \quad \forall k \in 1..\alpha\} \tag{4.16}$$

4.6.2. Propuesta de un Modelo MIP para la Selección de recursos

La *programación entera* (Integer Programming) es un procedimiento matemático mediante el cual podemos resolver problemas en los que se desea maximizar o minimizar el valor de una determinada función objetivo sujeta a ciertas restricciones. En este tipo de problemas la función objetivo y sus restricciones son expresiones lineales.

En nuestro caso particular queremos resolver un problema de asignación de tareas a recursos. Este tipo de problemas se resuelven mediante modelos de *programación entera binaria* (**M**ixed-**I**nteger **P**rogramming). La *programación entera binaria* (*MIP*) es un caso particular de la programación entera donde la solución óptima se compone de valores binarios indicándonos qué nodos debemos utilizar (1) y cuales no (0).

El objetivo principal de nuestro meta-planificador es la selección de recursos maximizando el rendimiento de las aplicaciones paralelas evitando los efectos negativos del proceso de comunicación entre clusters. Para cumplir con este objetivo proponemos un modelo **MIP** donde la función objetivo es minimizar el tiempo de ejecución de la aplicación paralela expresado mediante el modelo presentado en la sección 4.6.1.

Para poder encontrar la asignación óptima mediante el modelo **MIP**, se requiere cierta información sobre la aplicación y sobre el estado del Multicluster. La *información sobre la aplicación* contiene los requisitos de la aplicación paralela, la cantidad de tareas, la cantidad de memoria, el ancho de banda requerido por cada tarea ($PNBW^j$) y finalmente, la ratio del tiempo de cómputo y el tiempo total de ejecución. Los nodos sin suficiente memoria se descartan. El *estado del Multicluster* se compone de la disponibilidad de memoria y de CPU, la máxima capacidad de los canales entre clusters y su disponibilidad.

La mejor asignación es aquella que devuelve el menor tiempo de ejecución de la aplicación paralela. La aplicación puede ser o no co-asignada entre distintos clusters. El software utilizado para resolver el modelo **MIP** es *LINGO* [wLI] que tiene la posibilidad de utilizar distintos solvers en función de los requerimientos del problema.

4.6.2.1. Definición del modelo

Un modelo de programación entera se compone de un conjunto de parámetros de entrada, variables de decisión, un conjunto de restricciones que debe cumplir la solución propuesta y una función objetivo. El objetivo del modelo es encontrar aquellos valores de las variables de decisión que cumplen con las restricciones y que minimizan o maximizan el valor de la función objetivo.

En la Figura 4.11 se describe nuestra propuesta de modelo **MIP**. El modelo proporciona la mejor asignación posible de tareas de la aplicación paralela a nodos teniendo en cuenta los requerimientos de la aplicación paralela y las características de heterogeneidad y disponibilidad de los recursos del Multicluster.

El conjunto de parámetros de entrada se divide en dos grupos, los requerimientos de la aplicación paralela y el estado del Multicluster. Los parámetros sobre los requerimientos de la aplicación paralela son el número de tareas (τ^j) y el ancho de banda requerido por cada tarea ($PNBW^j$). Los parámetros que caracterizan el estado del Multicluster son la potencia efectiva de cada nodo (Γ_k^i), el ancho de banda disponible BW_k^{av} y el máximo ancho de banda BW_k^{max} de los canales entre clusters.

El conjunto de variables de salida está formado por un vector de variables binarias X_k^i, que indica para cada nodo del Multicluster si tiene asignada (1) o no (0) una tarea de la aplicación paralela j, y las variables *SP* y *SC* que indican el Slowdown de procesamiento y comunicación que toma la aplicación paralela en la solución proporcionada por el modelo.

Input arguments:

1. j: job to be matched.
2. τ^j: number of tasks making up job j.
3. $PNBW^j$: per-node bandwidth requirement for the job j.
4. $M = C..C_\alpha$: Multicluster composition.
5. $\mathscr{L} = \{\mathscr{L}_1....\mathscr{L}_\alpha\}$ and L=$\{L_k\}$=$\{L_k^i,\ 1 \leq i \leq \beta_k$ and $1 \leq k \leq \alpha\}$: set of inter- and intra- cluster links.
6. Γ_k^i: Effective Power weight for the cluster i node k ($1 \leq i \leq \beta_k$ and $1 \leq k \leq \alpha$).
7. BW_k^{av}: available bandwidth for each inter-cluster link $\mathscr{L}_k$, $1 \leq k \leq \alpha$.
8. BW_k^{max}: maximum bandwidth for each inter-cluster link $\mathscr{L}_k$, $1 \leq k \leq \alpha$.

Output parameters:

9. X_k^i, $1 \leq i \leq \beta_k$ and $1 \leq k \leq \alpha$: boolean variable associated to cluster k node i. X_k^i=1 if job j is matched to cluster k node i, and 0 otherwise.
10. SP: processing slowdown. $SP = max\{SP_k^i, 1 \leq i \leq \beta_k\, and\, 1 \leq k \leq \alpha\}$.
11. SC: inter-cluster link communication slowdown. $SC = max_k\{SC_k,\ 1 \leq k \leq \alpha\}$.

Objective Function:

12. $min\{T^e = T^P \cdot SP + T^C \cdot SC\}$

Constraints:

13. Gang matching.
14. Non inter-cluster link saturation.

Figura 4.11: Definición del modelo MIP

4.6.2.2. Función objetivo

Cuando existen varias soluciones posibles al problema de la asignación la función objetivo define la calidad de la solución. El solver utiliza la función objetivo para dirigir la búsqueda hacia la mejor solución posible. En nuestro caso, el objetivo es obtener el mejor rendimiento de las aplicaciones paralelas y esto lo podemos expresar minimizando el tiempo de ejecución.

Utilizando el modelo del tiempo de ejecución de una aplicación paralela definido en la sección 4.6.1 donde se considera la heterogeneidad y la disponibilidad de los recursos de cómputo y comunicación de todo el entorno Multicluster, definimos la función objetivo según la ecuación 4.17.

$$min\{T^e = T^P \cdot SP + T^C \cdot SC\} \tag{4.17}$$

4.6.2.3. Restricciones

El conjunto de restricciones ayuda a acotar el conjunto de soluciones que puede tomar la función objetivo y que pueden no ser soluciones válidas para el problema que se está planteando. La solución proporcionada por el modelo deberá minimizar la función objetivo y cumplir con las restricciones impuestas.

En el presente modelo definimos dos restricciones, la *asignación grupal* (Gang Matching) y la *no saturación* (non-saturation) de los canales entre clusters. La restricción de *asignación grupal* es de obligado cumplimiento mientras que la no saturación de los canales entre clusters la podemos o no introducir en el modelo aportando un mayor refinamiento de las soluciones o una relajación de las restricciones. Más adelante se analiza el efecto que tiene la utilización o no de esta restricción.

Restricción de asignación grupal (Gang matching)

Esta restricción es de obligado cumplimiento y permite asegurar que todas las tareas que componen la aplicación paralela han sido asignadas de modo que a toda tarea le corresponde un nodo. Esta restricción se formaliza mediante la ecuación 4.18.

$$\sum_{1 \le i \le \beta_i, 1 \le k \le \alpha} X_k^i = \tau^j \tag{4.18}$$

donde τ^j es el número de tareas que forman la aplicación paralela j y X_k^i es la variable de decisión que vale 1 cuando el nodo N_k^i ha sido asignado a una tarea y 0

cuando no lo ha sido. De este modo se garantiza que la suma de nodos asignados es igual a la cantidad de tareas que componen la aplicación paralela j.

Restricción de no saturación (non-saturation)

Esta restricción permite asegurar que el ancho de banda consumido de los canales entre clusters, una vez asignada la aplicación paralela j, no supera la capacidad total del canal produciendo la saturación del canal y retrasando las aplicaciones paralelas que lo comparten. Esta restricción se formaliza mediante la ecuación 4.19.

$$SC(j) \leq 1 \tag{4.19}$$

donde $SC(j) = max_k\{SC_k, 1 \leq k \leq \alpha\}$ es el slowdown de comunicación de los canales entre clusters utilizados por la aplicación paralela j, calculado mediante la ecuación 4.16.

4.6.3. Experimentación

El objetivo principal de esta experimentación es analizar la escalabilidad del modelo **MIP** propuesto. Para ello analizamos el comportamiento de nuestro modelo ante variaciones en los requerimientos de las aplicaciones paralelas, en la composición del Multicluster y en el estado de los recursos. De este modo podemos observar qué parámetros tienen un mayor efecto sobre el tiempo de respuesta del modelo y el grado de sensibilidad. Además, se ha evaluado mediante la ejecución en un entorno real la capacidad del modelo de obtener una estimación del tiempo de ejecución de las aplicaciones paralelas.

Para poder llevar a cabo esta experimentación necesitamos definir la configuración de partida del Multicluster, los requisitos de las aplicaciones paralelas y los requisitos de la actividad local introducida en el sistema.

El entorno Multicluster utilizado en la presente experimentación se compone de 2 clusters no dedicados. El cluster C_1 es un cluster homogéneo formado por 10 nodos uniprocesador PIV a 3GHz con 1GB de memoria principal unidos por una Giga-Ethernet. El cluster C_2 es un cluster heterogéneo formado por 10 nodos, 5 nodos uniprocesador a 3GHz con un 1GB de RAM con una interfaz de red Giga-Ethernet, y 5 nodos multiprocesador a 3GHz con 512 MB de memoria principal con una interfaz de red Fast-Ethernet.

El perfil de usuario local definido ha sido modelado con los parámetros de uso siguientes, un 15 % de uso de CPU, 35 % de memoria y una comunicación de

0,5MB/sec. El número de estaciones con actividad local varia del 0 % al 75 % de los nodos del multicluster con una distribución aleatoria.

Las aplicaciones paralelas utilizadas para llevar a cabo la siguiente experimentación son dos benchmarks de la NAS[BDBS92] bien conocidos: MG (Multigrid) y IS (Integer Sort). Cada benchmark tienen requisitos de procesamiento/comunicación distintos y se define para cada uno de ellos el número de tareas, el tiempo de ejecución en un entorno dedicado y el promedio del volumen de comunicación por cada tarea ($PNBW^j$). Para analizar el efecto de los requisitos de comunicación sobre el modelo, se varia el volumen de comunicación de cada tarea ($PNBW^j$) entre el 25 % y el 75 % de la capacidad máxima del canal.

Con el fin de analizar el efecto de las restricciones sobre el modelo MIP proponemos tres versiones distintas:

Optimal Este modelo busca la asignación óptima teniendo en cuenta los requisitos de la aplicación paralela, las características de los recursos y su disponibilidad. La asignación óptima corresponde con aquella solución que minimiza el Slowdown de procesamiento y comunicación obteniendo así el mínimo tiempo de ejecución de la aplicación paralela. Este modelo no evita la saturación de los canales entre clusters.

Non-saturated En este modelo aplicamos la restricción de no saturación. El solver busca el mínimo tiempo de ejecución sin saturación en los canales entre clusters.

Non-saturated with Non-Optimal Este modelo no busca la solución óptima. El solver devuelve la primera solución que evita la saturación de los canales entre clusters. Esta versión está ideada para ser utilizada en las situaciones con gran número de recursos, donde la obtención de una solución óptima puede resultar demasiado costosa.

Análisis del rendimiento

En primer lugar, comparamos el efecto del aumento de los requerimientos de comunicación y la cantidad de carga local para las versiones del modelo ***Optimal*** y ***Non-saturated***.

La Figura 4.12 muestra el efecto del aumento de los requerimientos de comunicación y carga local sobre el slowdown de comunicación obtenido en las soluciones. El modelo ***Non-saturated*** (Figura 4.12 (derecha)) asegura la obtención de soluciones sin saturación en los canales entre clusters y muestra un ligero crecimiento del

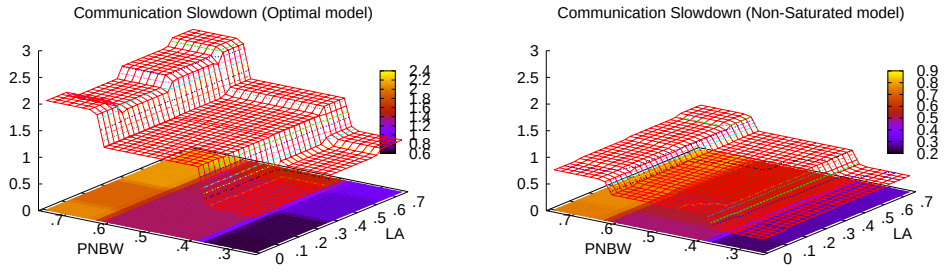


Figura 4.12: Slowdown de comunicación variando $PNBW^j$ y la actividad local

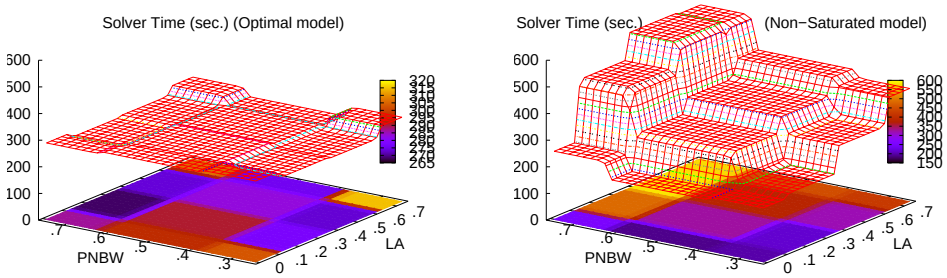


Figura 4.13: Tiempo de respuesta variando $PNBW^j$ y la actividad local (LA)

grado de ocupación de los canales entre clusters a medida que aumentan los requerimientos de comunicación de la aplicación paralela $(PNBW^j)$. El modelo ***Optimal*** (Figura 4.12 (izquierda)) no asegura la no saturación de los canales de comunicación y se observa como el slowdown de comunicación crece rápidamente con los requerimientos de comunicación de la aplicación paralela $(PNBW^j)$. No sucede lo mismo con la carga local que tiene un efecto poco significativo comparado con los requerimientos de comunicación.

En la Figura 4.13 se observa el efecto de los requerimientos de comunicación de la aplicación paralela $(PNBW^j)$ y la cantidad de carga local (LA) sobre el tiempo de respuesta del solver. El comportamiento de los modelos ***Optimal*** y ***Non-saturated*** es totalmente opuesto. En el modelo ***Optimal*** (Figura 4.13 (izquierda)) el tiempo de respuesta del solver permanece invariable con el aumento de los requisitos, mostrando pequeñas variaciones únicamente en los extremos del rango de variación de los requisitos de comunicación y carga local.

En el modelo ***Non-saturated*** (Figura 4.13 (derecha)), el tiempo de respuesta del solver crece rápidamente con el aumento de los requisitos, especialmente los requerimientos de comunicación de la aplicación paralela. Esto es así porque en el

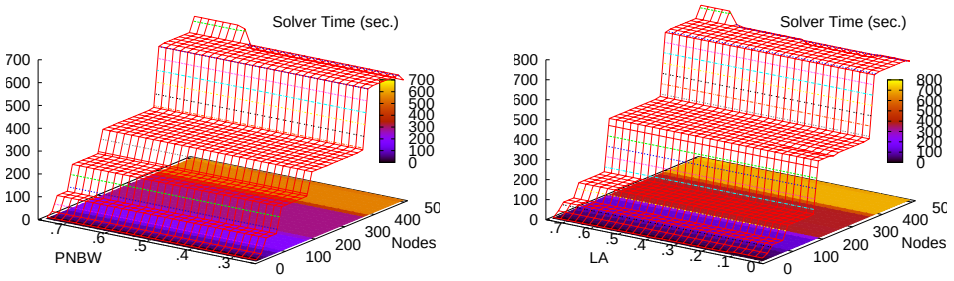


Figura 4.14: Tiempo de respuesta variando la cantidad de nodos x cluster.

modelo ***Non-saturated***, la restricción de no saturación reduce el espacio de posibles soluciones haciendo más costosa la búsqueda de una solución óptima.

La Figura 4.14 muestra el efecto que tiene sobre el tiempo de respuesta del solver el aumento de la cantidad de nodos en cada cluster con el modelo ***Optimal***. Como se puede observar, la cantidad de nodos en el cluster es el parámetro que tiene un mayor efecto sobre el tiempo de respuesta del solver. Estos resultados corroboran los obtenidos en la Figura 4.13(izquierda) donde el tiempo permanece invariable con el aumento de los requerimientos de comunicación de la aplicación paralela $(PNBW^j)$ y la cantidad de carga local (LA).

Por último, analizamos el impacto que tiene sobre el tiempo de respuesta del solver la cantidad de clusters (canales entre clusters) que componen el Multicluster. Para ello fijamos el número de nodos por cluster (8 nodos) y variamos entre 2 y 64 el número de clusters (canales entre clusters). En la Figura 4.15 se observa como las restricciones establecidas en los distintos modelos tienen un impacto directo sobre el tiempo de respuesta del solver.

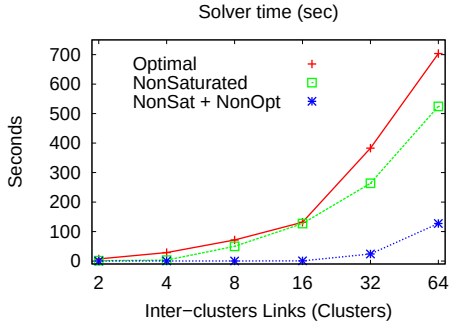


Figura 4.15: Tiempos de respuesta del modelo MIP

Como se puede observar, en todos los modelos, el tiempo de respuesta tiende a crecer exponencialmente. Este resultado era de esperar puesto que la búsqueda de la asignación óptima es un problema NP-completo. Sin embargo, para un Multicluster formado por 16 clusters (canales entre clusters) con 8 nodos x cluster (128 nodos) obtenemos un tiempo de respuesta cercano a los 2min., tiempo razonable para aplicaciones con tiempos de ejecución de varias decenas de minutos y/o superiores.

El modelo ***Non-Optimal*** es sin duda el modelo que proporciona mejores tiempos de respuesta debido a que no realiza la búsqueda de una solución óptima, asegurando únicamente la no saturación de los canales entre cluster. Este modelo puede ser de gran utilidad para la asignación de aplicaciones paralelas con un tiempo de ejecución corto en un entorno multicluster compuesto por un gran número de recursos (clusters y/o nodos).

Análisis de la precisión en la estimación

El objetivo principal del modelo **MIP** es obtener la mejor asignación de la aplicación paralela a los recursos del multicluster evitando la saturación de los canales entre clusters. El modelo del tiempo de ejecución presentado en la sección 4.6.1 además de ser utilizado como función objetivo del modelo MIP, permite obtener una estimación del tiempo de ejecución de la aplicación paralela en función de la asignación.

Con el fin de evaluar la utilidad del modelo como un estimador del tiempo de ejecución una vez obtenida la solución a la asignación, se compara el tiempo de ejecución estimado por el modelo **MIP** con ejecuciones reales de aplicaciones paralelas con distintos requerimientos, IS y MG. Ambos benchmarks han sido ejecutados varias veces con distinta cantidad de tareas y carga local.

La Figura 4.16 muestra los resultados obtenidos para distintos valores de carga local. Aunque existen diferencias entre los tiempos reales y estimados la tendencia de ambas lineas es similar. Esto demuestra la capacidad del modelo de aproximar el comportamiento de las aplicaciones paralelas en función del estado del sistema, siendo útil tanto para tomar decisiones de asignación como para estimar el tiempo de ejecución de la aplicación paralela en función de los recursos asignados.

Otro resultado importante es que el modelo tiende a subestimar el tiempo de ejecución corroborando los resultados obtenidos en el capítulo 3, para los métodos analíticos, sobre la precisión en la predicción del tiempo de ejecución en entornos cluster no dedicados.

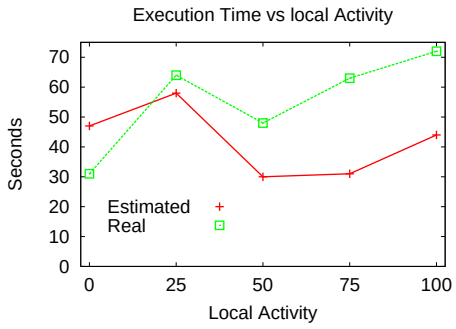


Figura 4.16: Valores estimados contra Reales

4.6.4. Conclusiones

El modelo **MIP** propuesto es capaz de capturar el comportamiento de una aplicación paralela en un entorno Multicluster no dedicado y obtener eficientemente asignaciones que permiten maximizar el rendimiento de las aplicaciones paralelas evitando la saturación de los canales entre clusters.

El tiempo de respuesta del modelo **MIP** está directamente relacionado con la cantidad de recursos del Multicluster, especialmente con el número de clusters (canales entre clusters) que aumenta enormemente el espacio de posibles soluciones.

Para entornos Multicluster como los encontrados en varios centros de investigación, formados por unos pocos clusters con varios centenares de nodos por cluster, el modelo **MIP** permite obtener soluciones en tiempos inferiores a los 10min. Estos tiempos de respuesta se pueden considerar razonables dado que los ficheros de traza de algunos centros de investigación [Gib97, SW02, LGTW04, TEF07] muestran promedios de ejecución del orden de centenares de minutos.

Mediante el manejo de las restricciones en el modelo **MIP**, es posible realizar la asignación de aplicaciones paralelas con tiempos de ejecución cortos en entornos Multicluster con un gran número de recursos (clusters y/o nodos) sin saturación en los canales entre clusters.

4.7. Refinamiento de la meta-planificación con co-asignación

En el presente trabajo estamos interesados en aprovechar las múltiples ventajas que nos ofrece la co-asignación, el aumento de la utilización de los recursos del multicluster y del rendimiento de las aplicaciones paralelas, la ejecución de trabajos

con mayores requisitos, etc, evitando los efectos negativos que su aplicación puede producir tanto en el sistema como en las propias aplicaciones paralelas.

En [BE02] se afirma que considerar los recursos de un Multicluster como un único super-cluster donde cualquier trabajo puede ser co-asignado, perjudica enormemente el rendimiento y propone limitar la co-asignación a trabajos con requisitos de comunicación reducidos. Por otro lado, Jones [JLPS05] propone limitar la co-asignación a aquellos trabajos donde el 85 % de las tareas pueda ser asignado a un solo cluster.

Sin embargo, aunque el uso de estas *heurísticas* permite obtener buenos resultados en algunos casos, estas técnicas no consideran las características particulares de las aplicaciones paralelas, la heterogeneidad de los recursos ni su estado, y por tanto no garantizan que los canales de comunicación estén libres de saturación.

Consideramos que la mejor solución es aplicar la co-asignación haciendo uso de un mecanismo de selección de recursos inteligente como el presentado en la sección 4.6, pero no hay una solución definitiva a los criterios qué se utilizan para determinar que aplicaciones deben o no ser co-asignadas.

En la tarea de meta-planificación propuesta en la sección 4.2, el uso de la co-asignación se reduce a los trabajos paralelos con requerimientos que no pueden ser satisfechos por ningún cluster en particular.

En la presente sección, proponemos un refinamiento a la tarea de meta-planificación con el fin de sacar mayor provecho de las ventajas de la co-asignación. El *Algoritmo 4.5* describe los pasos de la nueva tarea de meta-planificación obteniendo un mayor provecho de la técnica de *co-asignación*.

La idea principal es dividir la tarea de meta-planificación en dos fases. En la primera fase, los trabajos que llegan al *Meta-Planificador* se asignan al cluster más adecuado para su ejecución, según la estimación del tiempo de *turnaround* (paso 7). En el caso de trabajos con grandes requerimientos que no pueden ser satisfechos por ningún cluster en particular, se utiliza el mecanismo de co-asignación en busca de los recursos disponibles en los distintos clusters que mejor satisfacen las necesidades del trabajo paralelo (paso 6).

En la segunda fase de la tarea de meta-planificación, cuando todos los trabajos de la *Upper-level Queue* han sido asignados o están a la espera de que algún cluster se libere de carga, la tarea de meta-planificación recorre la cola de trabajos ya asignados a algún cluster en espera de ser ejecutados por falta de recursos, que denominaremos *Waiting-Lowlevel Queue*.

En esta cola el meta-planificador almacena de forma ordenada los trabajos asignados a los distintos clusters (paso 8). Cuando un trabajo paralelo en la cola de espera

Algorithm 4.5 Meta-Scheduling algorithm with co-allocation

Require:

***Upper-level Q*ueue:** List of waiting jobs in the *Upper-level Queue*.
***Waiting-Lowlevel Q*ueue:** List of jobs waiting for resources in the *Low-level Queue* of any cluster.

1: If the *Upper-level Queue* is empty **Go to** Step 10
2: Choose the next job (*J*) from the *Upper-level Queue.*
3: Obtain a list of Resources that satisfies the job *J* software requirements.
4: If the list is empty, reject the job *J* and **Go to** Step 1
5: Obtain a list of Clusters that have enough resources for execute the job *J*.
6: If the list is empty, **co-allocation** and **Go to** Step 8
7: Obtain the Cluster *C* to map job *J* according to the **Prediction Engine**.
8: Dispatch Job *J* and add to the *Waiting-Lowlevel Queue*.
9: **Go to** Step 1
10: Choose the next job (*J*) from the *Waiting-Lowlevel Queue.*
11: Apply **co-allocation** and Dispatch Job *J*.
12: When *J* is not the last job from the *Waiting-Lowlevel Queue* **Go to** Step 10
13: **Go to** Step 1

de un cluster es ejecutado, se envía un evento al meta-planificador y se elimina de la *Waiting-Lowlevel Queue.*

Para cada trabajo de la *Waiting-Lowlevel Queue* se aplica el mecanismo de *co-asignación*, presentado en la sección 4.6, en búsqueda de recursos de otros clusters donde poder ejecutar el trabajo sin producir saturación en los canales de comunicación (paso 11).

Cuando el mecanismo de co-asignación devuelve una solución, el meta-planificador asigna el trabajo a los recursos correspondientes y se sigue recorriendo la *Waiting-Lowlevel Queue* en búsqueda de otros trabajos a la espera de recursos libres (paso 12).

Tal como se describe en la sección 4.2, el meta-planificador realiza la asignación de trabajos a los recursos compartidos de distintos clusters mediante la virtualización de un cluster formado por los recursos libres del resto de clusters. Este cluster virtual es el encargado de ejecutar las aplicaciones paralelas co-asignadas en recursos de distintos clusters y mantener actualizado el estado de los recursos disponibles.

La implementación de esta nueva propuesta en el entorno M-CISNE, el análisis de rendimiento para distintas condiciones de carga y políticas de planificación locales, es una línea abierta de trabajo futuro en la que hemos estamos trabajando y esperamos poder realizar aportaciones muy pronto.

Capítulo 5

Conclusiones y Líneas abiertas

A principios de los años 90 muchas de las características y capacidades proporcionadas únicamente por sofisticados y costosos supercomputadores se lograron superar mediante el uso adecuado de conjuntos de ordenadores de sobremesa agrupados en sistemas denominados Cluster.

En la actualidad, el continuo aumento de las capacidades de cómputo y la disminución del coste de adquisición de los ordenadores de sobremesa ha provocado un uso extendido de los clusters en centros de investigación, instituciones, organizaciones, etc. No obstante, el crecimiento constante de los requerimientos de las aplicaciones científicas hacen necesaria la búsqueda de sistemas aún más potentes y/o con mayor número de recursos.

En la última década, la posibilidad de unir los recursos (clusters) de una misma organización para obtener mayor capacidad de cómputo ha despertado un gran interés. Esto ha propiciado la aparición de nuevos sistemas Multicluster que incorporan sofisticadas técnicas de planificación de trabajos y gestión de recursos orientadas a aprovechar al máximo los recursos de cómputo distribuidos en varios clusters.

Aunque el bajo coste de los recursos de cómputo facilita el crecimiento del número de clusters y recursos por cluster, este crecimiento tiene sus limites. En primer lugar debemos considerar los problemas de espacio que no siempre son fáciles de conseguir y en segundo lugar el elevado coste del consumo energético producido por las unidades de cómputo y por los sistema de refrigeración que se necesitan.

Bajo este panorama, un modo de proporcionar mayor número de recursos sin aumentar los costes es incorporar al Multicluster recursos de cómputo ociosos de los usuarios de una organización para la ejecución de las aplicaciones paralelas, como por ejemplo los ordenadores de laboratorio de una universidad.

La tarea de meta-planificación en entorno Multicluster es un tarea compleja y una línea de trabajo en pleno auge en la actualidad. En primer lugar, la cantidad de recursos que se deben gestionar es muy grande y los recursos pueden ser heterogéneos. En segundo lugar, deseamos considerar la planificación en un sistema no dedicado donde los recursos se comparten con otros usuarios, aumentando la dificultad de la meta-planificación.

La tarea de meta-planificación la podemos dividir en dos tareas fundamentales, la *selección de aplicaciones* y la *selección de recursos*. Las decisiones que se toman en cada una de estas tareas define el propósito del sistema Multicluster. Nuestro objetivo en el presente trabajo es conseguir el máximo rendimiento de las aplicaciones paralelas en un entorno Multicluster heterogéneo y no dedicado.

Como seleccionar el cluster más adecuado para la ejecución de una aplicación paralela, si debe existir un límite máximo de aplicaciones en las colas de espera de cada cluster, en que orden seleccionar las aplicaciones paralelas de la cola de espera del meta-planificador, como y cuando seleccionar recursos de distintos clusters (co-asignación) para la ejecución de las aplicaciones paralelas, son algunas de las cuestiones que debemos resolver para definir una tarea de meta-planificación que cumpla con los objetivos marcados.

En el presente trabajo se han propuesto soluciones a las distintas cuestiones planteadas con el fin de diseñar un sistema meta-planificador que permita minimizar el tiempo de ejecución de las aplicaciones paralelas sin perjudicar la interactividad de los usuario locales.

A continuación se resumen las puntos tratados en el presente trabajo añadiendo las aportaciones científicas a que han dado lugar.

- Se ha propuesto una arquitectura jerárquica de planificación para el sistema M-CISNE. Se ha analizado el rendimiento bajo distintas condiciones de carga y configuración del Multicluster y se compara con una arquitectura con una única cola global. Los resultados obtenidos demuestran que la jerarquía de colas permite tomar decisiones de meta-planificación más inteligentes y aumentar el rendimiento de las aplicaciones paralelas con respecto a un sistema totalmente centralizado.

  En el contexto de la nueva arquitectura, se han propuestos distintas técnicas de *selección de clusters, TAT y PTT. TAT* permite seleccionar el *cluster* más adecuado para ejecutar una aplicación paralela basándose en la estimación del tiempo de *turnaround*. *PTT* selecciona el *cluster* más adecuado considerando tanto la estimación del tiempo de *turnaround* como la ocupación de los recursos por el usuario local. Se realiza una análisis del comportamiento de estas técnicas bajo distintas configuraciones de carga y configuración del

Multicluster. Además se compara nuestra propuesta con otras técnicas de la literatura.

La principal conclusión es que las técnicas propuestas permiten obtener mejores resultados de rendimiento que las políticas clásicas y reducen la intrusión al usuario local. La técnica *PTT,* además permite al administrador del sistema primar el rendimiento de las aplicaciones paralelas y/o la no perturbación al usuario local según las necesidades de cada cluster.

Las aportaciones realizadas en este apartado aparecen publicadas en:

JL. Lérida, F. Solsona, F. Giné, M. Hanzich, P. Hernández, E. Luque
MetaLoRaS: A Predictable MetaScheduler for Non-dedicated Multiclusters.
Parallel and Distributed Processing and Applications (ISPA-06). LNCS 4330, p.630-641

M. Hanzich, J. Ll. Lérida, M. Torchinsky, F. Giné, P. Hernández, E. Luque
Using On-the-Fly Simulation for Estimating the Turnaround Time on Non-dedicated Clusters.
Euro-Par 2006 Parallel Processing. LNCS 4128, pp. 177-187.

Torchinsky, M., Hanzich, M., Giné, F., Solsona, F., Lérida, J. Ll., Hernández, P., Luque, E.
CISNE-P: A Portable and Integral Approach for Scheduling Parallel Jobs on Non-Dedicated.
Journal of Computer Science & Technology. Vol. 7- No. 1. March 2007

F. Solsona, J. García, M. Hanzich, P. Hernández, J. Ll. Lérida, E. Luque
Job Scheduling considering Best-Effort and Soft Real-Time Applications on Non-dedicated Clusters.
CACIC'07.

- El objetivo de las políticas de *selección de clusters* propuestas, es realizar asignaciones que permitan obtener el máximo rendimiento de las aplicaciones paralelas sin perjudicar al usuario local. Estas políticas se basan en la estimación del tiempo de *turnaround* en cada uno de los cluster. El rendimiento de estas políticas radica en la obtención de estimaciones lo más precisas posibles. La heterogeneidad y la naturaleza no dedicada de los recursos hacen aún más difícil la tarea de predicción.
  Con el objetivo de obtener estimaciones lo más precisas posible, se han propuesto distintas técnicas de predicción, *IBL* y *SPDN*. *IBL* se basa en un modelo estadístico basado en el aprendizaje de casos. Esta técnica se basa en el

análisis de ejecuciones pasadas y considera cierta información sobre el estado del sistema a la hora de buscar similitudes en la historia pasada. *SDPN* es un modelo analítico del tiempo de ejecución de una aplicación paralela que considera la heterogeneidad y la ocupación de los recursos tanto de cómputo como de comunicación. Este modelo analítico, es utilizado dentro de un motor de simulación que permite simular la ejecución de las aplicaciones que se encuentran en el sistema, proporcionando tanto el tiempo de espera como el de ejecución.
Se ha analizado el comportamiento de ambas técnicas de estimación bajo distintas condiciones de carga y se han comparado con las técnicas más habituales de la literatura.
Las aportaciones realizadas en este apartado aparecen publicadas en:

JL. Lérida, F. Solsona, F. Giné, JR. García, M. Hanzich, P. Hernández
Enhancing Prediction on Non-dedicated Clusters.
Euro-Par 2008 Parallel Processing. LNCS 5168, p.233-242.

M. Hanzich, P. Hernández, E. Luque, F. Giné, F. Solsona, J. Ll. Lérida
Using Simulation, Historical and Hybrid Estimation Systems for Enhacing Job Scheduling on NOWs
IEEE International Conference on Cluster Computing (Cluster 2006). p.1-12.

A raíz de los resultados obtenidos, en el presente trabajo, se ha propuesto un sistema de predicción que permite el uso combinado de las técnicas que obtienen los mejores resultados en la desviación de las estimaciones para el tiempo de espera en las colas y ejecución de las aplicaciones. Obteniendo así mejores resultados en la precisión de las estimaciones del tiempo de *turnaround*.

- Una vez desarrollada la arquitectura de planificación nos centramos en el estudio de como se debía gestionar la jerarquía del sistema de colas para llevar a cabo una tarea de meta-planificación que nos permita obtener el máximo rendimiento de las aplicaciones paralelas.

  En primer lugar, se ha realizado un estudio para determinar si debía existir o no un límite máximo de trabajos en las colas locales y cual era su efecto sobre el rendimiento de las aplicaciones paralelas y las desviaciones en las estimaciones llevadas a cabo por el motor de predicción de cada cluster.

  La principal conclusión es que existe un valor para el tamaño de las colas de cada cluster que ofrece mejores resultados en el rendimiento de las aplicaciones paralelas y desviaciones del tiempo de *turnaround* razonables. Este valor

tiene que ver con las condiciones de estabilidad de la cola. Con tamaños de cola mayores las aplicaciones paralelas se acumulan en la cola de espera a mayor velocidad de la que el cluster es capaz de servirlas, aumentando los tiempos de espera. A tamaños de cola pequeños, los trabajos se acumulan en la cola de espera del meta-planificador aumentando allí su tiempo de espera o bien son asignados directamente a otros clusters que proporcionan peores rendimientos.
A raíz de estos resultados, en el presente trabajo (sección 4.5) proponemos un mecanismo de auto-regulación del tamaño máximo de trabajos en la cola de espera, que permite a cada cluster limitar el acceso de nuevos trabajos a la cola de espera cuando el cluster esté saturado.

Además, se ha propuesto una nueva técnica de selección de trabajos, denominada *NB.RS* (***N**on-**B**locking with **R**escheduling*), con el fin de eliminar los efectos negativos del bloqueo de trabajos en la cola del meta-planificador, y se ha realizado un estudio de los resultados de rendimiento de las aplicaciones paralelas y desviación en las predicciones comparando nuestra técnica con las más habituales en la literatura.

NB.RS persigue dos objetivos distintos. En primer lugar, reducir los efectos negativos producidos por el bloqueo de las aplicaciones paralelas en la cola del meta-planificador, adelantando las aplicaciones que pueden ser asignadas en otros clusters. En segundo lugar, reducir los efectos negativos del dinamismo de la carga local, re-asignando las aplicaciones en la cola del meta-planificador cuando se detecta un cambio de estado en los recursos.
Los resultados demuestran que con la utilización de nuestra propuesta (*NB.RS*) se consigue un aumento en el rendimiento de las aplicaciones paralelas incluso en situaciones con dinamismo de la carga local.
Las aportaciones realizadas en este apartado aparecen publicadas en:

JL. Lérida, F. Solsona, F. Giné, M. Hanzich, JR. García, P. Hernández
MetaLoRaS: A Re-scheduling and Prediction MetaScheduler for Non-dedicated Multiclusters.
Recent Advances in Parallel Virtual Machine and Message Passing Interface (EuroPVM/MPI'07). LNCS 4757, p.195-203.

- Con el fin de poder compartir los recursos libres de distintos cluster para la ejecución de aplicaciones paralelas, se ha desarrollado una técnica de asignación basada en un modelo de programación entera binaria (*Mixed Integer Programming*). El modelo realiza la asignación de una aplicación paralela a un conjunto de recursos de distintos clusters con dos objetivos, la obtención del mejor rendimiento de la aplicación paralela y la no saturación de los ca-

nales de comunicación.
Esta técnica se basa en el modelo analítico del tiempo de ejecución propuesto por nosotros, que considera la heterogeneidad y la ocupación de los recursos tanto de cómputo como de comunicación.
Los resultados obtenidos demuestran que para un entorno Multicluster compuesto por un conjunto reducido de clusters con unos pocos centenares de nodos por cluster el modelo presentado permite obtener asignaciones en tiempos razonables en situaciones con elevada carga. Para la asignación de aplicaciones en situaciones con un gran número de recursos y alta carga en donde el tiempo de respuesta es importante, se propone una variación del modelo que aunque no proporciona la mejor asignación posible, asegura la no saturación de los canales de comunicación, evitando las consecuencias negativas sobre el rendimiento tanto de las aplicaciones paralelas como del sistema.
Las aportaciones realizadas en este apartado aparecen publicadas en:

JL. Lérida, F. Solsona, F. Giné, JR. García, P. Hernández
Resource Matching in Non-dedicated Multicluster Environments.
High Performance Computing for Computational Science (VECPAR-08). LNCS 5336, p.160-173.

Con el fin de poder sacar aún mayor provecho de los recursos libres, en le presente trabajo proponemos un refinamiento de la tarea de meta-planificación. La técnica consiste principalmente en tratar de co-asignar aquellos trabajos que se encuentran en la cola de algún cluster a la espera de recursos libres. El objetivo principal es obtener mayores rendimientos de las aplicaciones paralelas reduciendo su tiempo de espera y un mayor aprovechamiento de los recursos del Multicluster.

Líneas abiertas

El presente trabajo profundiza en distintos aspectos de la tarea de meta-planificación. Durante su desarrollo hemos trabajado en múltiples disciplinas (estadística, investigación operativa, análisis de series temporales, modelización, simulación, etc) que nos han permitido obtener una visión mas amplia de hacia donde se pueden extender nuestras aportaciones. Las líneas abiertas de mayor interés en el ámbito de la meta-planificación son la predicción y la co-asignación.

La obtención de una mayor precisión en las predicciones proporciona innumerables ventajas en muchos ámbitos de la planificación. La mejora de la toma de decisiones que permita cumplir con los objetivos de rendimiento marcados por el administrador

del sistema, la posibilidad de ofrecer al usuario QoS e implementar sistemas de costes, la posibilidad de monitorizar y ajustar el comportamiento de aplicaciones y/o sistemas a unos requerimientos específicos, son solo algunos de los muchos ejemplos. Con el objetivo de obtener mejores prestaciones del sistema de predicción se propone las siguientes lineas de trabajo futuro:

- La incorporación de mayor información sobre el estado de los recursos en los métodos de estimación estadísticos, puede aportar mayor precisión a los estimadores. Los métodos estadísticos buscan similitudes de la situación actual con la historia pasada. La principal dificultad está en determinar que parámetros son relevantes a la hora de comparar dos experiencias.
  Existen numerosos trabajos [STF99, SW02, LGTW04] que mediante la utilización de algoritmos genéticos, búsquedas tabú o greedy, realizan una búsqueda de los parámetros más significativos. La mayoría de estos trabajos no utilizan información sobre la heterogeneidad de los recursos o su capacidad y se centran únicamente en las características de las aplicaciones.

- Uno de los mayores inconvenientes a la hora de proponer nuevos mecanismos de predicción, es que en general la información que se solicita para poder llevar a cabo las estimaciones no está siempre disponible. Una alternativa que se propone en varios trabajos [Din99, Din01, Wol03, WSH00] para solucionar este problema es no realizar una estimación del rendimiento de la aplicación sino realizar estimaciones sobre la ocupación de los recursos. Estas estimaciones pueden ser utilizadas como información para predictores que tratan de estimar el rendimiento de las aplicaciones.
  La técnica *SPDN,* propuesta en el presente trabajo, utiliza un simulador que guiado por un modelo analítico permite estimar los tiempos de ejecución de todas las aplicaciones que se encuentran en el sistema. Cada vez que una aplicación finaliza se aproxima el nuevo estado del sistema considerando la información que se tiene de los recursos que consume la aplicación que ha finalizado. Esta información que normalmente no es exacta produce ciertos errores en la estimación de la ocupación de los recursos. En el presente trabajo, se ha demostrado que el error cometido en cada aplicación se propaga por el resto de aplicaciones en la cola de espera.
  Un modo de evitar la propagación de estos errores sería que a cada paso de simulación se realizará una estimación del estado de los recursos basándonos en el análisis de series temporales sobre información obtenida mediante la monitorización del uso de los recursos.

- El objetivo del análisis de series temporales es aproximar una determinada ecuación matemática que permite representar el comportamiento del sistema.

A partir de esta ecuación se puede intentar estimar en un determinado instante de tiempo la situación de ocupación de los recursos.
Realizando un estudio de los tiempos de ejecución obtenidos por una determinada aplicación en el pasado, podría aplicarse de forma análoga el análisis de series temporales para obtener una ecuación que refleje el comportamiento del tiempo de ejecución en el pasado. Esta ecuación podría ser utilizada como un estimador del tiempo de ejecución de las aplicaciones paralelas.

Otra de las líneas abiertas con un fuerte interés en la actualidad es la *co-asignación*. El problema de la asignación óptima de una aplicación a los recursos es un problema NP-completo con un coste computacional elevado. En la actualidad existen muchas alternativas para tratar los problemas de este tipo que permiten obtener soluciones en mucho menor tiempo. Estas alternativas en el caso de la asignación en entornos Multicluster heterogéneos y no dedicados no está siendo suficientemente explotada. En este sentido proponemos las siguientes líneas de trabajo:

- Utilizar métodos meta-heurísticos para la búsqueda de la asignación óptima considerando el modelo MIP presentado en el presente trabajo.

- El modelo MIP que se ha propuesto en el presente trabajo puede ser planteado como un problema multi-criterio (MOP). Existen en la literatura distintos métodos de resolución para problemas multi-criterio. Uno de los más utilizados es la programación por metas, donde se trata de relajar la condición de búsqueda del valor óptimo proponiendo distintos niveles de aspiración a las funciones objetivo de los distintos criterios. De este modo se puede aproximar una solución que satisface las necesidades del usuario en menor tiempo.

- Uno de los problemas más comunes en las técnicas de predicción es que se basan en la historia pasada. Pero muy pocos tienen en cuenta lo que pueda suceder en el futuro. De modo que el error en la predicción siempre vendrá condicionado por el tipo de aplicaciones que en un futuro sean asignadas a los mismos recursos.
  De modo que sería interesante poder añadir al modelo información sobre la probabilidad de que venga un determinado tipo de aplicación, ayudando de esta forma a tomar mejores decisiones de planificación que permitan aumentar el rendimiento de las aplicaciones paralelas considerando el futuro próximo.

Apéndice A

Diseño del sistema MetaLoRas

Partiendo de la arquitectura que se ha diseñado para el entorno M-CISNE en el apartado 2.2, en esta sección se detalla el diseño del sistema MetaLoRaS desarrollado en el presente trabajo. En la figura A.1, se pueden observar los módulos que componen la arquitectura diseñada para MetaLoRaS: *Gestor de Colas* (*Input Queue* o *Low-level Queue*), *Políticas* (Policies), *Sistema de Admisión* (Admission System) y *Sistema de control de clusters* (SCC), también denominado *Multicluster Controller* (MCC).

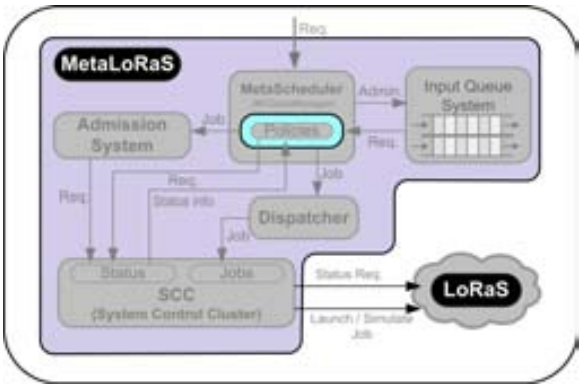


Figura A.1: Módulos funcionales de MetaLoRaS

En la sección A.1 se describen los componentes estáticos del diseño, conjunto de clases necesarias para implementar el sistema. Las responsabilidades de cada una de estas clases, la interrelación entre ellas y su distribución en los módulos están representados en la figura A.1.

En el apartado A.2 se describen los componentes dinámicos del diseño. Esto incluye los diagramas de secuencia y actividad que permiten comprender con mayor detalle el funcionamiento interno del sistema.

En la última sección (sección A.3) se explica como se integra LoRaS en el sistema M-CISNE, mediante una explicación de los mecanismos de interacción establecidos entre los subsistemas LoRaS y MetaLoRaS.

A.1. Definición estática de MetaLoRaS

El conjunto de clases que compone el sistema diseñado se ha dividido en dos módulos, el módulo de Gestión y el de Políticas. El módulo de Gestión contiene principalmente las clases centrales encargadas de gestionar y controlar los distintos componentes del subsistema MetaLoRaS: colas, trabajos, nodos, despachador, etc. El módulo de Políticas en cambio, contiene el conjunto de clases encargadas de definir y gestionar las políticas de planificación.

En la figura A.2 se pueden observar los módulos (paquetes) contenedores de las distintas clases, sus dependencias y las clases contenidas en cada uno de ellos.

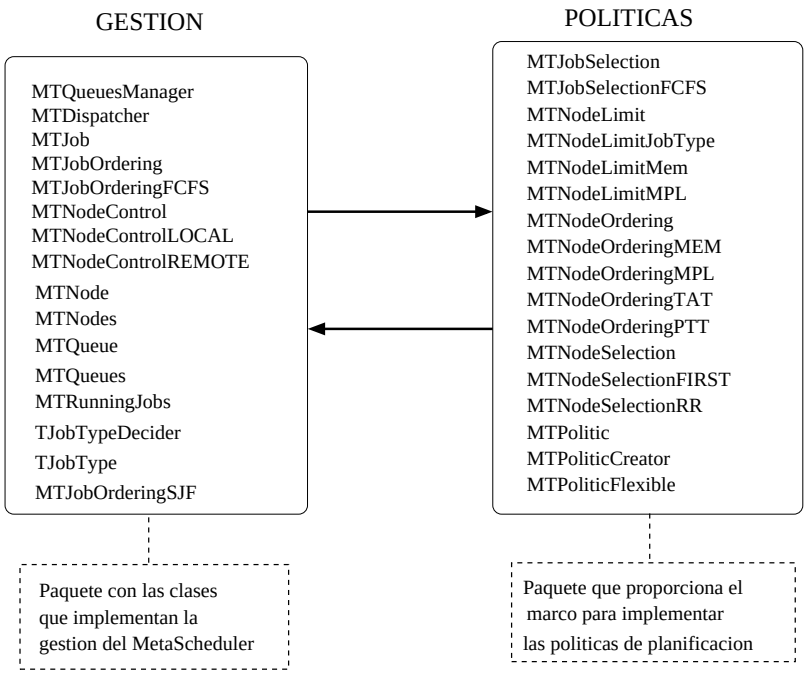


Figura A.2: Paquetes del subsistema MetaLoRaS

A.1.0.1. Módulo de Gestión

El módulo de gestión contiene el conjunto de clases que se encarga de gestionar los componentes principales del sistema de planificación. En la figura A.3 se observa el conjunto de clases agrupadas según su responsabilidad y funcionalidad. Según esta clasificación, las clases se han agrupado en: clases principales, clases para la gestión de las colas, clases para la gestión de recursos y eventos, y clases representativas de elementos externos. Las clases contenidas en cada uno de estos grupos y sus responsabilidades son:

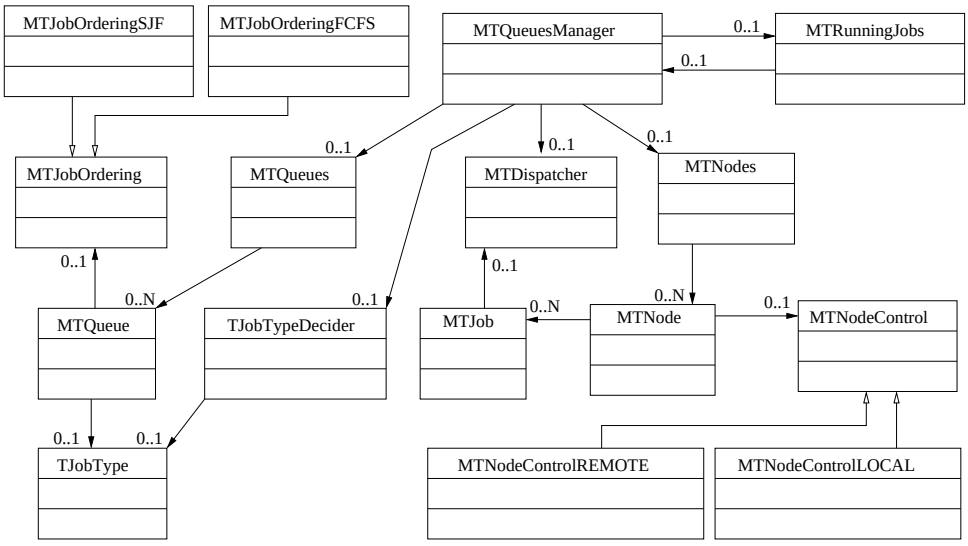


Figura A.3: Módulos de gestión del subsistema MetaLoRaS

1. ***Clases principales***: estas clases representan el núcleo de la planificación. Contienen y gestionan las clases principales encargadas de la planificación de los trabajos. La clase *MTQueuesManager* es el núcleo del sistema MetaLoRaS. Esta es una clase contenedora de los componentes principales del sistema de planificación: colas, nodos, controlador de eventos y clasificador de trabajos. *MTDispatcher* es la responsable de despachar el trabajo en el nodo seleccionado por la política de planificación definida y actualizar el estado del planificador con el resultado del proceso.

2. ***Clases para la gestión de las colas***: forman parte del módulo gestor de colas y agrupan aquellas clases relacionadas con el proceso de encolamiento de un trabajo entrante en el sistema. La Clase *MTQueues* es la clase contenedora de las colas del sistema. Esta clase mantiene una lista de las colas definidas en el

MetaLoRaS. *MTQueue* se encarga de almacenar los trabajos en las colas, eliminarlos y entregárselos a la política de planificación asociada cuando recibe un evento de planificación por parte del MetaScheduler. *TJobTypeDecider* y *TJobType* definen el tipo de trabajo cuando éste llega al sistema. Esto permite al planificador asignar el trabajo a la cola que le corresponda según su tipo. *MTJobOrdering* es una clase abstracta que permite definir distintas políticas de ordenación de trabajos en las colas. *MTJobOrderingFCFS y MTJobOrderingSJF* implementan políticas de ordenación de trabajos.

3. ***Clases para el control de recursos y eventos***: forman parte del sistema de control de clusters y agrupan aquellas clases encargadas de controlar y comunicarse con los planificadores de nivel inferior, LoRaS. La clase MTMetaNodeControl es una clase abstracta que proporciona los mecanismos de control de un nodo (cluster). MTMetaNodeControlLOCAL y MTMetaNodeControlREMOTE implementan los mecanismos de control y comunicación para nodos (clusters) locales y remotos respectivamente. La clase MTRunningJobs gestiona información sobre las aplicaciones en ejecución y se encarga de recibir cualquier evento sobre el cambio de estado de una aplicación en el subsistema LoRaS (llegada, ejecución, finalización, error).

4. ***Clases de representación de componentes externos***: estas clases representan componentes externos del sistema controlados por el planificador. La clase *MTJob* representa un trabajo. Esta clase gestiona información estática y dinámica de los trabajos. La información estática es la información que se conoce de la aplicación antes de ejecutarla: parámetros, consumo de los recursos, tipo de aplicaciones, etc. La información dinámica gestiona información sobre el estado de ejecución de la aplicación. La clase *MTNode* representa un nodo (cluster). Esta clase mantiene información tanto del estado de los recursos del cluster que controla como de las aplicaciones paralelas que se ejecutan. La clase *MTNodes* es una clase contenedora de Nodos. Esta clase almacena una lista de nodos y se usa tanto para gestionar una lista de los clusters controlados por MetaLoRas como la lista de clusters candidatos para la planificación de una aplicación paralela.

A.1.1. Módulo de Políticas

El módulo de políticas contiene el conjunto de clases que permiten la implementación de nuevas políticas de planificación. Una política de planificación se compone de un conjunto muy variado de procesos. Uno de los objetivos de MetaLoRaS es proporcionar un mecanismo que permita implementar e incorporar de forma sencilla nuevas políticas de planificación.

Para poder hacer esto, se ha realizado un diseño basado en una clase central (*MTPoliticFlexible*) que proporciona un entorno donde incorporar el conjunto de procesos que la política va a seguir para tomar una decisión. Además se dispone de un conjunto de clases abstractas, donde cada una proporciona mecanismos para implementar uno de los procesos de la política de planificación. Mediante la implementación de clases derivadas de cada clase abstracta podemos implementar distintas técnicas para llevar a cabo el proceso de planificación.

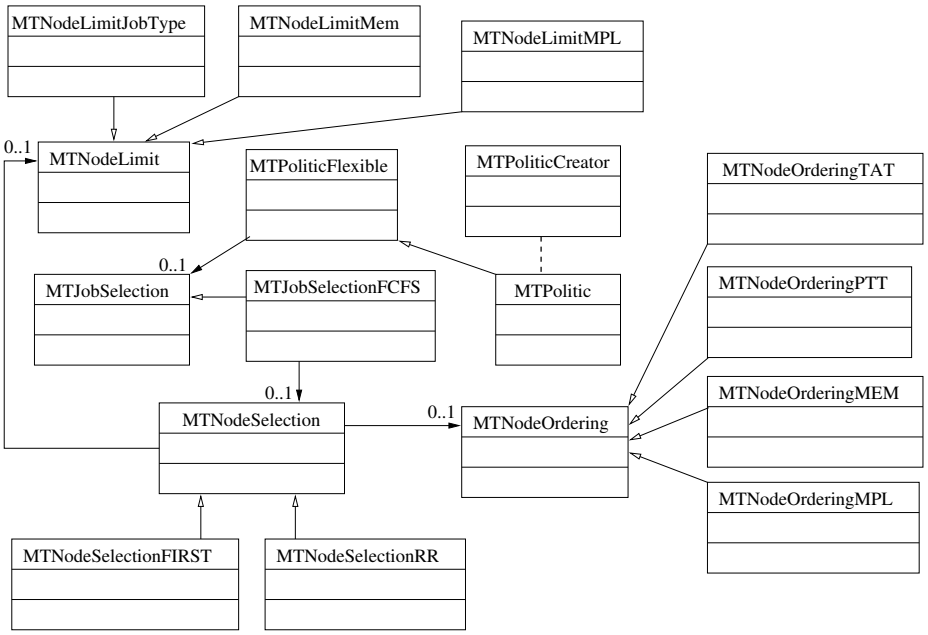


Figura A.4: Módulos de planificación del subsistema MetaLoRaS

En la figura A.4 se observa el conjunto de clases del módulo de políticas agrupadas según su responsabilidad y funcionalidad. Según esta clasificación, las clases se han agrupado en: política de planificación, políticas de selección de trabajos, políticas de limitación de clusters, políticas de ordenación de clusters y políticas de selección de clusters. Las clases contenidas en cada uno de estos grupos y sus responsabilidades son:

1. ***Política de planificación***: las clases contenidas en este grupo son las que proveen la interfaz de planificación que interactúa con el módulo Gestor de Colas

(Input Queue). Estas clases proporcionan un marco dentro del cuál se gestionan las distintas técnicas que se aplican durante el proceso de planificación. La clase principal de este marco es la clase base *MTPolitic*, la cual posee una clase de fabricación (factory) denominada *MTPoliticCreator*, responsable de crear instancias de subclases de *MTPolitic* que implementen las políticas de planificación deseadas. En nuestro caso particular, con el fin de proporcionar un entorno que permita la incorporación de nuevas técnicas de planificación de forma sencilla, se ha implementado una sola política configurable que hemos llamado *Flexible*, implementada a partir de la clase *TPoliticFlexible*.

2. ***Políticas de selección de trabajos***: las clases contenidas en este grupo implementan distintas técnicas para seleccionar el siguiente trabajo de la cola de espera que va a ser panificado. La clase *MTJobSelection* es la clase abstracta que proporciona los mecanismos para llevar a cabo la técnica de selección. Mediante clases derivadas podemos implementar distintos mecanismos de selección. En el presente trabajo se ha utilizado únicamente la política FCFS (MTJobSelectionFCFS). Esto es así porque el objetivo principal del trabajo es validar el funcionamiento del sistema diseñado y la incorporación de políticas basadas en la predicción. Para poder hacer uso de la predicción en la planificación es necesario utilizar técnicas que no alteren el orden de los trabajos en las colas.

3. ***Políticas de limitación de clusters***: en el momento de seleccionar un conjunto de clusters donde poder ejecutar una aplicación paralela determinada, se debe tener en cuenta el uso de los recursos en cada cluster para poder respetar la interactividad del usuario local. Para poder hacer esto, el sistema permite establecer límites para el uso de recursos por parte de la aplicaciones paralelas.

   Aunque en el presente trabajo no se ha hecho uso de estas técnicas, se han implementado algunas de ellas que podrían ser útiles en trabajos futuros. Es importante destacar que la información que se tiene del uso de los recursos es a nivel de cluster y no de cada nodo del cluster en particular. Esto significa que se puede dar el caso que existan nodos muy ocupados y otros totalmente libres con suficientes recursos para ejecutar la aplicación paralela.
   Estas políticas de limitación nos proporcionan únicamente métodos de preselección que pueden ayudar a reducir de forma sencilla el espacio de posibles candidatos. Esto puede ser útil para Multiclusters con gran cantidad de recursos. En aquellos con menor número de recursos sería necesario implementar técnicas que permitieran mecanismos de exclusión más precisos.
   MTNodeLimit es la clase abstracta que proporciona los mecanismos para el filtrado de los nodos candidatos. MTNodeLimitJobType permite descartar un

cluster según el tipo de trabajo que está ejecutando en aquel momento, MTNodeLimitJobMEM permite descartar un cluster según el nivel de utilización de recursos de la Memoria, y por último MTNodeLimitJobMPL permite descartar un cluster según el máximo MPL de entre todos los nodos que lo componen.

4. ***Políticas de ordenación de clusters***: una vez aplicadas las políticas de limitación, disponemos de un conjunto de clusters candidatos donde podemos ejecutar las aplicaciones paralelas respetando los recursos reservados a los usuarios locales. En aquellos casos en que el conjunto de clusters candidatos sea mayor que uno, puede ser conveniente establecer criterios que permitan distinguir unos clusters de otros. Las políticas de ordenación ordenan el conjunto de clusters candidatos según un criterio determinado.
   La clase MTNodeOrdering es una clase abstracta que proporciona mecanismos de ordenación sobre una lista de nodos. La implementación de distintos criterios se lleva a cabo mediante la implementación de clases derivadas. MTNodeOrderingMEM ordena la lista de clusters candidatos de menor a mayor, según el promedio de utilización de los recursos de Memoria. MTNodeOrderingMPL ordena la lista de candidatos de menor a mayor, según el MPL máximo del cluster. Este criterio teniendo en cuenta que LoRaS pretende maximizar el uso de los recursos balanceando las aplicaciones paralelas, puede ser un buen indicativo sobre el estado de ocupación del cluster.
   MTNodeOrderingTAT ordena la lista de candidatos de menor a mayor, según el tiempo de retorno estimado (*Turnaround Time*, *TAT*) por el simulador de cada cluster. MTNodeOrderingPTT ordena la lista de candidatos de menor a mayor, según el tiempo de retorno ponderado (Ponderated Turnaround Time, PTT).

   Todas estas políticas se comunican, a través del SCC (System Cluster Control), con los módulos LoRaS para obtener la información necesaria que les permite aplicar los distintos criterios. Una explicación detallada del mecanismo de planificación general y la implementación de las políticas basadas en predicción se realiza en el capítulo 2.5.

5. ***Políticas de selección de clusters***: Una vez ordenada la lista de clusters candidatos, se necesita establecer un criterio para decidir en que orden se seleccionan los clusters de la lista. La clase MTNodeSelection es una clase abstracta que proporciona mecanismos de selección dentro de una lista. Las políticas de selección de nodos implementadas en nuestro trabajos son dos: JFIRST (MTNodeSelectionJFIRST) y RR (MTNodeSelectionRR). La primera escoge siempre el primer nodo de la lista de cluster candidatos, mientras que la segunda realiza un recorrido secuencial por la lista de candidatos.

Es importante destacar que, aunque la implementación de la mayoría de las técnicas aplicadas en las distintas fases de planificación es sencilla, la característica más destacable de este diseño radica en el hecho que mediante el uso de múltiples, conocidas y sencillas técnicas, somos capaces de desarrollar complejas políticas que permiten aplicar un gran conjunto de criterios e incorporar nuevas técnicas de forma sencilla.

A.2. Definición Dinámica de MetaLoRaS

Para proporcionar una mejor comprensión del funcionamiento del sistema desarrollado es importante detallar el conjunto de acciones que los distintos componentes llevan a cabo como consecuencia de los eventos que van surgiendo en el sistema. El conjunto de eventos tratados en el sistema MetaLoRaS los podemos agrupar en dos grupos, internos y externos. Los eventos internos son aquellos producidos por el propio sistema durante la gestión de los distintos componentes del sistema y la planificación de aplicaciones paralelas. Los eventos externos son aquellos producidos por peticiones de los usuarios paralelos o bien por los módulos LoRaS que gestionan los planificadores locales del Multicluster.

En esta sección se describe los procedimientos llevados a cabo por el sistema a la llegada de algunos de estos eventos y se detalla como se realiza la comunicación entre los distintos componentes internos de MetaLoRaS así como entre el MetaLoRaS y subsistema LoRaS.

A.2.1. Proceso de inserción de un trabajo en el sistema

Dado que el objetivo principal de la herramienta implementada es la planificación de trabajos paralelos en un entorno Multicluster, una de la primeras acciones que debemos analizar es la inserción de un trabajo paralelo en el sistema.

MetaLoRaS ofrece al usuario paralelo una interfaz que le permite interactuar con el sistema. El proceso de inserción se inicia cuando el usuario paralelo realiza una llamada a la función *scheduleJob(string)*, indicando el trabajo que desea ejecutar. Cuando el MetaScheduler (*MTQueuesManager*) recibe esta petición, lo primero que hace es caracterizar el trabajo que se quiere lanzar, para poder colocarlo en la cola correspondiente (*getJobType(TJTSpec)*), si es que existe más de una. Esta caracterización puede hacerse en función de la cantidad de procesadores solicitados, el tiempo de ejecución, la cantidad de memoria requerida, etc. Todos estos parámetros se definen en la configuración del sistema cuando se inicia.

Una vez caracterizado el trabajo se obtiene el tipo de despachador necesario para lanzarlo (*getInstance(int, TRunningJobs*)*). En el trabajo desarrollado se ha hecho uso de un único despachador independiente del tipo de aplicación que permite trabajar con aplicaciones paralelas tanto PVM como MPI. Aún así, pensando en futuras necesidades de portabilidad del sistema se ha tenido en cuenta la posibilidad de implementar distintos tipos de despachador. No con el objetivo de distinguir aplicaciones PVM o MPI, sino más bien para poder soportar distintos sistemas de planificación a nivel de cluster.

Si las características del trabajo no se adecúan a ninguna cola en particular, el trabajo es rechazado. En caso contrario, el trabajo es insertado en la cola correspondiente en función de la caracterización realizada (*scheduleJob(TJob*, TQueue*)*). Una vez insertado el trabajo en la cola, ésta utiliza la política de planificación para lanzar o almacenar el trabajo hasta que sea posible lanzarlo, en función de los criterios establecidos por la política configurada (*schedule(TEvent)*).

A.2.2. Proceso de Planificación

El sistema planificador se encarga tanto de la gestión de la planificación como de la gestión de los eventos derivados de ella. Los eventos derivados de la planificación son: llegada de una aplicación al sistema (*ARRIVE*), finalización de una aplicación (FINISH), y notificación de un error en la ejecución de una aplicación (*ERROR*).

El responsable de detectar y gestionar estos eventos es el sistema planificador (*TQueuesManager*). Tal y como se ha descrito en la sección anterior, el evento de llegada de una aplicación se recibe a través de la interfaz del usuario paralelo. Los eventos resultado de la planificación de un trabajo en los subsistemas LoRaS son recibidos y luego enviados al MetaScheduler a través del Sistema de control de Clusters (SCC).

Una vez recibido el evento, el MetaScheduler lo comunica a la política de planificación *(Politic::schedule(TEvent)*), que se encarga de gestionarlo en función de sus especificaciones.

Hay que destacar que el SCC dispone de una lista (instancia *MTNodes*) con todos los clusters que componen el Multicluster. En cada elemento de esta lista (*MTNode*) se guarda información sobre los recursos del cluster, y el estado de las aplicaciones. Esta información debe ser actualizada a la llegada de nuevos eventos.

Cuando el evento llega a la política, ésta lo evalúa para determinar las acciones que debe realizar. Cuando se trata de un evento de *ERROR* o *FINISH,* las acciones son prácticamente las mismas. En primer lugar, se obtiene del SCC la lista de nodos donde se ha lanzado la aplicación. A continuación, se elimina la aplicación de la

lista de trabajos en ejecución que el SCC (Sistema de Control de Clusters) mantiene asociada a cada cluster y finalmente registra en un fichero de *log* el evento producido.

Cuando se produce un evento de llegada de una nueva aplicación (*ARRIVE*), la política busca el siguiente trabajo a planificar (*setStartableJob(MTNodes *, int)*). En el caso que haya trabajos anteriores, iniciará la planificación del trabajo al que le corresponda. En caso contrario planificará el trabajo recién llegado.

Las tareas que la política lleva a cabo durante la planificación son las siguientes: localización del siguiente trabajo a planificar (*getNextJob()*), control del sistema de admisión (*checkJobStartability(Job, NodesParam)*) y planificación del trabajo (*dispatch(LaunchingNodes)*).

En primer lugar, la política consulta al sistema de colas (*TQueues*) mediante la política de selección de trabajos (*MTJobSelection*) el siguiente trabajo a planificar. Una vez localizado el trabajo, el *Sistema de Admisión* comprueba si existe algún cluster con suficientes nodos para ejecutar el trabajo (*checkJobStartability(MTJob*, string)*). En el caso que no sea así, el trabajo es rechazado (*REJECT*). En caso contrario, el sistema de admisión comprueba entre los cluster candidatos si existe alguno con suficientes recursos de memoria. Si no hay ningún cluster que cumpla este requerimiento, el trabajo se almacena en la cola a la espera de que haya recursos suficientes. En caso contrario, el trabajo es planificado (*TDispatcher*::*dispatch(TNode**)*) mediante el despachador asociado.

A continuación, la política recibe del despachador un código indicando si la planificación ha funcionado correctamente y genera uno de los eventos siguientes: *SCHED* en el caso que la planificación en LoRaS sea correcta y ERROR en caso contrario. En este último caso, el código de error es almacenado junto con el resto de la información de la aplicación (*MTJob*).

A.2.3. Lanzamiento de aplicaciones

En la sección anterior se ha mostrado el funcionamiento del sistema hasta el momento en que se decide despachar una aplicación en un cluster determinado. En este punto, el despachador es el encargado de lanzar la aplicación al planificador local del cluster seleccionado.

En nuestro trabajo hemos considerado por el momento que todos los clusters están controlados por nodos LoRaS. El despachador de trabajos gestionará por tanto el lanzamiento de aplicaciones con módulos LoRaS.

Cuando un trabajo es planificado (*TDispatcher*::*dispatch(TNode**)*), el despachador debe comunicarse con el módulo LoRaS del cluster donde ha sido asignada la

aplicación. Para poder comunicarse con los módulos LoRaS, el despachador utiliza la clase de control de nodos (*MTNodeControl*) proporcionada por el *Sistema de Control de Clusters* (SCC).

En este punto es importante recordar que por razones de flexibilidad y portabilidad, los módulos LoRaS que componen el Multicluster pueden configurarse de forma local o remota. Es por este motivo que el despachador obtiene el controlador (*MTMetaNodeControl::getInstance(TNode*)*) correspondiente al módulo LoRaS escogido para la planificación.

Una vez obtenido el controlador, el trabajo es entregado al módulo LoRaS correspondiente (*dispatch(TdispatchData)*). En este punto, el módulo LoRaS recibe el trabajo e inicia el proceso de planificación. El despachador comprueba el código devuelto por el módulo LoRaS. Un código positivo indica que el trabajo fue planificado correctamente mientras que un código negativo indica que ha habido un error.

Si el trabajo se ha planificado correctamente, el despachador debe actualizar cierta información sobre este proceso. En primer lugar se registra en el trabajo, clase *MTJob*, el identificador que LoRaS le ha asignado a la aplicación y el nodo LoRaS donde se ha lanzado la aplicación. Por último, se debe añadir la aplicación a la lista de trabajos en ejecución dentro del Multicluster (*RunningJobs->addJob(MTJob*)*). Esta lista es gestionada por la clase *MTRunningJobs*. Para finalizar, el despachador devuelve el control a la política de planificación.

A.2.4. Finalización de un trabajo

Una vez lanzada una aplicación en un determinado cluster, se necesitan mecanismos que permitan al MetaLoRaS conocer cuando una aplicación ha finalizado su ejecución. Esto permite al MetaLoRaS mantener actualizada la información sobre el estado de las aplicaciones que se están ejecutando en el Multicluster.

Para poder obtener esta información se ha dotado a LoRaS de mecanismos de notificación que proporcionan a MetaLoRaS información sobre los eventos relacionados con la ejecución de las aplicaciones paralelas.

Existen dos formas de finalizar un trabajo en LoRaS, con normalidad o debido a un error. Cuando un trabajo finaliza en LoRaS, este envía (*metafinishJob(TJob*)* o *metaabortJob(TJob*)*) un mensaje a MetaLoRaS indicando la aplicación que ha finalizado y el tipo de finalización. El encargado de recibir las peticiones de finalización de una aplicación es el *Sistema Controlador de Clusters* (SCC). Este sistema recibe a través de *MTRunningJobs* la notificación del evento (*MTRunningJobs::checkFinished()*).

Una vez recibida la notificación, se actualiza la información en el SCC (*finishedJob(TJob*, int)*) eliminando la aplicación finalizada de la lista de trabajos en ejecución (*removeJob(TJob*)*). A continuación, el SCC comunica el evento al MetaScheduler (*finishedJob(TJob*, int)*). Finalmente, la política de planificación (*schedule(event)*) actualiza la información correspondiente.

A.3. Interacción de MetaLoRaS con LoRaS

En la sección anterior se ha descrito como se llevan a cabo los procesos de aceptación y planificación de aplicaciones paralelas en el sistema MetaLoRaS. Además se explica como se comunican los componente internos de MetaLoRas y como se comunica MetaLoRaS con los componentes externos, usuario y subsistema LoRaS, durante la planificación de las aplicaciones.

Aún así, la interacción entre MetaLoRaS y LoRaS no consiste solo en mecanismos para poder lanzar un trabajo o para recibir la notificación de la finalización de una aplicación en un cluster. El sistema MetaLoRaS necesita disponer de información actualizada sobre el estado de los recursos y de una aplicación en un cluster, o bien preguntar acerca del estado de las colas en cada cluster e incluso solicitar al simulador de un cluster estimaciones de tiempo de ejecución de alguna aplicación paralela. Para poder proporcionar estos mecanismos de control e información ha sido necesario adaptar el interfaz de LoRaS.

En esta sección se describen los procesos de comunicación necesarios entre MetaLoRaS y LoRaS para poder llevar a cabo otras actividades de gestión y control sobre LoRaS, paralelas a la planificación.

A.3.1. Consulta de información sobre el estado de los nodos

La información del estado de los nodos se almacena en el Sistema de Control de Clusters (SCC). Este módulo gestiona una lista de los nodos LoRaS del Multicluster, con información sobre su ubicación en el sistema, su estado y las aplicaciones que se están ejecutando.

Mantener esta información actualizada es de vital importancia para la gestión de los nodos y para la toma de decisiones en la planificación. La necesidad de mantener actualizada esta información provoca que las consultas a los módulos LoRaS sean frecuentes. Además, el número de subsistemas LoRaS del Multicluster puede influir negativamente en el rendimiento del sistema debido al incremento del volumen de comunicaciones.

Para ello hemos diseñado un sistema de *buffering* que para un volumen pequeño de información a transmitir minimiza al máximo la intrusión de la red de las comunicaciones de control del sistema.

Cuando el MetaScheduler desea obtener el estado de un cluster, solicita esta información al módulo SCC, aplicando la función *getStatus()* sobre la representación (*MTNode*) del cluster en el SCC.

Para evitar la consulta masiva de información sobre el estado de los clusters y evitar así afectar negativamente al rendimiento del sistema, la función getStatus() implementa técnicas de *buffering* controladas por temporizador. Cuando se consulta el estado del cluster por primera vez, se solicita la información a LoRaS a través de la clase de Control correspondiente (*MTMetaNodeControl::getNodeStatusData(TNode*)*).

La información es recibida en memoria en una estructura de datos (*TQMStatus*). En consultas sucesivas, en un intervalo de tiempo *TimeStamp*, la información que se devuelve ya no se consulta a LoRaS sino que se obtiene de la almacenada en memoria. Pasado el límite de *TimeStamp*, la información se vuelve a consultar al sistema LoRaS, actualizando la estructura *TQMStatus*. El intervalo *TimeStamp* es configurable y nos permite ajustarlo en función de parámetros como el tamaño del Multicluster o el nivel de saturación del mismo.

A.3.2. Consulta de información sobre el estado de un trabajo

Otra de las funciones que implementa el sistema es la consulta del estado de una aplicación paralela. Esta información es especialmente útil para que el usuario paralelo pueda realizar un seguimiento de la aplicación. Por otro lado, esta información es imprescindible para llevar a cabo tareas de estimación. En el presente trabajo, dado que no se han implementado mecanismos de estimación a nivel de MetaLoRas, esta función no se hace tan imprescindible. Sin embargo, ha sido implementada en previsión de necesidades futuras.

El funcionamiento es similar a la consulta de información sobre el estado de los clusters. En este caso, el MetaScheduler consulta la información de la aplicación para saber en que cluster (*MTNode*) se está ejecutando. Una vez conocido el cluster, se solicita información del estado del trabajo al SCC, aplicando la función *getJobStatus()* sobre la representación (*MTNode*) del cluster en el SCC.

El SCC solicita, a intervalos de tiempo *TimeStamp*, la información al subsistema LoRaS mediante la clase de control correspondiente (*MTMetaNodeControl::getJobStatusData(TJob*)*). La información se almacena en memoria a través de la estructura de datos *MTJobStatusData*.

A.3.3. Utilización del módulo de simulación de LoRaS

Las políticas de planificación implementadas en este trabajo que utilizan mecanismos de predicción, basan sus decisiones en la estimación del tiempo de retorno de la aplicación en los diferentes clusters. Para poder calcular el tiempo de retorno se ha utilizado un sistema de simulación incorporado dentro del subsistema LoRaS. Para poder hacer uso de este simulador, necesitamos que la política de planificación pueda comunicarse con LoRaS para poder enviarle la aplicación y recibir de este el resultado de la estimación.

Cuando una política desea obtener una estimación de un determinado cluster, proporciona información del trabajo al nodo que representa el cluster dentro del módulo SCC y solicita su estimación (*getSimJobData(MTJob*)*).

Cada vez que se realiza una estimación de un trabajo en un cluster, la información que se devuelve se almacena en el módulo SCC, dentro del nodo (*MTNode*) que representa al cluster. Cada vez que un política solicita información sobre la estimación de una aplicación a un cluster, el módulo SCC consulta si existe la estimación. Si existe devuelve directamente el resultado. En caso contrario, se solicita la estimación al subsistema LoRaS a través de la clase de Control correspondiente (*MTMetaNodeControl::getSimulationData(MTJob*)*).

Una vez recibido el resultado de la simulación, el módulo SCC almacena la información resultante de la estimación en el nodo representativo del cluster (*insert(pair<int,TsimulaData>)*). Teniendo en cuenta el gran número de cálculos de estimación que se pueden llegar a hacer en cada uno de los clusters, es necesario proporcionar un acceso eficiente a la información.

A.3.4. Consulta del estado de las colas de LoRaS

Esta función se implementa para resolver algunos problemas derivados de la característica experimental del entorno. Cuando finaliza la ejecución de una carga se eliminan las instancias principales de MetaLoRaS. Esto puede suceder incluso antes de que MetaLoRaS haya finalizado algunas de las tareas de gestión de eventos o el registro de los ficheros de *log*. Es por ese motivo que es necesario que se dote al sistema de una función bloqueante que determina el estado de la colas de espera y ejecución del Multicluster y no devuelva el control hasta que detecte que han finalizado todas las tareas de gestión.

Esta función es llamada a través de la interfaz que ofrece el MetaScheduler, *waitActiveJobs()*. A continuación se comprueba el estado de las colas de espera y ejecución tanto a nivel de MetaScheduler como a nivel de planificadores locales LoRaS. Para hacer esto se recorre el conjunto de colas del sistema de colas (Input Queue)

y se comprueba la cantidad de aplicaciones en espera (*getJobCount()*). Luego se comprueba la cantidad de aplicaciones que todavía se encuentran en ejecución (*getRunningCount()*). Esta consulta se realiza de forma bloqueante. De este modo no se devolverá el control al sistema hasta que no existan aplicaciones en ninguna de las colas.

Esta función deja de tener sentido en un modelo de aplicación Cliente-Servidor, donde MetaLoRaS se configura como un servidor de peticiones. Este servidor atiende peticiones de ejecución y información de los usuarios paralelos y permanece siempre activo aún cuando no hay aplicaciones para ejecutar, hasta que se para el servicio. Este modelo de aplicación es el que se desea dotar al sistema M-CISNE en un trabajo futuro.

Apéndice B

Diseño del Repositorio de datos

En este capítulo se describe el modelo de datos que utiliza el repositorio de datos para almacenar la información sobre la ejecución de las aplicaciones paralelas y el estado del sistema durante su ejecución. En la sección B.1 se muestra el modelo de datos relacional que se ha diseñado para el repositorio de datos históricos. En la sección B.2 se describen de forma detalla la entidades que aparecen en el modelo relacional y su funcionalidad.

B.1. Modelo de datos

En la Figura B.1 se puede ver el diagrama Entidad-Relación que define el modelo de datos implementado en el repositorio de datos utilizado por M-CISNE en el presente trabajo.

B.2. Entidades

En esta sección se hace un descripción del modelo de datos diseñado para el repositorio de datos históricos utilizado por M-CISNE. Para cada entidad se muestra una descripción de su funcionalidad, los atributos definidos y como la entidad se relaciona con el resto de entidades del repositorio de datos.

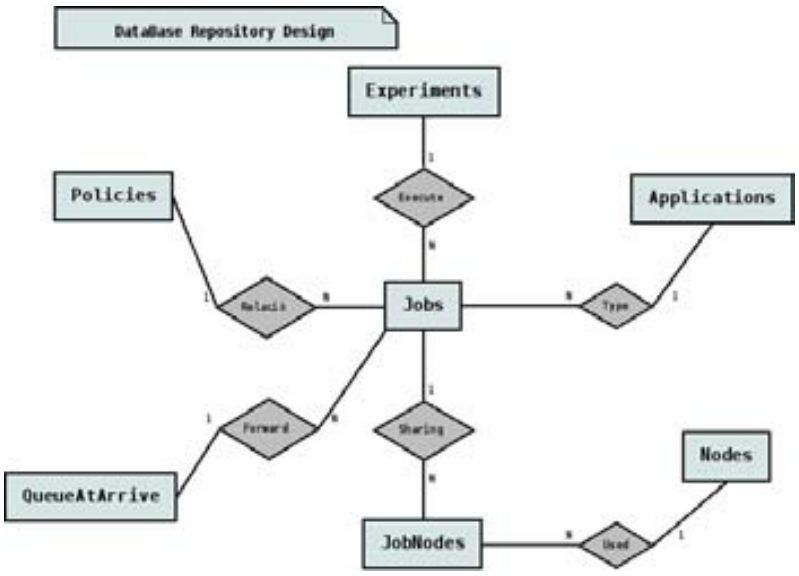


Figura B.1: Modelo de datos relacional para el repositorio de datos históricos

B.2.1. Experiments

Mantiene un registro de los experimentos que se han realizado. Se almacena principalmente el momento en que se realiza cada experimento (*Time*), el conjunto de clases que lo configuran y el estado final del Experimento (*State*) válido o no válido.

Nombre entidad: Experiments					
Clave principal		EID			
Nombre atributo	**Tipo de atributo**	**Longitud**	**Rango**	**Entidad Relacionada**	**Observaciones**
EID	Integer	10			Clave principal
Time	Time				
ConfFile	Varchar	30			
WTClass	Varchar	30			
EUJClass	Varchar	30			
ERTClass	Varchar	30			
State	Varchar	15			

B.2.2. Applications

Mantiene un registro de las aplicaciones ejecutadas en el sistema y la información básica que las define.

Nombre entidad: Experiments					
Clave principal	AID				
Nombre atributo	**Tipo de atributo**	**Longitud**	**Rango**	**Entidad Relacionada**	**Observaciones**
AID	Integer	10			Clave principal
Name	Varchar	30			
Dir	Varchar	255			
Params	Varchar	100			
Type	TinyInt	3			
Memory	Integer	10			
Processors	Integer	5			
MaxTime	Integer	10			
Jiffies	Integer	10			
Ppbw	Float		4,3		
CpuRatio	Float		4,3		
BoundedBy	Integer	5			

B.2.3. Policies

Mantiene un registro de las políticas de planificación utilizadas.

Nombre entidad: Policies					
Clave principal	PID				
Nombre atributo	**Tipo de atributo**	**Longitud**	**Rango**	**Entidad Relacionada**	**Observaciones**
PID	Integer	10			Clave principal
Description	Varchar	255			
Name	Varchar	50			

B.2.4. Nodes

Mantiene un registro de los nodos que forman parte de un cluster. La información que se almacena hace referencia a las características y las capacidades de los recursos tanto de cómputo como de comunicación. *PWeidth* es la potencia efectiva del nodo, *MaxBW* es el ancho de banda máximo del canal de comunicación del nodo y *ChannelType* especifica el tipo de comunicación dedicada o compartida.

Nombre entidad: Nodes					
Clave principal		NID			
Nombre atributo	**Tipo de atributo**	**Longitud**	**Rango**	**Entidad Relacionada**	**Observaciones**
NID	Integer	10			Clave principal
Name	Varchar	20			
Memory	Integer	10			
PWeidth	Float		6,4		
MaxBW	Float		6,4		
ChannelTYPE	Integer	4			

B.2.5. Jobs

Es la entidad central de nuestro modelo de datos. Mantiene un registro de experiencias previas a la ejecución de las aplicaciones paralelas. La información que se almacena es básicamente un control de la ejecución formado por los tiempos de llegada (ArriveTime), ejecución (StartTime) y salida (FinishTime) del sistema de la aplicación paralela. Los requisitos de CPU y comunicación de las aplicaciones paralelas. La cantidad de Jiffies que consume la aplicación y los jiffies por delante en la cola de espera a su llegada (*AheadJiffies*). Los tiempos que tarda el motor de predicción en devolver las estimaciones mediante los métodos estadísticos (TimetoHis) y los analíticos (TimetoSim). Si la aplicación no se ejecutó correctamente también se deja constancia del tipo de error que sucedió.

Nombre entidad: Jobs					
Clave principal		HJID			
Nombre atributo	**Tipo de atributo**	**Longitud**	**Rango**	**Entidad Relacionada**	**Observaciones**
HJID	Integer	10			Clave principal
EID	Integer	10		Experiments	Clave foránea
PID	Integer	10		Policies	Clave foránea
AID	Integer	10		Applications	Clave foránea
JID	Integer	5			
ArriveTime	Integer	10			
StartTime	Integer	10			
FinishiTime	Integer	10			
CpuLoad	float		4,5		
NetLoad	float		6.3		
SimArriveTime	Integer	10			
SimStartTime	Integer	10			
SimFinishTime	Integer	10			
UsedJiffies	Integer	10			
AheadJiffies	Integer	10			
BackwardJiffies	Integer	10			
TimetoHis	Float		6,5		
TimetoSim	Float		6,5		
Error	Integer				

B.2.6. JobNodes

Mantienen un registro de las aplicaciones que comparten nodos durante su ejecución. La información que se almacena es el identificador de las dos aplicaciones que comparten el nodo, el identificador del nodo que comparten y los tiempos de inicio y fin de la compartición. Esta es la tabla con mayor número de registros de la base de datos.

Nombre entidad: JobNodes					
Clave principal		HJID1, HJID2, NID			
Nombre atributo	**Tipo de atributo**	**Longitud**	**Rango**	**Entidad Relacionada**	**Observaciones**
HJID1	Integer	10		Jobs	Clave foránea y principal
HJID2	Integer	10		Jobs	Clave foránea y principal
NID	Integer	10		Nodes	Clave principal
StartTime	Integer	10			
FinishTime	Integer	5			

B.2.7. QueueAtArrive

Mantiene un registro de las aplicaciones que han coincidido en la cola de espera. A la llegada de una aplicación al sistema se registra el conjunto de aplicaciones que se encuentran en la cola de espera y en qué posición. Esto es útil cuando el motor de predicción utiliza información sobre el número de trabajo en la cola para realizar estimaciones.

Nombre entidad: QueueAtArrive					
Clave principal		HJID1, HJID2			
Nombre atributo	**Tipo de atributo**	**Longitud**	**Rango**	**Entidad Relacionada**	**Observaciones**
HJID1	Integer	10		Jobs	Clave foránea y principal
HJID2	Integer	10		Jobs	Clave foránea y principal
Position	Integer	10			

Apéndice C

Métodos estadísticos

En el presente trabajo se han aplicado métodos estadísticos muy diversos para resolver distintas cuestiones especialmente en la tarea de predicción. Algunas de las cuestiones que se han resuelto mediante estos métodos han sido, medida de la distancia entre experiencias en la predicción estadística mediante un modelo *IBL*, factor de corrección en la desviación de las predicciones, cálculo del error, comparación estadística entre los distintos métodos de predicción y la validación de los resultados obtenidos. En este apéndice describimos las bases de los métodos estadísticos aplicados durante el desarrollo del presente trabajo.

C.1. Métrica "Heterogenous Euclidean-Overlap"

Un modo de manejar problemas con atributos continuos y nominales es utilizando la función de distancia heterogénea que permite aplicar distintas funciones de distancia para distintos tipos de atributos.

Esta función define la distancia entre dos valores x e y de un atributo como:

$$d_a(x,y) = \begin{cases} 1 & if\ x\ or\ y\ is\ unknow, else \\ overlap(x,y) & if\ a\ is\ nominal, else \\ diff_a(x,y) & \end{cases}$$

A los valores de atributos desconocidos se les asigna la distancia 1. La función de superposición entre dos valores x e y, $d_a(x,y)$, y la diferencia normalizada $diff_a(x,y)$ se definen como:

$$d_a(x,y) = \begin{cases} 0 & if\ x = y \\ 1 & otherwise \end{cases}$$

$$diff_a(x,y) = \frac{|x-y|}{range_a}$$

La definición anterior de d_a devuelve un valor comprendido (normalmente) en el rango 0 .. 1. El valor $range_a$ es usado para normalizar los atributos y se define como:

$$range_a = max_a - min_a$$

donde max_a y min_a son los valores máximo y mínimo respectivamente, observados en la formación del conjunto de atributos a.

La distancia total entre los dos vectores de entrada x e y viene dado por la función "Heterogeneous Euclidean-Overlap Metric" *HEOM(x,y)*:

$$HEMO(x,y) = \sqrt{\sum_{a=1}^{m} d_a(x_a, y_a)^2}$$

Apéndice D

Entorno de Experimentación

D.1. Modelado de las cargas

Para poder llevar a cabo la experimentación del presente trabajo ha sido necesario modelar el conjunto de aplicaciones que van a ser ejecutados en el Multicluster. Teniendo en cuenta que el presente trabajo está situado en la meta-planificación en entornos Multicluster no dedicados, debemos modelar tanto cargas paralelas como locales.

D.1.1. Caracterización de la carga paralela

En la práctica se utilizan principalmente dos métodos para el modelado de cargas paralelas. El primero consiste en llevar a cabo un análisis de los ficheros de traza generados por grandes sistemas de computación mediante clusters. Muchos centros de investigación disponen de estos sistemas y algunos ofrecen este tipo de información a través de la red. La información que ofrecen este tipo de ficheros incluye normalmente los tiempos de llegada al sistema, el tamaño de los trabajos y los tiempos de ejecución. Algunos de estos ficheros pueden contener de centenares a miles de trabajos paralelos. Una de las principales desventajas del uso de este tipo de traza es que el tamaño de los trabajos está directamente relacionado con el tamaño del cluster que se usa. Además se pueden dar distintos factores [Fei] que pueden alterar la representatividad de las medidas de rendimiento, como por ejemplo: intervalos demasiado grandes entre la llegada de trabajos, tareas poco usuales que añaden ruido al comportamiento habitual del cluster, etc. Esto requiere un análisis cuidadoso de las trazas que se utilizan e incluso el filtraje y extracción de información de las mismas, técnicas que en la actualidad está siendo objeto de muchos estudios.

Una segunda alternativa es la generación de un modelo sintético de carga. En este tipo de modelos los parámetros principales (tiempo de llegada, tamaño de la carga y los tiempos de ejecución se generan mediante distribuciones aleatorias). Este método permite el uso de distribuciones como la Uniforme o Poison, o hasta incluso complejas distribuciones basadas en el comportamiento de una traza real. La ventaja principal del uso de este tipo de modelos es que nos permiten modelar con mayor exactitud el comportamiento del sistema, facilitando el análisis de los resultados y la identificación de los reajustes necesarios.

En el presente estudio, donde el objetivo principal es validar el funcionamiento de la herramienta diseñada y analizar el comportamiento de distintas políticas de planificación, necesitamos poder analizar los resultados y detectar posibles errores de la forma más rápida y sencilla posible. Para poder hacer esto necesitamos conocer bien la composición y la distribución en el tiempo de la carga que vamos a introducir en el sistema. Además, necesitamos controlar el tamaño de esta carga y conocer bien las aplicaciones paralelas que la van a formar.

Por todo esto, en el presente estudio hemos decidido utilizar un generador de carga sintética, que nos permita configurar la distribución de la carga y parametrizar su tamaño. Además, las cargas están formadas de aplicaciones paralelas muy conocidas y ampliamente utilizadas.

D.1.1.1. Generador de cargas paralelas

Para generar la carga que vamos a introducir en el sistema nos hemos basado en un generador de carga sintética basado en modelos estadísticos, definido por Feitelson et al. en [Fei96] y utilizado en numerosos estudios posteriores [FR98, LMW98, Aid00, Fei01, Fei03].

La función principal del generador de carga es generar, siguiendo una distribución aleatoria, una secuencia de registros donde cada registro proporciona información sobre las características de la aplicación que se debe ejecutar y el instante de llegada.

Concretamente, el generador de carga que hemos utilizado nos proporciona la información siguiente:

- Tiempo de llegada. Mediante una función de distribución de Poison se determina el momento en que las aplicaciones paralelas llegan al sistema.
- Cantidad de nodos. Cada petición lleva asociada la cantidad de nodos que requiere la aplicación.
- Tiempo de ejecución. Cada petición lleva asociado el tiempo de ejecución de la aplicación.

Una vez disponemos de la información sobre las características de las aplicaciones que se deben ejecutar y su distribución en el tiempo, ya podemos generar la carga paralela que vamos a introducir en el sistema. Para hacer esto, solo hace falta disponer de un conjunto de aplicaciones paralelas que podamos parametrizar para poder ajustarlas a las características proporcionadas por el generador de carga.

El fichero de carga paralela que se le entregará al sistema consiste en un conjunto de registros donde cada registro contiene el instante de llegada de la aplicación al sistema y el nombre de la aplicación paralela que más se ajusta a los requerimientos proporcionados por el generador de carga. En la tabla D.1 se muestra una lista de las aplicaciones paralelas y sus características.

D.1.1.2. Caracterización de las aplicaciones paralelas

Una vez disponemos del modelo de carga paralela que queremos introducir en nuestro sistema, necesitamos un conjunto de aplicaciones paralelas parametrizables que se ajusten a los requisitos especificados en el modelo.

Para ello, se han escogido un par de benchmarks de la NAS, el *IS* y el *MG*. Los motivos de esta elección son varios. En primer lugar, estos benchmarks son muy conocidos dentro del grupo de investigación y ampliamente utilizados en estudios anteriores. En segundo lugar, nos permiten configurar diferentes cargas de CPU, Memoria, Comunicaciones, y cantidad de nodos. Además, la naturaleza de ambos benchmarks es bien distinta, siendo *IS* una aplicación paralela orientada a comunicación, mientras que *MG* está orientada al cómputo. Esto permite mayor diversidad en el conjunto de pruebas y por tanto mayor representatividad en la experimentación.

Para seleccionar la aplicación paralela que mejor se ajusta a cada registro del generador en cuanto a número de nodos y tiempo de ejecución, necesitamos caracterizar las aplicaciones según sus posibles parámetros de entrada. Para poder realizar esta caracterización, ejecutamos en un cluster dedicado ambos benchmarks para distintos parámetros de configuración. La tabla D.1, muestra los valores que caracterizan las distintas aplicaciones paralelas que se han usado en este estudio.

Para que el sistema de planificación sea capaz de ejecutar las aplicaciones paralelas de la forma más eficiente, es necesario además conocer ciertos parámetros de las mismas que aporten datos a las políticas de planificación. Los parámetros que se han extraído durante el proceso de caracterización son los siguientes:

- Nodos requeridos. Especifica el número de nodos que requiere la aplicación para ser ejecutada. En nuestro caso tratamos con aplicaciones rígidas, de manera que la cantidad de nodos solicitados por una aplicación es fija. Aquellas

aplicaciones que pueden ser ejecutadas con diferente número de nodos (moldeables) serán consideradas como aplicaciones distintas.

- Tiempo de ejecución. Es el tiempo resultante de la ejecución de la aplicación paralela en un cluster dedicado con unos parámetros de entrada determinados.
- Recursos consumidos. Con el objetivo de realizar planificaciones eficientes asegurando un mínimo de recursos al usuario local debemos conocer los recursos que en media consumirán las aplicaciones paralelas. En este sentido se han implementado técnicas a nivel de planificador local (LoRaS) que permiten limitar el uso de los recursos de Memoria y CPU de las aplicaciones paralelas. Concretamente, en el caso de la Memoria se pretende evitar un nivel de paginación que influya negativamente en las tareas locales. En el caso de la CPU se pretende evitar la asignación de aplicaciones paralelas a un procesador que no permita la interactividad con el sistema.
- Naturaleza de la aplicación. En función de los recursos consumidos por una aplicación, se puede determinar si ésta se encuentra orientada hacia el cómputo o hacia la entrada/salida. Esta información es muy útil para algunas políticas de planificación a la hora de asignar nodos de un cluster a la aplicación paralela.
- Ejecutable y Parámetros: se define el nombre del ejecutable de la aplicación paralela y su ubicación. Además se incluyen los parámetros de entrada que la han caracterizado.

En este estudio se han desarrollado cargas paralelas de 30, 60 y 90 aplicaciones. El conjunto de aplicaciones utilizado es el que se muestra en la tabla D.1.

Según la naturaleza de las aplicaciones se han generado cargas orientadas al cómputo (mayor cantidad de aplicaciones orientadas al cómputo), a la comunicación (mayor cantidad de aplicaciones orientadas a la comunicación) y balanceadas.

Por último, destacar que se han creado un conjunto de aplicaciones paralelas para que sean consideradas por el sistema como aplicaciones erróneas. Esto permite crear cargas paralelas que representan con mayor fidelidad la realidad, dado que este tipo de fallos son muy comunes tal y como se ha descrito en [CB01, Fei02].

D.1.2. Caracterización de la carga local

Teniendo en cuenta que el sistema Multicluster donde realizamos nuestro estudio es un entorno no dedicado, necesitamos disponer de una caracterización de la carga

	NAS-IS (CPU(%), Mem. (MB), Tiempo (seg)								
	Clase A			Clase AB			Clase B		
Nodos	CPU	Mem	Tiempo	CPU	Mem	Tiempo	CPU	Mem	Tiempo
2	1015	150	19	2438	297	29	5326	593	69
4	695	73	19	1600	144	29	3216	285	69
8	602	37	19	1139	74	29	2455	152	49

	NAS-MG (CPU(%), Mem. (MB), Tiempo (seg)								
	CLASE A			CLASE AB			CLASE B		
Nodos	CPU	Mem	Tiempo	CPU	Mem	Tiempo	CPU	Mem	Tiempo
2	2133	227	29	6746	227	79	11939	227	130
4	984	117	19	3448	117	39	5961	117	69
8	120	62	9	1294	62	19	1640	62	39

Tabla D.1: Caracterización de las aplicaciones paralelas

Perfil	CPU (%)	Mem (%)	E/S a disco (bloques/s) Leídos - Escritos	Red (Bytes/s) Recibidos - Enviados
Xwin	0.15	35 %	67 - 17	608 - 30
Internet	0.2	60 %	99 - 52	3154 - 496
Shell	0.25	20 %	13 - 12	108 - 3

Tabla D.2: Descripción de los perfiles locales

local que nos permita introducir en el sistema algún tipo de actividad que represente al usuario local. Con ello se pretende analizar el comportamiento de las aplicaciones paralelas en función de la carga local y evaluar la influencia de las aplicaciones paralelas en el usuario local.

Para poder modelar la carga local nos hemos basado en un estudio realizado anteriormente dentro del grupo de investigación [Gin04], donde se han definido un conjunto de perfiles de usuarios locales en función de los recursos consumidos. Estos perfiles han sido generados a partir de la monitorización de los recursos consumidos por usuarios reales en laboratorios de informática de la Universitat de Lleida. La tabla D.2 muestra una descripción de los perfiles obtenidos y los recursos que éstos consumen.

En la figura D.2 se puede observar la distribución de los perfiles definidos durante la monitorización en un laboratorio abierto. Estos resultados nos muestran que un 62 % de los usuarios tiene un marcado perfil ofimático. Es decir, pasa la mayor parte del tiempo leyendo y editando documentos mediante el uso de distintas aplicaciones

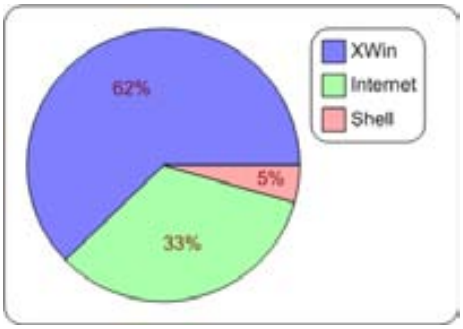


Figura D.1: Distribución de los perfiles de usuarios locales

bajo interfaces gráficas. En el caso del usuario de Internet podemos observar que el consumo de recursos aumenta, sobretodo en cuanto a red se refiere y supone un 33 % de los usuarios. Por último los usuarios *Shell* destacan únicamente por el consumo de recursos de CPU y representan un escaso 5 % de los usuarios. Del estudio anterior se destaca también que el número de ordenadores utilizados en promedio en un laboratorio abierto está entre el 25 % y el 50 %, valores que se tendrán en cuenta en el análisis de la experimentación.

Para modelar una actividad que represente al usuario local, se ha construido un benchmark parametrizable con los valores obtenidos para cada perfil en cuanto a la utilización de recursos de CPU, Memoria, y E/S. Durante el periodo de ejecución de este benchmark (valor también parametrizable), se realiza un conjunto de llamadas a sistema representativas de un usuario local, se mide la latencia de estas llamadas y se registra en un fichero de monitorización el promedio de la latencia obtenida. Los valores de latencia obtenidos en un sistema sin carga paralela son comparados con los valores obtenidos en el mismo sistema después de ejecutar la carga paralela. Los resultados de esta comparativa nos van a permitir evaluar el efecto que causan la aplicaciones paralelas sobre la interactividad de las tareas locales.

Bibliografía

[AAB+00] Y. Amir, B. Awerbuch, A. Barak, R. S. Borgstrom, and A. Keren. An opportunity cost approach for job assignment in a scalable computing cluster. *IEEE Transactions on Parallel and Distributed Systems*, 11(7):760–768, 2000.

[Aba04] J.H. Abawajy. Dynamic parallel job scheduling in multi-cluster computing systems. In *ICCS'04: Proceedings of the International Conference on Computational Science*, pages 27–34, Kraków, Poland, June 2004.

[ABG02] David Abramson, Rajkumar Buyya, and Jonathan Giddy. A computational economy for grid computing and its implementation in the nimrod-g resource broker. *Future Generation Computer Systems*, 18(8):1061–1074, 2002.

[AD00] J.H. Abawajy and S.P. Dandamudi. Distributed Hierarchical Workstation Cluster Co-Ordination Scheme. In *PARELEC'00: Proceedings of the International Conference on Parallel Computing in Electrical Engineering*, volume 00, page 111, Quebec, Canada, Aug. 2000. IEEE Computer Society.

[AD03] J. H. Abawajy and S. P. Dandamudi. Parallel job scheduling on multicluster computing systems. *In (CLUSTER'03): Proceedings of IEEE International Conference on Cluster Computing*, 00:11–18, 2003.

[ADV+95] Remzi H. Arpaci, Andrea C. Dusseau, Amin M. Vahdat, Lok T. Liu, Thomas E. Anderson, and David A. Patterson. The interaction of parallel and sequential workloads on a network of workstations. In *SIGMETRICS'95: Proceedings of the 1995 ACM SIGMETRICS International conference on Measurement and modeling of computer systems*, pages 267–278, Ottawa, Ontario, Canada, 1995.

[AFKT00] Olaf Arndt, Bernd Freisleben, Thilo Kielmann, and Frank Thilo. A comparative study of online scheduling algorithms for networks of workstations. *Cluster Computing*, 3(2):95–112, 2000.

[Aid00] Kento Aida. Effect of job size characteristics on job scheduling performance. In *JSSPP'00: Proceedings of the 6th Workshop on Job Scheduling Strategies for Parallel Processing*, pages 1–17, Canun, Mexico, May 2000.

[AKN98] Kento Aida, Hironori Kasahara, and Seinosuke Narita. Job scheduling scheme for pure space sharing among rigid jobs. In *JSSPP'98: Proceedings of the 4th Workshop on Job Scheduling Strategies for Parallel Processing*, pages 98–121, Orlando, FL, USA, March 1998. Springer-Verlag.

[AMS97] Christopher G. Atkeson, Andrew W. Moore, and Stefan Schaal. Locally weighted learning for control. *Artificial Intelligence Review*, 11(1-5):75–113, 1997.

[AS97] Stergios V. Anastasiadis and Kenneth C. Sevcik. Parallel application scheduling on networks of workstations. *Journal of Parallel and Distributed Computing*, 43:109–124, 1997.

[AS99] A. Acharya and S. Setia. Availability and utility of idle memory in workstation clusters. In *SIGMETRICS'99: Proceedings of the ACM SIGMETRICS Conference on the Measurement and Modeling of Computer Systems*, pages 35–46, Atlanta, Georgia, USA, May 1999.

[BAG00] R. Buyya, D. Abramson, and J. Giddy. Nimrod/g: An architecture for a resource management and scheduling system in a global computational grid. In *HPC'00: Proceedings of the 4th International Conference on High Performance Computing in Asia-Pacific Region*, Beijing, China, May 2000.

[BB01] R. Buyya and M. Baker. Emerging technologies for multicluster/grid computing. In *CLUSTER '01: Proceedings of the 3rd IEEE International Conference on Cluster Computing*, page 457, Saxony, Germany, Oct. 2001. IEEE Computer Society.

[BBC+04] Cyril Banino, Olivier Beaumont, Larry Carter, Jeanne Ferrante, Arnaud Legrand, and Yves Robert. Scheduling strategies for master-slave tasking on heterogeneous processor platforms. *IEEE Transactions on Parallel and Distributed Systems*, 15(4):319–330, 2004.

[BBE03] S. Banen, A. I. D. Bucur, and D. H. J. Epema. A measurement-based simulation study of processor co-allocation in multicluster systems. In *JSSPP'03: Proceedings of the 9th Workshop on Job Scheduling Strategies for Parallel Processing*, pages 105–128, Seattle, WA, USA, June 2003. Springer-Verlag.

[BDBS92] David H. Bailey, Leonardo Dagum, E. Barszcz, and Horst D. Simon. NAS parallel benchmark results. In *SC1992: Proceedings of the 1992 ACM/IEEE conference on Supercomputing*, pages 386–393, Minneapolis, MN, USA, Nov. 1992.

[BE00] A. I. D. Bucur and D. H. J. Epema. The influence of the structure and sizes of jobs on the performance of co-allocation. In *JSSPP'00: Proceedings of the 6th Workshop on Job Scheduling Strategies for Parallel Processing*, pages 154–173, Cancun, Mexico, May 2000. Springer-Verlag.

[BE01] Anca I. D. Bucur and Dick H. J. Epema. The influence of communication on the performance of co-allocation. In *JSSPP'01: Revised Papers from the 7th International Workshop on Job Scheduling Strategies for Parallel Processing*, pages 66–86, Cambridge, MA, USA, June 2001. Springer-Verlag.

[BE02] Anca I. D. Bucur and Dick H. J. Epema. Local versus global schedulers with processor co-allocation in multicluster systems. In *JSSPP '02: Revised Papers from the 8th International Workshop on Job Scheduling Strategies for Parallel Processing*, pages 184–204, Edinburgh, Scotland, UK, July 2002. Springer-Verlag.

[BE03a] A. I. D. Bucur and D. H. J. Epema. The maximal utilization of processor co-allocation in multicluster systems. In *JSSPP'03: Proceedings of the 17th International Symposium on Parallel and Distributed Processing*, page 60, Seattle, WA, USA, June 2003. IEEE Computer Society.

[BE03b] A. I. D. Bucur and D. H. J. Epema. The performance of processor co-allocation in multicluster systems. In *CCGRID '03: Proceedings of the 3st International Symposium on Cluster Computing and the Grid*, page 302, Tokyo, Japan, May 2003. IEEE Computer Society.

[BSA05] A. Barak, A. Shiloh, and L. Amar. An organizational grid of federated MOSIX clusters. In *CCGrid'05: Proceedings of the Fifth IEEE Inter-*

national Symposium on Cluster Computing and the Grid, volume 1, pages 350–357, Cardiff, UK, May 2005. IEEE Computer Society.

[CB01] W. Cirne and F. Berman. A comprehensive model of the supercomputer workload. In *WWC '01: IEEE International Workshop on Proceedings of the Workload Characterization*, pages 140–148, Austin, Texas, USA, December 2001. IEEE Computer Society.

[CC00] Brent N. Chun and David E. Culler. Market-based proportional resource sharing for clusters. Technical report, 2000.

[CC06] Y. Cardinale and H. Casanova. An evaluation of job scheduling strategies for divisible loads on grid platforms. High Performance Computing & Simulation (HPC&S) Conference. Libro: "IEEE". Vol. 1. pp. 705 - 712., May 2006.

[CFGK95] Nicholas Carriero, Eric Freeman, David Gelernter, and David Kaminsky. Adaptive parallelism and piranha. *Computer*, 28(1):40–49, 1995.

[CFK+98] Karl Czajkowski, Ian Foster, Carl Kesselman, Stuart Martin, Warren Smith, and Steven Tuecke. A resource management architecture for metacomputing systems. In *JSSPP'98: Proceedings of the 4th Workshop on Job Scheduling Strategies for Parallel Processing*, pages 62–82, Orlando, FL, USA, March 1998. Springer-Verlag.

[Con] Condor project. http://www.cs.wisc.edu/condor.

[DA06] F. Dong and S. G. Akl. Scheduling algorithms for grid computing: State of the art and open problems. In *Technical Report 2006-504. School of Computing, Queen's University, Kingston, Ontario, Canada*, 2006.

[DI89] M.V. Devarakonda and R.K. Iyer. Predictability of process resource usage: a measurement-based study on unix. *IEEE Transactions on Software Engineering*, 15(12):1579–1586, Dec 1989.

[Din99] P. Dinda. The statistical properties of host load. *Scientific Programming*, 7(3-4):211–229, 1999.

[Din01] P.A. Dinda. Online prediction of the running time of tasks. In *HPDC'01: Proceedings of the Tenth IEEE International Symposium on High-Performance Distributed Computing*, pages 469 – 47, Redondo Beach, CA, USA, Aug. 2001.

[DMS05] P. Domingues, P. Marques, and L. Silva. Resource usage of windows computer laboratories. In *ICPP'05: Proceedings of the 2005 International Conference on Parallel Processing Workshops*, pages 469–476, Oslo, Norway, June 2005.

[Dow97] A. Downey. Predicting queue times on space-sharing parallel computers. In *IPPS'97: Proceedings of the 11th International Symposium on Parallel Processing*, pages 209–218, Geneva, Switzerland, April 1997.

[DZ97] Xing Du and Xiaodong Zhang. Coordinating parallel processes on networks of workstations. *Journal of Parallel and Distributed Computing*, 46(2):125–135, 1997.

[EHS+02] Carsten Ernemann, Volker Hamscher, Uwe Schwiegelshohn, Ramin Yahyapour, and Achim Streit. On advantages of grid computing for parallel job scheduling. In *CCGRID '02: Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid*, page 39, Berlin, Germany, May 2002.

[EHYS02] Carsten Ernemann, Volker Hamscher, Ramin Yahyapour, and Achim Streit. Enhanced algorithms for multi-site scheduling. In *GRID 2002: Proceedings of Third International Workshop in Grid Computing*, pages 219–231, Baltimore, MD, USA, Nov. 2002. Springer-Verlag.

[ELvD+96] D. H. J. Epema, M. Livny, R. van Dantzig, X. Evers, and J. Pruyne. A worldwide flock of condors: load sharing among workstation clusters. *Future Generation Computer Systems*, 12(1):53–65, 1996.

[FBCL91] Michael J. Feeley, Brian N. Bershad, Jeffrey S. Chase, and Henry M. Levy. Dynamic node reconfiguration in a parallel-distributed environment. In *SIGPLAN'91: Proceedings of the 1991 ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 114–121, Toronto, Ontario, Canada, June 1991.

[Fei] Dr. Feitelson. Job scheduling strategies for parallel processing. http://www.cs.huji.ac.il/ feit/parsched/.

[Fei96] D. G. Feitelson. Packing schemes for gang scheduling. In *IPPS '96: Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing*, volume 1162, pages 89–110, Honolulu, Hawaii, USA, April 1996. Springer-Verlag.

[Fei97] Dror G. Feitelson. Job scheduling in multiprogrammed parallel systems. IBM Research Report RC 19790 (87657), Aug. 1997.

[Fei01] Dror G. Feitelson. Metrics for parallel job scheduling and their convergence. In *JSSPP'01: Revised Papers from the 7th International Workshop on Job Scheduling Strategies for Parallel Processing*, volume 2221, pages 188–206, Cambridge, MA, USA, June 2001. Springer-Verlag.

[Fei02] Dror G. Feitelson. Workload modeling for performance evaluation. In *Performance Evaluation of Complex Systems: Techniques and Tools, Performance 2002, Tutorial Lectures*, pages 114–141, London, UK, 2002. Springer-Verlag.

[Fei03] Dror G. Feitelson. Metric and workload effects on computer systems evaluation. *Computer*, 36(9):18–25, 2003.

[FK97] I. Foster and C. Kesselman. Globus: A Metacomputing Infrastructure Toolkit. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(2):115–128, Summer 1997.

[FK98] Ian Foster and Nicholas T. Karonis. A Grid-Enabled MPI: Message passing in heterogeneous distributed computing systems. In *SC1998: Proceedings of the 1998 ACM/IEEE conference on Supercomputing*, pages 1–11, San Jose, CA, Nov. 1998. ACM Press.

[FR90] D.G. Feitelson and L. Rudolph. Distributed hierarchical control for parallel processing. *Computer*, 23(5):65–77, May 1990.

[FR98] Dror G. Feitelson and Larry Rudolph. Metrics and benchmarking for parallel job scheduling. In *JSSPP'98: Proceedings of the 5th Workshop on Job Scheduling Strategies for Parallel Processing*, pages 1–24, Orlando, FL, USA, March 1998. Springer-Verlag.

[Gib97] Richard Gibbons. A historical application profiler for use by parallel schedulers. In *JSSPP'97: Proceedings of the 3th Job Scheduling Strategies for Parallel Processing*, pages 58–77, 1997.

[Gin04] F. Giné. *Cooperating Coscheduling: a coscheduling proposal for non-dedicated multiprogrammed clusters*. PhD thesis, Universitat Autonoma de Barcelona, 2004.

[GK92] David Gelernter and David Kaminsky. Supercomputing out of Recycled Garbage: Preliminary Experience with Piranha. In *SC1992: Proceedings of the Sixth ACM International Conference on Supercomputing*, pages 417–427, Minneapolis, Minnesota, USA, Nov. 1992. ACM Press.

[GLD04] A. Gupta, B. Lin, and P. Dinda. Measuring and understanding user comfort with resource borrowing. In *HPDC'04: Proceedings of the 13th IEEE International Symposium on High Performance Distributed Computing*, pages 214–224, Honolulu, Hawaii USA, June 2004. IEEE Computer Society.

[Glo] Globus. Community scheduler framework. http://www.platform.com/products/globus/.

[GSB+03] Francesc Giné, Francesc Solsona, Jesus Barrientos, Porfidio Hernández, Mauricio Hanzich, and Emilio Luque. Multiprogramming Level of PVM jobs in a non-dedicated linux now. In *PVM/MPI'03: Proceedings of the 10th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface*, pages 577–585, Venice, Italy, Sep. 2003.

[GSHL03] F. Giné, F. Solsona, P. Hernández, and E. Luque. Cooperating coscheduling in a non-dedicated cluster. In *EUROPAR'03: Proceeding of International Conference on Parallel and Distributed Computing*, pages 212–218, Klagenfurt, Austria, Aug. 2003.

[HBLO+03] M. Harchol-Balter, C. Li, T. Osogami, A. Scheller-Wolf, and M. S. Squillante. Cycle stealing under immediate dispatch task assignment. In *SPAA'03: Proceedings of the fifteenth annual ACM symposium on Parallel algorithms and architectures*, pages 274–285, San Diego, California, USA, June 2003.

[HGH+04] Mauricio Hanzich, Francesc Giné, Porfidio Hernández, Francesc Solsona, and Emilio Luque. Coscheduling and multiprogramming level in a non-dedicated cluster. In *PVM/MPI'04: Proceedings of the 12th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface*, pages 327–336, Budapest, Hungary, Sep. 2004.

[HGH+05a] M. Hanzich, F. Giné, P. Hernández, F. Solsona, and E. Luque. Cisne: A new integral approach for scheduling parallel applications on non-dedicated clusters. In *EUROPAR'05: Proceeding of International*

Conference on Parallel and Distributed Computing, volume 3648, pages 220–230, Lisboa, Portugals, Sep. 2005.

[HGH+05b] Mauricio Hanzich, Francesc Giné, Porfidio Hernández, Francesc Solsona, and Emilio Luque. A space and time sharing scheduling approach for pvm non-dedicated clusters. In *PVM/MPI'05: Proceedings of the 12th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface*, pages 379–387, Sorrento, Italy, Sep. 2005.

[HGH+06] M. Hanzich, F. Giné, P. Hernández, F. Solsona, and E. Luque. Using on-the-fly simulation for estimating the turnaround time on non-dedicated clusters. In *EUROPAR'06: Proceeding of International Conference on Parallel and Distributed Computing*, pages 117–187, Dresden, Germany, Aug. 2006.

[HHL+06] M. Hanzich, P. Hernández, E. Luque, F. Giné, F. Solsona, and J.L. Lérida. Using simulation, historical and hybrid estimation systems for enhacing job scheduling on nows. In *CLUSTER'06: Proceedings of 2006 IEEE International Conference on Cluster Computing*, pages 1–12, Sep. 2006.

[HJSN04] L. He, S.A. Jarvis, D.P. Spooner, and G.R. Nudd. Optimising static workload allocation in multiclusters. In *IPDPS'04: Proceedings of 18th International Parallel and Distributed Processing Symposium*, pages 39–46, April 2004.

[HSSY00] Volker Hamscher, Uwe Schwiegelshohn, Achim Streit, and Ramin Yahyapour. Evaluation of job-scheduling strategies for grid computing. In *GRID'00: Proceedings of the First IEEE/ACM International Workshop on Grid Computing*, pages 191–202, Bangalore, India, Dec. 2000. Springer-Verlag.

[IEEa] IEEE computer society task force on cluster computing. http://www.ieeetfcc.org/.

[IEEb] IEEE international symposium on cluster computing and the grid. http://www.ccgrid.org/.

[IOP99] M.A. Iverson, F. Ozguner, and L. Potter. Statistical prediction of task execution times through analytic benchmarking for scheduling in a heterogeneous environment. *IEEE Transactions on Computers*, 48(12):1374–1379, Dec 1999.

[JA05] B. Javadi and J. Abawajy. Performance analysis of heterogeneous multi-cluster systems. In *ICPPW'05: Proceedings of the 2005 International Conference on Parallel Processing Workshops*, pages 493–500, Oslo, Norway, June 2005.

[JAA06] Bahman Javadi, Mohammad K. Akbari, and Jemal H. Abawajy. A performance model for analysis of heterogeneous multi-cluster systems. *Parallel Computing*, 32(11-12):831–851, 2006.

[JLPS05] W. Jones, W. Ligon, L. Pang, and D. Stanzione. Characterization of bandwidth-aware meta-schedulers for co-allocating jobs across multiple clusters. *The Journal of Supercomputing*, V34(2):135–163, Nov. 2005.

[JSC01] David Jackson, Quinn Snell, and Mark Clement. Core algorithms of the maui scheduler. In *JSSPP'01: Proceedings of the 7th International Workshop on Job Scheduling Strategies for Parallel Processing*, pages 87–102, 2001.

[JSK+06] S. Jarvis, D. Spooner, H. Lim Choi Keung, J. Cao, S. Saini, and G. Nudd. Performance prediction and its use in parallel and distributed computing systems. *Future Generation of Computer Systems*, 22(7):745–754, 2006.

[KC91] P. Krueger and R. Chawla. The stealth distributed scheduler. In *ICDCS'91: Proceedings of the 11th International Conference on Distributed Computing Systems*, pages 336–343, May 1991.

[KFB99] Nirav H. Kapadia, Jose A. B. Fortes, and Carla E. Brodley. Predictive application-performance modeling in a computational grid environment. In *HPDC'99: Proceedings of The Eighth International Symposium on High Performance Distributed Computing*, pages 47–54, Aug. 1999.

[Kle75] Leonard Kleinrock. *Queueing Systems. Volume 1: Theory*. Wiley-Interscience; 1 edition (January 2, 1975), June 1975.

[KNOW08] Krzysztof Kurowski, Jarek Nabrzyski, Ariel Oleksiak, and Jan Węglarz. A multicriteria approach to two-level hierarchy scheduling in grids. *Journal of Scheduling*, 11(5):371–379, 2008.

[KNP01] Krzysztof Kurowski, Jarek Nabrzyski, and Juliusz Pukacki. User preference driven multiobjective resource management in grid environments. In *CCGRID '01: Proceedings of the 1st International*

Symposium on Cluster Computing and the Grid, page 114, Brisbane, Australia, May 2001. IEEE Computer Society.

[LGTW04] H. Li, D. Groep, J. Templon, and Lex Wolters. Predicting job start times on clusters. In *CCGrid'04: Proceedings of the 4th IEEE ACM International Symposium on Cluster Computing and the Grid*, Chicago, Illinois, USA, April 2004. IEEE Computer Society.

[Lif95] David A. Lifka. The ANL/IBM SP scheduling system. In *JSSPP'95: Proceedings of the 1st Workshop on Job Scheduling Strategies for Parallel Processing*, pages 295–303, Santa Barbara, CA, USA, April 1995. Springer-Verlag.

[LLM88] Michael Litzkow, Miron Livny, and Matthew Mutka. Condor - a hunter of idle workstations. In *Proceedings of the 8th International Conference of Distributed Computing Systems*, pages 104–111, June 1988.

[LLWN94] Wanqian Liu, V. Lo, K. Windisch, and B. Nitzberg. Non-contiguous processor allocation algorithms for distributed memory multicomputers. In *SC1994: Proceedings of the 1994 ACM/IEEE conference on Supercomputing*, pages 227–236, Washington, D.C., USA, Nov. 1994.

[LMW98] Virginia Lo, Jens Mache, and Kurt Windisch. A comparative study of real workload traces and synthetic workload models for parallel job scheduling. In *JSSPP'98: Proceedings of the 4th Workshop on Job Scheduling Strategies for Parallel Processing*, pages 25–46, Orlando, FL, USA, March 1998. Springer-Verlag.

[Loa] LoadLeveler. http://publib.boulder.ibm.com/infocenter/clresctr/vxrx/index.jsp?topic=/com.ibm.cluster.loadl.doc/llbooks.html.

[LS05] B. Lafreniere and A. Sodan. Scopred-scalable user-directed performance prediction using complexity modeling and historical data. In *JSSPP'05: Proceedings of the 11th Workshop on Job Scheduling Strategies for Parallel Processing*, pages 62–90, Cambridge, MA, USA, June 2005.

[LSF] LSF platform computing. http://www.platform.com/products/lsf/.

[LSG+06] J. Ll Lérida, Francesc Solsona, Francesc Giné, Mauricio Hanzich, Porfidio Hernández, and Emilio Luque. Metaloras: A predictable metascheduler for non-dedicated multiclusters. In *ISPA'06: Proceedings*

of the 4th International Symposium on Parallel and Distributed Processing with Applications, pages 630–641, Sorrento, Italy, Dec. 2006.

[LSG+08] Josep Ll. Lérida, F. Solsona, F. Giné, J. R. García, M. Hanzich, and P. Hernández. Enhancing prediction on non-dedicated clusters. In *Euro-Par'08: Proceedings of the 14th International Conference on Parallel and Distributed Computing*, pages 233–242, Las Palmas, Spain, Aug. 2008. Springer-Verlag.

[Mau] Maui scheduler. http://www.supercluster.org/maui.

[MF01] A. W. Mu'alem and D. G. Feitelson. Utilization, predictability, workloads, and user runtime estimates in scheduling the ibm sp2 with backfilling. *IEEE Transaction on Parallel and Distributed Systems*, 12(6):529–543, 2001.

[Moa] Moab, meta-scheduler (silver). http://www.supercluster.org/silver.

[MSST08] Verdi March, Alok Ranjan Singh, Simon See, and Yong Meng Teo. An overview to performance prediction of computer systems. Technical report, Asia Pacific Science and Technology Center Sun Microsystems Inc., 2008.

[MWW04] Marek Mika, Grzegorz Waligora, and Jan Weglarz. *Grid Resource Management - State of the Art and Future Trends*, chapter 19. A MetaHeuristic approach to Scheduling Workflow Jobs on a GRID. Kluwer Academic Publishers, 2004.

[NLYW05] V.K. Naik, C. Liu, L. Yang, and J. Wagner. Online resource matching for heterogeneous grid environments. *CCGRID*, 2:607–614, 2005.

[Oma77] Kenneth J. Omahen. Capacity bounds for multiresource queues. *Journal of the ACM (JACM)*, 24(4):646–663, 1977.

[Ope] PBS: The portable batch system. http://www.openpbs.org/.

[Ous82] John K. Ousterhout. Scheduling techniques for concurrent systems. In *ICDCS'82: Proceedings of Third International Conference on Distributed Computing Systems*, pages 22–30, Lauderdale, FL, USA, Oct. 1982.

[PCS07] Ji Su Park, Kwang-Sik Chung, and Jin Gon Shon. A design of cooperation management system to improve reliability in resource sharing computing environment. In *GPC'07: Proceedings of International*

Conference on Grid and Pervasive Computing, pages 496–506, Paris, France, May 2007.

[PL95] Jim Pruyne and Miron Livny. Parallel processing on dynamic resources with CARMI. In *JSSPP'95: Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing*, pages 259–278, Santa Barbara, CA , USA, April 1995. Springer-Verlag.

[PL96] Jim Pruyne and Miron Livny. Interfacing condor and pvm to harness the cycles of workstation clusters. *Future Generation of Computer Systems*, 12:67–85, 1996.

[RBMS97] D. Ridge, D. Becker, P. Merkey, and T. Sterling. Beowulf: Harnessing the power of parallelism in a pile-of-pcs. In *Proceedings of IEEE Aerospace*, pages 79–91, Feb. 1997.

[RH00] K. D. Ryu and J. K. Hollingsworth. Exploiting fine-grained idle periods in networks of workstations. *IEEE Transactions on Parallel and Distributed Systems*, 11(7):683–698, 2000.

[RI01] Matei Ripeanu and Adriana Iamnitchi. Cactus application: Performance predictions in a grid environment. In *EUROPAR'01: Proceedings of International Conference on Parallel and Distributd Computing*, pages 807–816, Manchester, UK, Aug. 2001.

[RL04] Alain Roy and Miron Livny. *Condor and preemptive resume scheduling*, chapter 9, pages 135–144. Grid resource management: state of the art and future trends. Kluwer Academic Publishers, Norwell, MA, USA, 2004.

[SAL+04] Jahanzeb Sherwani, Nosheen Ali, Nausheen Lotia, Zahra Hayat, and Rajkumar Buyya. Libra: a computational economy-based job scheduling system for clusters. *Software-Practice & Experience*, 34(6):573–590, 2004.

[SB98] J.M. Schopf and F. Berman. Performance prediction in production environments. In *IPPS/SPDP 1998: Proceedings of the First Merged International and Symposium on Parallel and Distributed Processing 1998*, pages 647–653, Orlando, FL, USA, Apr. 1998.

[SCBG03] Daniel Paranhos Da Silva, Walfredo Cirne, Francisco Vilar Brasileiro, and Campina Grande. Trading cycles for information: Using replication to schedule bag-of-tasks applications on computational grids.

In *Euro-Par'03: Proceedings of the 14th International Conference on Parallel and Distributed Computing*, pages 169–180, Klagenfurt, Austria, Aug. 2003.

[SCJG00] Q. Snell, M. Clement, D. Jackson, and C. Gregory. The performance impact of advance reservation meta-scheduling. In *JSSPP'00: Proceedings of the 6th International Workshop on Job Scheduling Strategies for Parallel Processing*, pages 137–153, Cancun, Mexico, May 2000.

[SFT98] W. Smith, I. Foster, and V. Taylor. Predicting application run times with historical information. *Journal of Parallel and Distributed Computing*, 64:1007–1016, 1998.

[SHb00] Bianca Schroeder and Mor Harchol-balter. Evaluation of task assignment policies for supercomputing servers: The case for load unbalancing and fairness. In *HPDC'00: Proceedings of the Ninth International Symposium on High-Performance Distributed Computing*, pages 211–219, Pittsburgh, Pennsylvania, Aug. 2000.

[SKSJ02] V. Subramani, R. Kettimuthu, S. Srinivasan, and J. Johnson. Selective buddy allocation for scheduling parallel jobs on clusters. In *CLUSTER'02: Proceedings of the 4th IEEE International Conference on Cluster Computing*, pages 107–116, Chicago, IL, USA, Sep. 2002.

[SKSS02] V. Subramani, R. Kettimuthu, S. Srinivasan, and S. Sadayappan. Distributed job scheduling on computational grids using multiple simultaneous requests. In *HPDC'02: Proceedings of 11th IEEE International Symposium on High Performance Distributed Computing*, pages 359–366, Edinburgh, Scotland, UK, July 2002.

[SME04] Jove M. P. Sinaga, Hashim H. Mohamed, and Dick H. J. Epema. A dynamic co-allocation service in multicluster systems. In *JSSPP'04: Proceedings in 10th Job Scheduling Strategies for Parallel Processing*, pages 194–209, 2004.

[Smi04] Warren Smith. *Grid Resource Management - State of the Art and Future Trends*, chapter 16. Improving Resource Selection and Scheduling Using Predictions. Kluwer Academic Publishers, 2004.

[SSB+95] T. Sterling, D. Savarese, D. J. Becker, J. E. Dorband, U. A. Ranawake, and C. V. Packer. Beowulf: A parallel workstation for scientific computation. In *ICPP'95: Proceedings of the 24th Internatio-*

nal Conference on Parallel Processing, pages I:11–14, Oconomowoc, WI, 1995.

[SSN99] Sanjeev Setia, Mark S. Squillante, and Vijay K. Naik. The impact of job memory requirements on gang-scheduling performance. *SIGMETRICS Performance Evaluation Review*, 26(4):30–39, 1999.

[STF99] W. Smith, V. Taylor, and I. Foster. Using run-time predictions to estimate queue wait times and improve scheduler performance. In Dror G. Feitelson, editor, *JSSPP'99: Proceedings of the 5th Workshop on Job Scheduling Strategies for Parallel Processing*, pages 202–219, San Juan, Puerto Rico, April 1999.

[Sto07] Heinz Stockinger. Defining the grid: a snapshot on the current view. *Journal of Supercomputing*, 42(1):3–17, Oct. 2007.

[SvANS00] J. Santoso, G. D. van Albada, B. A. A. Nazief, and P. M. A. Sloot. Hierarchical job scheduling for clusters of workstations. In *ASCI'00: Proceedings of the sixth annual conference of the Advanced School for Computing and Imaging*, pages 99–105, Lommel, Belgium, June 2000.

[SW02] W. Smith and P. Wong. Resource selection using execution and queue wait time predictions. *NAS Technical Reports*, 2002.

[Tan05] Andrew S. Tanenbaum. *Structured Computer Organization (5th Edition)*. Prentice Hall, June 2005.

[TD01] Thyagaraj Thanalapati and Sivarama Dandamudi. An efficient adaptive scheduling scheme for distributed memory multicomputers. *IEEE Transaction on Parallel and Distributed Systems*, 12(7):758–768, 2001.

[TEF07] D. Tsafrir, Y. Etsion, and D. G. Feitelson. Backfilling using system-generated predictions rather than user runtime estimates. *IEEE Transaction on Parallel and Distributed Systems*, 18(6):789–803, 2007.

[TOP] TOP500 supercomputer sites - http://www.top500.org/.

[TRA+05] Andrei Tchernykh, Juan Manuel Ramírez, Arutyun Avetisyan, Nikolai Kuzjurin, Dmitry Grushin, and Sergey Zhuk. Two level job-scheduling strategies for a computational grid. In *PPAM'05: Proceedings of the sixth International Conference on Parallel Processing and Applied Mathematics*, pages 774–781, Poznan, Poland, Sep. 2005.

[TWG+01] Valerie Taylor, Xingfu Wu, Jonathan Geisler, Xin Li, Zhiling Lan, Mark Hereld, Ivan R. Judson, and Rick Stevens. Prophesy: Automating the modeling process. In *AMS '01: Proceedings of the Third Annual International Workshop on Active Middleware Services*, page 3, San Francisco, CA, USA, Aug. 2001. IEEE Computer Society.

[US04] B. Urgaonkar and P. Shenoy. Sharc: Managing cpu and network bandwidth in shared clusters. *IEEE Transaction on Parallel and Distributed Systems*, 15(1):2–17, 2004.

[Vol] Volunteer@Home volunteer computing. http://www.volunteerathome.com.

[VRMCL09] Luis M. Vaquero, Luis Rodero-Merino, Juan Caceres, and Maik Lindner. A break in the clouds: towards a cloud definition. *SIGCOMM Computing Communication Review*, 39(1):50–55, 2009.

[wLI] LINGO. http://www.lindo.com/index.php?option=com_content &view=article&id=2&itemid=10.

[WM97] D. Randall Wilson and Tony R. Martinez. Improved heterogeneous distance functions. *Journal of Artificial Intelligence Research*, 6:1–34, 1997.

[Wol03] R. Wolski. Experiences with predicting resource performance online in computational grid settings. *ACM SIGMETRICS Performance Evaluation Review*, 30(4):41–49, 2003.

[WSH99] Rich Wolski, Neil T. Spring, and Jim Hayes. The network weather service: a distributed resource performance forecasting service for metacomputing. *Future Generation of Computer Systems*, 15(5-6):757–768, 1999.

[WSH00] Richard Wolski, Neil T. Spring, and Jim Hayes. Predicting the CPU availability of time-shared unix systems on the computational grid. *Cluster Computing*, 3(4):293–301, 2000.

[Xu01] Ming Q. Xu. Effective metacomputing using LSF multicluster. In *CCGRID'01: Proceedings of the 1st International Symposium on Cluster Computing and the Grid*, page 100, Brisbane, Australia, May 2001. IEEE Computer Society.

[YA07] E. Yom-Tov and Y. Aridor. A self-optimized job scheduler for heterogeneous server clusters. In *JSSPP'07: Proceeding of 13th Workshop*

on Job Scheduling Strategies for Parallel Processing, pages 169–187, Seattle, WA, USA, June 2007. Springer-Verlag.

[YMM05] Leo T. Yang, Xiaosong Ma, and Frank Mueller. Cross-platform performance prediction of parallel applications using partial execution. In *SC2005: Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, page 40, Seattle, WA, USA, Nov. 2005. IEEE Computer Society.

[YSF03] L. Yang, J.M. Schopf, and I. Foster. Conservative scheduling: Using predicted variance to improve scheduling decisions in dynamic environments. In *SupcerComputing'03: Proceedings of the ACM/IEEE International Conference for High Performance Computing and Communications*, pages 262–273, Phoenix, AZ, USA, Nov. 2003.

[Yue04] Jianhui Yue. Global backfilling scheduling in multiclusters. In *AACC'04: Proceedings of the International Conference on Information Technology - Asian Applied Computing Conference*, pages 232–239, Kathmandu, Nepal, Oct. 2004. Springer-Verlag.

[ZZWD93] Songnian Zhou, Xiaohu Zheng, Jingwen Wang, and Pierre Delisle. Utopia: a load sharing facility for large, heterogeneous distributed computer systems. Technical report, 1993.