UNIVERSITAT JAUME I DE CASTELLÓ
E. S. DE TECNOLOGIA I CIÈNCIES EXPERIMENTALS

Metodología de predicción de rendimiento de aplicaciones MapReduce iterativas sobre una nube híbrida

Castellón de la Plana, Junio 2017

Tesis Doctoral

Presentado por: Francisco José Clemente Castelló
Supervisado por: Juan Carlos Fernández Fernández
Bogdan Nicolae

UNIVERSITAT JAUME I DE CASTELLÓ
E. S. DE TECNOLOGIA I CIÈNCIES EXPERIMENTALS

Metodología de predicción de rendimiento de aplicaciones MapReduce iterativas sobre una nube híbrida

Francisco José Clemente Castelló

Índice general

Índice de figuras

Índice de tablas

Dedicat a tu, pare,
per tot allò que m'has donat a la vida!
Espere que estigues orgullós de mi allà on estigues.
T'estime! Gràcies per tot!

Sisco.

Summary

Due to exploding data sizes and the need to combine multiple data sources, traditional on-premise big data analytics becomes insufficient, as the capacity of private-owned data centers cannot accommodate the increasing scale and scope of the applications. To address this issue, *hybrid cloud bursting* [48] (complement on-premise virtual machines with temporary off-premise virtual machines) has seen a rapid increase in popularity among big data analytics users.

However, leveraging hybrid cloud bursting for big data analytics is challenging. On one hand, a core data-centric design choice of big data analytics frameworks like Hadoop MapReduce is the ability to exploit data locality to schedule computations close to the data. On the other hand, the use of both on-premise and off-premise virtual machines creates a "weak link" that is at least an order of magnitude slower than the intra-cloud network links. Despite these challenges, certain classes of applications, in particular iterative applications are well positioned to take advantage of hybrid cloud bursting. However, since the potential benefits are difficult to predict, an equally important challenge is to *estimate the hybrid runtime and cost* in advance. This enables more informed business decision making where speedup vs. extra cost can be weighted before committing an investment.

This thesis addresses the aforementioned challenges by overarching several contributions that coverge towards a holistic solution to optimize running iterative MapReduce applications in hybrid cloud bursting setups. Specifically, the thesis introduces an instrumentation and statistical extraction toolchain that facilitates a fine-grain analysis and visualization of each of the stages of a MapReduce job. Using this toolchain, the thesis explains the challenges and brings new insight into the implications they have on the design of the MapReduce runtime. This insight is then leveraged to propose two novel techniques to leverage data locality efficiently in a hybrid cloud setup: on one hand, an asynchronous off-premise replica migration strategy to enable the MapReduce runtime to exploit data locality, on the other hand a hybrid-aware task scheduling policy that complements the migration strategy to avoid redundant data transfers over the weak link. Based on these two techniques that make iterative MapReduce applications on hybrid cloud bursting deployments feasible, the thesis contributes with a cost model that enables reasoning about the pay-as-you-go cost. Finally, it concludes with a methodology to estimate the completion time of the applications, which, together with the cost model, facilitate informed business decisions.

Resumen

La capacidad de los centros de datos privados para el tradicional análisis de grandes volúmenes de datos (o Big Data) se está volviendo insuficiente. Esto es debido al actual crecimiento del tamaño de los datos y a la necesidad de combinar múltiples fuentes de datos. Para solucionar este problema, las nuevas tecnologías como la computación en la nube, y en concreto el *cloud bursting* [48] (despliegue de nube híbrida que complementa temporalmente máquinas virtuales locales con máquinas virtuales remotas) ha experimentado una rápida adopción entre los usuarios de análisis de Big Data.

Conseguir un buen rendimiento en este tipo de despliegues de nube híbrida para el análisis de Big Data supone un reto. Por una parte, el diseño central de los *frameworks* de análisis de Big Data como Hadoop MapReduce están orientados a explotar la localidad de los datos planificando las tareas cerca de la ubicación de los mismos. Por otra parte, el uso de una nube híbrida implica la existencia de una red de comunicación entre ambas nubes (*enlace débil*), que posee un ancho de banda un orden de magnitud inferior al resto de redes internas. A pesar de estas implicaciones, hay ciertos tipos de aplicaciones que pueden aprovechar sus características intrínsecas para obtener un buen rendimiento en este tipo de despliegues, en concreto las aplicaciones MapReduce iterativas. Sin embargo, dado que estos beneficios son difíciles de predecir, otro reto importante es estimar el tiempo de ejecución y el coste económico de este tipo de aplicaciones. Esto permite tomar mejores decisiones empresariales que obtengan un buen compromiso entre rendimiento y coste económico de las aplicaciones, gracias a su estimación previa.

En esta tesis se abordan los retos anteriormente citados mediante la aportación de varias contribuciones que convergen a una solución holística, que optimiza la ejecución de aplicaciones MapReduce iterativas sobre un despliegue de *cloud bursting*. Concretamente, esta tesis introduce un entorno compuesto por una colección de herramientas de extracción estadística para Hadoop MapReduce, que facilita el análisis detallado y la visualización de tareas de cada una de las etapas de un trabajo MapReduce. Esta colección permite obtener un nuevo enfoque de las implicaciones que conllevan este tipo de aplicaciones en una nube híbrida. A partir de este nuevo enfoque se proponen dos nuevas estrategias de aprovechamiento eficiente de la localidad de los datos: por una parte, una estrategia de migración asíncrona de una réplica de los datos, ubicados inicialmente en las máquinas virtuales locales, hacia las máquinas virtuales remotas; por otra parte, una nueva política de planificación de tareas adaptada a despliegues de *cloud bursting*, que complementa la estrategia de migración de datos para evitar transferencias redundantes sobre el *enlace débil*. Basándose en

las dos estrategias propuestas, que hacen factible el empleo de aplicaciones MapReduce iterativas en un despliegue de *cloud bursting*, esta tesis contribuye con un modelo de coste económico que permite estimar todos los gastos incurridos. Finalmente, se contribuye con una metodología de predicción del tiempo de finalización de aplicaciones MapReduce, la cual, junto con el modelo de coste económico, facilita la toma de decisiones empresariales acerca de la viabilidad de este tipo de despliegues.

Agradecimientos

Después de un largo viaje de cuatro intensos años de dedicación, esfuerzo, y trabajo, la tesis ha llegado a su fin. Ha sido una etapa difícil de mi vida que ha supuesto muchos sacrificios para conseguir este objetivo, pero también ha conllevado grandes satisfacciones. Sobre todo, ha sido una etapa de retos, experiencias y superación personal que me ha ayudado a adquirir grandes conocimientos y a mejorar en todos los aspectos de mi vida, tanto a nivel personal como profesional. Sin embargo, todo el esfuerzo que hay detrás de esta tesis no hubiera sido posible sin la colaboración, el apoyo y la dedicación que me han brindado muchas personas. A todas ellas, mi más sincero agradecimiento.

En primer lugar me gustaría agradecer la dedicación y el apoyo de mis directores y tutores Rafael Mayo y Juan Carlos Fernández, los cuales confiaron en mí desde el primer momento y me han guiado a lo largo de esta investigación. A ellos, darles las gracias por todos los consejos, el apoyo, la constancia y el esfuerzo que me han dedicado en todo momento para culminar esta tesis. Gracias por hacer realidad este sueño.

I would like to express my gratitude to my director Bogdan Nicolae. He is an inexhaustible source of brilliant ideas, which have guided the research of this thesis. This would not have been possible without his help. I'm very thankful for all his support and effort during the whole research.

Agraceder a la Universitat Jaume I, al grupo HPC&A y al *Ministerio de Educación, Cultura y Deporte* por financiar esta investigación a través del programa FPI, y por ofrecerme todas las instalaciones y recursos sin los cuales no hubiera sido posible este trabajo.

Asimismo, dar las gracias a las personas del grupo HPC&A de la Universitat Jaume I y a las que en algún momento han sido parte de él: Enrique, Germán L., Maribel, Sergio B., Rafa M., Juan Carlos, José Manuel, Asun, Maribel, José I. Aliaga, Manel, Maria, Héctor, Sandra, Adrián, Sergio I., Rafa R., José Antonio, Goran, Rocío y Sonia. Gracias a todos por vuestra colaboración y por el buen ambiente de trabajo. Agradecer también a los técnicos Vicente y Gustado todo el soporte técnico que me han proporcionado. Me gustaría hacer una mención especial a Enrique Quintana por ofrecerme la oportunidad de pertenecer a este grupo, por abrirme las puertas al centro de investigación IBM Research, y por darme todo el soporte necesario para hacer realidad esta tesis.

De mi estancia en IBM Research Dublín, una de la experiencias más intensas que he tenido en mi vida, quiero dar las gracias a todos los miembros del grupo de investigación *High Performance*

Systems, y en especial a Kostas Katrinis y a Bogdan Nicolae por darme la oportunidad de formar parte de su equipo y aportar tantos conocimientos a mi investigación.

A todos mis amigos y amigas de la NBA, a mis compañeros de la orquesta Supermagic, de la Onda Big Band, de la Fet a Posta, de la banda y del coro de la Unión Musical de Onda, por estar a mi lado y animarme en todo momento. Gracias por ayudarme a hacer más ameno este camino y por la paciencia que habéis tenido conmigo.

A todos mis compañeros de Photonic Sensors and Algorithms: Adolfo, Carles, Iván, Leticia, Arnau, Javier, Ann, y en especial a Jorge Blasco, por todo el apoyo y flexibilidad que me habéis dado para conseguir este objetivo. Me siento orgulloso de formar parte de este magnífico equipo.

Por supuesto, agradecer a mis padres Paco y Carmen, así como a mi hermana Carmina todo su apoyo, sus incansables ánimos, y su paciencia durante todo este tiempo. *Gràcies mare, pare i germana per regalar-me esta oportunitat a la vida. I a tu pare, encara que ja no estigues, gràcies per haver cregut en mi fins l'últim moment. No tinc prou paraules d'agraïment per a vosaltres.*

Finalmente, darle infinitas gracias a Carla, la mejor persona que he conocido en mi vida. Ella ha sido y es mi apoyo incondicional, mi fuente de inspiración, mi compañera de vida. Persona a la que admiro y de la cual nunca dejaré de aprender. A ella no me cansaré de agradecerle el cariño, la paciencia y el apoyo que me ha dado en todo momento. Gracias por levantarme cuando estaba cayendo, por ser mi compañera en todas las experiencias buenas y malas de esta larga aventura. Gracias por su confianza en mí, y en definitiva, por estar siempre a mi lado. *Perquè junts anem més lluny! Gràcies Carla! Sobren les paraules, gràcies per tot!*

– ¡Gracias! · Gràcies! · Thanks! –

Castellón, junio de 2017.

CAPÍTULO 1

Introducción

1.1 Motivación

La cantidad de datos que generamos a diario crece de forma exponencial a lo largo de los años. Vivimos en una era en la que todos los datos generados por el ser humano en cualquier tipo de dispositivo electrónico pasan a ser un valor muy preciado para el mundo empresarial, y esto es solo el principio. Durante los dos últimos años hemos generado más datos que en toda la historia previa de la raza humana [66]. Las previsiones indican que en 2020, cada persona de la tierra generará 1,7 Megabytes de datos nuevos por segundo [37], y nuestro universo digital habrá crecido de los 4,4 Zettabytes de hoy en día a los 44 Zettabytes, o 44 trillones de Gigabytes [38]. Toda esta capacidad de almacenar datos sería totalmente vana si no fuéramos capaces de tratarlos y extraer información que fuera útil para mejorar distintos aspectos de la vida. La extracción de información relevante a partir de esta gran cantidad de datos es lo que ha hecho crecer una serie de nuevas ramas de la tecnología como: *Data Science*, *Data Analytics* y *Big Data*. Todas estas ramas tienen en común el tratamiento de grandes volúmenes de datos (aunque desde diferentes puntos de vista) con el fin de obtener información relevante que permita tomar decisiones a partir de ella. Es evidente que el almacenamiento de todos estos datos y el tratamiento de los mismos requiere, por una parte de grandes infraestructuras, y por otra de paradigmas de programación que faciliten la implementación de los algoritmos (normalmente paralelos) utilizados. Es aquí donde entra en juego el paradigma de la computación en la nube (*cloud computing*), como plataforma donde poder ejecutar los trabajos relativos al tratamiento de inmensos volúmenes de datos. Almacenar y tratar inmensos volúmenes de datos requiere un gran despliegue de infraestructura, normalmente solo al alcance de grandes empresas o entidades públicas. Sin embargo, el desarrollo de la tecnología de computación en la nube posibilita que medianas y pequeñas empresas puedan acceder, con unos costes asumibles, a grandes plataformas y obtener beneficios del análisis de datos a gran escala. La computación en la nube se puede ver como la confluencia de dos tecnologías cuya madurez se consiguió durante la primera década del siglo XXI: el *grid computing* y la virtualización de sistemas. De hecho, las ideas que aparecieron al principio con la tecnología de *grid* siguen estando vigentes en la computación en la nube, pero añadiéndole la ventaja de la infraestructura de virtualización. La conjunción de estas dos tecnologías ha permitido el diseño de sistemas de servicios informáticos

bajo demanda (desde un nivel de plataforma básica hasta el de aplicación), con un esquema de pago por uso.

Las nuevas tecnologías aparecidas en los últimos años, como la computación en la nube, han revolucionado el uso del análisis de Big Data gracias a la capacidad de disponer de recursos de cómputo como un servicio en red bajo demanda, escalable y adaptable a las necesidades de cada entidad.

El modelo de servicio conocido como *Nube Pública* ha transformado la tecnología de la información (TI) durante los últimos años gracias a un modelo de pago por uso [12] (p. ej. Amazon AWS), y la habilidad de satisfacer rápidamente las necesidades de un gran volumen de usuarios a través de un conjunto común de recursos TI virtualizados y/o físicamente localizados. Sin embargo, distintos requisitos tanto técnicos (p. ej. aislamiento del rendimiento, necesidades de tecnología personalizada) como empresariales (p. ej. seguridad, confidencialidad) han provocado la aparición de un modelo alternativo de nube conocido como *Nube Privada* [52]. A pesar de sus muchos puntos en común con la nube pública en términos de tecnología subyacente y pilas de software, el principal diferenciador del modelo privado radica en la entrega de servicios en la nube a través de una infraestructura física dedicada exclusivamente al usuario, que normalmente es gestionada por la propia entidad o un proveedor externo (p. ej. IBM Softlayer Private Cloud). Además, suele combinarse con otros servicios de infraestructura de base gestionados por el proveedor, como aprovisionamiento, seguridad o calidad de servicio.

Recientemente, ha surgido un nuevo modelo de servicio en la nube conocido como *Nube Híbrida*. Como el término implica, el modelo híbrido abarca una combinación de dos o más instancias de los modelos descritos anteriormente, típicamente en forma de público/privado: un conjunto de máquinas virtuales (MVs) alojadas en la nube privada (*on-premise* o infraestructura local) son complementadas con MVs alojadas en una nube pública (*off-premise* o infraestructura remota). Las razones empresariales que impulsan tal evolución son múltiples, algunas de las cuales son por ejemplo el *cloud bursting* [48], es decir, la extensión temporal de nubes privadas con recursos de nubes públicas para superar picos de cargas de trabajo, sin extender la nube privada con infraestructura física adicional; o la recuperación de desastres [57] (mover/duplicar los datos a una nube pública para sobrevivir a los posibles errores extensivos de la nube privada), combinando datos sensibles de la nube privada con datos abiertos alojados en las nubes públicas, etc. Los servicios ofrecidos por las nubes híbridas han comenzado a estar ya comercialmente disponibles. Ejemplos de ellos los podemos encontrar en VMWare vCloud vAir [94] (infraestructura como servicio) o Rackspace Cloud Hosting Híbrido [79] (plataforma como servicio). Además, los estudios de mercado [1] pronostican un tamaño de mercado de nubes híbridas valorado en 80.000 millones de dólares en 2018, es decir, una tasa media de crecimiento anual del 30 % hasta 2018.

1.1.1 Nubes Híbridas para el análisis de Big Data

El análisis de Big Data requiere de grandes infraestructuras de cómputo que permitan almacenar y procesar grandes volúmenes de datos de forma rápida y eficiente para extraer información de valor intrínseco. Debido al gran crecimiento y a la diversidad de datos que se está produciendo actualmente, no todas las empresas u organizaciones son capaces de almacenarlos y procesarlos. Concretamente, la capacidad de las infraestructuras informáticas de pequeñas y medianas empresas es muy limitada, lo que hace que solo con dichas infraestructuras las ventajas que presenta el análisis de Big Data queden mermadas.

Una de las tecnologías que pueden ayudar a solventar este problema es el *cloud bursting*, cuya utilización está aumentando entre los usuarios de análisis de Big Data. Este tipo de nube es una forma de computación en la nube de forma híbrida. El *cloud bursting* permite el incremento temporal de recursos locales (de ahora en adelante *on-premise*) de un centro de datos privados con recursos remotos (de ahora en adelante *off-premise*) adicionales de un proveedor de servicios de nube pública para cubrir la demanda producida por un pico de trabajo. De esta forma, permite superar las limitaciones de los centros de datos privados en momentos puntuales donde se requiere más capacidad de cómputo de la que puede proporcionar la infraestructura propia, todo ello de una forma flexible a través de un modelo de pago por uso.

Sin embargo, el aprovechamiento de este tipo de nubes híbridas para la ampliación temporal de recursos no viene sin retos: compartir, diseminar y analizar los conjuntos de datos puede ocasionar un frecuente movimiento de datos a gran escala entre la nube pública y privada. Esto plantea un desafío importante para el *cloud bursting*, ya que los usuarios normalmente acceden a las nubes públicas a través de redes de área extensa de alta latencia y bajo rendimiento, haciendo que las transferencias de datos entre las nubes sean más lentas que las transferencias de área local. Por esta razón, en esta tesis denominamos *enlace débil* al enlace que existe entre una nube privada y una nube pública.

Entre los diversos servicios de software que hacen uso del modelo de nube híbrida, el análisis de datos es uno de los casos más difíciles, especialmente cuando hay grandes volúmenes de datos para analizar [104, 87]. Los *frameworks* convencionales para el análisis de Big Data asumen que su ejecución será en un entorno físicamente localizado entre servidores con conexiones de alta velocidad. Esta suposición queda relegada para entornos de nube híbrida, ya que comprenden centros de datos separados físicamente, los cuales están interconectados a través de una red (*enlace débil*) que es al menos un orden de magnitud menor en cuanto a rendimiento y ancho de banda (ya sea conectividad dedicada o a través de Internet). Además, debido al gran volumen de datos que se analiza, la mayoría de las técnicas de análisis de datos hacen un uso intensivo de la localidad de los datos para que su ejecución sea lo más cercana posible a los datos a procesar, evitando así un innecesario tráfico de datos a través de la red. Este aspecto supone un reto cuando se realiza un despliegue de recursos temporales de una nube pública para complementar los recursos de una nube privada con el fin de finalizar más rápidamente el análisis de Big Data: las MVs recientemente aprovisionadas no contienen datos y, como tal, es necesario enviar grandes cantidades de datos a través del *enlace débil* para poder aprovechar la capacidad computacional adicional.

En este contexto, los grandes *frameworks* de análisis de datos a gran escala, como Hadoop [8] o Spark [86], no han alcanzado un nivel suficiente de implementación para adaptarse a un despliegue temporal de una nube híbrida como es el *cloud bursting*. Concretamente, las diferentes opciones de diseño para la optimización del tiempo de ejecución de las aplicaciones, como por ejemplo las políticas de planificación de tareas cercanas a los datos, son aspectos fundamentales que han demostrado su eficacia en centros de datos individuales. En cambio, en un escenario de *cloud bursting*, los recursos recién aprovisionados en remoto no tienen ningún dato para procesar; por tanto, la localidad de datos no puede ser aprovechada directamente, sino que es necesario enviar los datos a los recursos *off-premise* a través del *enlace débil* antes de poder aprovechar la potencia computacional adicional.

Muchas de las técnicas de análisis de datos requieren un procesamiento iterativo de los datos, incluyendo PageRank [76], HITS (Hypertext-Induced Topic Search) [59], consultas recursivas a bases de datos [14], clustering, análisis de redes neuronales, análisis de redes sociales, o análisis

de tráfico de red [19]. Todas estas técnicas comparten el mismo procedimiento: ejecutan el mismo trabajo una y otra vez apoyándose de los mismos datos de entrada y datos intermedios de las iteraciones anteriores para refinar el resultado. En este tipo de aplicaciones la localidad de los datos resulta ser un factor clave en un entorno de nube híbrida, ya que permite aprovechar que los datos de entrada se hayan replicado *off-premise*. Por lo tanto, el problema de ejecutar la misma aplicación de forma híbrida a través de una mezcla de ambos recursos *on-premise* y *off-premise* se puede considerar desde una nueva perspectiva.

1.1.2 Enfoque del trabajo

Para finalizar esta motivación vamos a introducir el enfoque del trabajo que se presenta en esta tesis. Ha quedado patente en la motivación que tanto el análisis de Big Data como la utilización de la computación en la nube, y en particular la utilización de modelos de nube híbrida, son tecnologías recientes que todavía tienen margen de mejora y estudio. Nuestro trabajo se centra precisamente en dicha combinación de **Big Data+Nube Híbrida**. Uno de los problemas abiertos en este ámbito es la predicción del tiempo de finalización y del coste económico de la ejecución de una aplicación de Big Data sobre un entorno de computación en la nube. Este problema es de especial interés dado que el pago por uso hace que se tengan que acomodar los parámetros de rendimiento al coste económico (en cuanto a tiempo de finalización).

Como se verá en el siguiente apartado este problema ya ha sido tratado cuando nos encontramos con un entorno de nube estándar, donde todos los recursos pertenecen a una misma nube. Sin embargo, las soluciones aportadas en esta situación, como se mostrará a lo largo de esta tesis, no son aplicables cuando nos encontramos ante el uso de un entorno de computación de nube híbrida, que presenta unas características especiales. En lo que resta de documento se estudiará y propondrá una solución a este problema, aportando además distintas soluciones para mejorar el rendimiento de estas aplicaciones sobre un entorno de nube híbrida.

1.2 Estado del arte

En esta tesis se trabaja sobre cuatro focos muy concretos:

- Las implicaciones de utilizar una arquitectura de nube híbrida, mediante un despliegue de *cloud bursting*, para la ejecución de aplicaciones MapReduce iterativas.
- A partir del estudio de dichas implicaciones, proponer alternativas que permitan mejorar el rendimiento en ese ámbito concreto.
- Proporcionar una metodología que permita estimar el tiempo de finalización de una aplicación MapReduce iterativa ejecutada sobre una nube híbrida, a través de un despliegue de *cloud bursting*.
- Proporcionar un modelo de coste económico que permita estimar el coste que conlleva ampliar temporalmente los recursos locales con recursos de cómputo adicionales de una nube pública, para aplicaciones MapReduce iterativas.

A continuación, se analiza el estado del arte de otros trabajos previos que se han realizado en estos ámbitos y que sirven de referencia para el trabajo desarrollado en esta investigación.

1.2.1 Gestión de recursos en la ejecución de trabajos MapReduce

Uno de los aspectos que se trata en este trabajo es la gestión de los recursos disponibles en la ejecución de aplicaciones MapReduce. Dentro de la implementación Hadoop de MapReduce se han introducido progresivamente los planificadores *FIFO*, *Capacity* y *Fair Scheduler*. Este último intenta maximizar la localidad de acceso a los datos. La elección de la mejor política de acceso a los datos es el gran problema que se plantea a la hora de planificar las diferentes tareas de una aplicación MapReduce sobre un clúster de computadores. A partir de la política de planificación *Fair Scheduler* se han propuesto otras políticas que intentan mejorar este esquema, como por ejemplo las descritas en el trabajo de Isard et al. [55] o de Wolf et al. [100]. LATE [102] es un planificador que consigue mejoras en el tiempo de finalización de trabajos MapReduce en entornos heterogéneos. Kllapi et al. [60] proponen un algoritmo de planificación general que intenta optimizar tanto el tiempo de finalización como el gasto económico en una nube pública mediante un mapeado previo de las tareas y de los nodos de cómputo.

Otros trabajos están basados en la gestión de recursos para cumplir con determinados requisitos económicos. Silva et al. [84] presentan un heurístico para optimizar el número de MVs para un conjunto de trabajos, mientras se minimiza el tiempo de finalización global bajo un presupuesto determinado. Este trabajo asume que el usuario no tiene conocimiento del tiempo de finalización del trabajo, comienza con una sola MV y va añadiendo más MVs a la nube pública de forma gradual basándose en el tiempo de finalización medio del trabajo, el cual se actualiza cada vez que finaliza un trabajo.

Wang et al. [97] han diseñado un algoritmo de planificación consciente del gasto económico de aplicaciones MapReduce en una nube heterogénea. Consideran trabajos MapReduce iterativos que constan de múltiples etapas para su finalización, donde cada etapa contiene un conjunto de tareas *Map* o *Reduce*. El objetivo de la optimización es seleccionar una MV de un conjunto fijo de MVs heterogéneas para cada tarea, con el objetivo de minimizar el tiempo de finalización o el coste económico.

1.2.2 Modelos de coste económico para cloud computing

También se han realizado esfuerzos para definir modelos de coste económico. La importancia de considerar múltiples factores de coste se enfatiza en el trabajo de Kashef et al. [58], donde los autores identifican hasta 20 factores que clasifican en seis categorías: electricidad, hardware, software, mano de obra, infraestructura del local y servicios de la nube. La última categoría incluye el cargo por conectividad de red, el coste de uso de las horas de CPU, el coste de transferencia de datos desde/hacia la nube y la cantidad de almacenamiento utilizado en la nube. Algunos de estos factores, tales como el tiempo de uso de CPU o el coste de las transferencias de datos entre nubes, se han considerado en esta tesis a la hora de presentar el modelo de coste económico desarrollado en esta investigación, centrado en el coste económico de un despliegue de *cloud bursting* (es decir, el coste de emplear temporalmente recursos *off-premise*).

Una comparación económica que considera varios factores de la categoría de servicios de la nube se presenta en el trabajo de Kondo et al. [61]. Armbrust y et. al [13] han propuesto una fórmula sencilla para tomar decisiones sobre el compromiso coste-rendimiento de la migración de una carga de trabajo a una nube, lo que incluye determinar qué se debe mantener *on-premise* y qué se debe transferir a los recursos *off-premise*.

Truong et al. [89] presentan un modelo para estimar y monitorizar el coste económico. Este modelo considera tres situaciones: el uso completo de los recursos *on-premise*, el uso parcial de los recursos *off-premise* o el uso completo de los recursos *off-premise*. Alford et al. [4] presentan un modelo de coste que es propiedad de Booz Allen Hamilton Inc. Este modelo de coste se utiliza para comparar tres escenarios: nube pública, nube híbrida y nube privada. Específicamente en el contexto MapReduce, un modelo simple de coste para nubes híbridas se presenta en el trabajo de Ruiz-Alvarez et al. [81] para dos clases de aplicaciones: MapReduce y Monte Carlo.

El modelo de coste presentado en esta tesis se centra en aplicaciones iterativas de MapReduce que se ejecutan en un escenario de *cloud bursting*. En este contexto, se necesita un modelo de coste flexible que pueda aplicarse a múltiples estrategias híbridas de *cloud bursting* para extraer tendencias de comportamiento para un número creciente de iteraciones. Para ello, hemos creado un nuevo modelo de coste, basado en los modelos presentados en la literatura, que incorpora los factores necesarios para ofrecer un modelo de coste adaptado a este nuevo escenario. Desde nuestro conocimiento, el modelo presentado en esta tesis es el primero en enfocarse en este tipo de despliegues y aplicaciones.

1.2.3 Optimizaciones de rendimiento de MapReduce en entornos heterogéneos

Originalmente, Hadoop fue diseñado para un entorno homogéneo, pero hay un interés reciente en el estudio de MapReduce en entornos heterogéneos, en concreto centrado en la optimización del rendimiento de la ejecución de aplicaciones MapReduce en dichos entornos. Zaharia et al. [102] se centran en eliminar el efecto negativo que producen las tareas *stragglers* (tareas que progresan lentamente) sobre el tiempo de finalización del trabajo, mejorando la estrategia de planificación con tareas especulativas. El proyecto Tarazu [3] proporciona una planificación consciente de la comunicación en el cálculo de la fase *Map*, que tiene como objetivo disminuir la sobrecarga de comunicación cuando los nodos más rápidos procesen las tareas *Map* con datos de entrada almacenados en nodos lentos. También propone un equilibrio de carga para asignar cantidades diferentes de trabajo para la fase *Reduce* en función de la capacidad del nodo. Xie et al. [101] intentan mejorar el rendimiento de MapReduce a través de una estrategia de asignación de datos consciente de la heterogeneidad: un nodo más rápido almacena una mayor cantidad de datos de entrada. De esta manera, se pueden ejecutar tareas en nodos más rápidos sin una transferencia de datos para la ejecución de la tarea *Map*. Polo et al. [78] muestran que algunas aplicaciones MapReduce pueden mejorar su rendimiento mediante el uso de hardware especial. De esta forma el planificador que diseñan asigna dichos trabajos a los nodos con dicho hardware especial.

1.2.4 Optimizaciones de rendimiento de MapReduce en nubes híbridas

Las aplicaciones MapReduce han sido ampliamente estudiadas en plataformas de computación en la nube [90, 88, 47, 105]. Sin embargo, a diferencia de las nubes híbridas, donde los recursos locales

son limitados, la suposición general es que la aplicación completa se está ejecutando remotamente y tiene acceso a recursos de computación prácticamente ilimitados gracias a la elasticidad.

Varios esfuerzos se han centrado en mejorar el rendimiento de los *frameworks* de MapReduce para entornos híbridos. HybridMR [83] propone una solución para aprovechar las redes de escritorio híbridas y nodos externos de computación. Sin embargo, no se considera el aspecto de expandir y reducir una configuración de MapReduce dinámicamente. HadoopDB [2] propone un sistema híbrido que combina un despliegue de Hadoop y un sistema de bases de datos paralelo para aprovechar al mismo tiempo la elasticidad y escalabilidad de MapReduce, junto con el rendimiento y la eficiencia de las bases de datos paralelas.

En varios estudios se han propuesto estrategias de planificación de aplicaciones Hadoop MapReduce para su ejecución en nubes híbridas. Cardosa et al. [20] se centran en el problema de cómo los procesos *Map* y *Reduce* se dividen en plataformas de nube híbridas. Heintz et al. [50] muestran estrategias para planificar tareas *Map* sobre plataformas de nube distribuidas, con el fin de minimizar la degradación del rendimiento que puede resultar de grandes tiempos de comunicación entre las plataformas. Estas estrategias parten de la premisa de que los datos ya se encuentran distribuidos en todos los nodos, sin embargo no tratan el problema de la migración inicial de datos cuando se incorporan nuevos nodos al sistema. En esta tesis presentamos varias estrategias para mejorar la planificación de las tareas en dicho proceso inicial de migración datos entre nodos remotos.

Con respecto a la predicción del rendimiento de cargas de trabajo y a su optimización en nubes híbridas, Van den Bossche et al. [91] han propuesto un modelo de planificación lineal para cargas de trabajo relevantes, mostrando una mejora sustancial sobre dichas ejecuciones. Sin embargo, este modelo es típicamente difícil de escalar y no tiene en cuenta complejidades específicas del *framework*, tales como el movimiento o la localidad de los datos.

Mattess et al. [67] proponen una extensión de la plataforma Aneka Cloud para ejecutar trabajos MapReduce sobre una nube híbrida, que constituye otro trabajo estrechamente relacionado con el desarrollado en esta tesis. Se trata de planificar los trabajos MapReduce de forma que se ejecuten o bien en la nube privada o bien en la nube híbrida. Nuestro trabajo se basa en la ejecución de una aplicación utilizando ambas nubes a la vez. Además, ese trabajo se basa en la suposición de que el sistema de archivos ya se encuentra distribuido en ambas nubes antes de la ejecución de MapReduce. Por tanto, no considera la fase inicial de distribución de datos entre nubes y el equilibrio de datos, una fase importante en un despliegue de *cloud bursting*.

Los estudios de rendimiento han señalado con frecuencia el impacto negativo del *enlace débil* entre los recursos de las nubes híbridas *on-premise* y *off-premise* sobre el rendimiento y la escalabilidad de las grandes aplicaciones de análisis de datos. Ohnaga et al. [74] se centran en el rendimiento de las aplicaciones Hadoop en particular, sin embargo no tratan el problema del envío inicial de los datos locales. Roman et al. [80] se centran en Spark y señalan los gastos indirectos provocados por la fase *shuffle* cuando el ancho de banda entre los recursos *on-premise* y *off-premise* es significativamente pequeño. Esto es similar a los estudios previos que enfatizaron el impacto negativo de las comunicaciones de red en Spark [75].

Bicer et al. [16] abordan el desafío del análisis de datos en una nube híbrida utilizando datos que son pre-particionados y almacenados en los recursos tanto locales como remotos. Proponen un tiempo de ejecución transparente al usuario en el que un componente principal actúa como planificador de trabajos MapReduce entre las nubes locales y remotas. Una suposición similar sobre los datos que ya están disponibles remotamente *off-premise* se hace en el trabajo de Mattess

et al. [67], donde el foco se centra en la aceleración de las solicitudes MapReduce con límites de plazo. A diferencia del enfoque de esta tesis, en estos estudios las transferencias de datos entre MVs locales y remotas no son una preocupación importante.

Zhang et al. [103] proponen un modelo basado en un factor de carga de trabajo para permitir la gestión proactiva de las cargas de trabajo. Se basa en un algoritmo específicamente diseñado para detectar los elementos de datos de acceso frecuente, que luego se utiliza como una predicción de cuándo escalar la nube local a la nube pública más allá de los recursos locales para lograr el equilibrio de carga. Este enfoque se utiliza en aplicaciones donde los procesos están ligeramente acoplados (es decir, poca necesidad de comunicación entre procesos) y altamente sensibles al número de solicitudes por unidad de tiempo (por ejemplo, transmisión de vídeo). La arquitectura híbrida propuesta consiste en una carga de trabajo base que se ejecuta en la nube local y una carga de trabajo puntual que se ejecuta en la nube remota. En cambio, el enfoque desarrollado en esta tesis se centra en una única carga de trabajo MapReduce ejecutada sobre una arquitectura de nube híbrida.

1.2.5 Modelos de rendimiento de aplicaciones MapReduce

En los últimos años, sobre todo a partir de 2010, se han realizado y publicado muchos trabajos relacionados con la obtención de modelos de rendimiento de aplicaciones MapReduce. Morton et al. [71] y en una evolución de dicho trabajo en [72] se plantean las herramientas *ParaTimer* y *Parallax*, que tienen como finalidad la estimación del tiempo de ejecución de un conjunto de accesos a bases de datos convertidas en aplicaciones MapReduce. Esta estimación se realiza en base a información de rendimiento de ejecuciones previas de dichos accesos realizados con la finalidad de depuración. Las limitaciones de este trabajo vienen dadas por la simplicidad de las funciones *Map* y la utilización de un planificador de tareas exclusivamente FIFO. Otro esfuerzo en esta línea es el que se presenta en el trabajo de Herodotos et al. [51]. En este trabajo los autores siguen una línea similar a la nuestra en cuanto a la utilización de una herramienta de monitorización que permite obtener información detallada de diferentes métricas relativas a la ejecución de una aplicación MapReduce. En este caso, esta información se utiliza para intentar optimizar los parámetros de configuración de una instalación Hadoop. Sin embargo, para conseguir esta optimización se necesita un número muy elevado de ejecuciones de la aplicación, lo que hace esta estrategia poco útil cuando se enfrenta a problemas de gran dimensión.

El trabajo de Zhang et al. [108] es el más cercano al nuestro en cuanto a lo que predicción se refiere. En él se analiza con detalle el *pipeline* de ejecución de una aplicación MapReduce, distinguiendo entre las partes genéricas para todas las aplicaciones y las partes específicas de cada aplicación MapReduce. Se diseña un test que permite extraer las características generales comunes a todas las aplicaciones, mientras que las características específicas de cada aplicación se obtienen a partir de ejecuciones completas de la misma. Con esta técnica se permiten hacer predicciones con una media de error del 10 %. Sin embargo, presenta el problema de necesitar de ejecuciones completas de la aplicación para obtener los parámetros específicos, además de no contemplar la posibilidad de realizar su predicción en infraestructuras de nube híbrida.

Otros trabajos que intentan obtener modelos de rendimiento, centrados en entornos de ejecución homogéneos, son por ejemplo el trabajo de Tian et al. [88], en el cual proponen un modelo que estima el rendimiento de un trabajo a partir de la ejecución de un conjunto de tests sobre un reducido número de nodos y de datos de entrada. Chen et al. [21] proponen el modelo denominado

CRESP, que emplea la técnica de búsqueda a través de fuerza bruta para proporcionar los recursos óptimos que necesita un clúster para ejecutar trabajos Hadoop, en términos de slots para las fases *Map* y *Reduce*. Sin embargo, el número de tareas *Reduce* debe ser igual al número de slots para la fase *Reduce*, lo que significa que solo consideran una sola onda para dicha fase.

Lama et al. [63] proponen AROMA, un sistema que obtiene automáticamente los recursos óptimos para que un trabajo Hadoop logre los objetivos establecidos. Sin embargo, AROMA no proporciona un modelo matemático completo para estimar el tiempo de finalización del trabajo, ni los valores óptimos de parámetros de configuración de Hadoop.

Otra alternativa diferente para obtener modelos de rendimiento es mediante la utilización de simuladores. Por ejemplo en el trabajo de Wang et al. [95] se presenta el simulador MRPerf, que realiza una simulación de grano fino del comportamiento de aplicaciones MapReduce. En la parte de comunicaciones utiliza el simulador *ns-2* de redes, con lo cual el trabajo se centra en determinar el impacto de la utilización de distintos tipos de red en la ejecución de aplicaciones MapReduce en un clúster Hadoop. Otra estrategia de simulación diferente es la que se sigue en el trabajo de Verma et al. [93]. En este caso se enfoca en la simulación de las decisiones del *job master* y en las políticas de asignación de recursos.

Verma et al. [92] presentan ARIA (Automatic Resource Inference and Allocation), un modelo de rendimiento analítico MapReduce que calcula los límites inferior y superior del tiempo de ejecución de un trabajo MapReduce. En el siguiente trabajo de Verma et al. [93] se extiende dicho modelo añadiendo factores de escalado para obtener el tiempo de ejecución para datos de mayor tamaño mediante una regresión lineal simple. En esta tesis se emplea ARIA como base para mostrar que las técnicas empleadas para estimar el tiempo de finalización de aplicaciones MapReduce iterativas en escenarios no híbridos no son lo bastante precisas para su uso en escenarios de *cloud bursting*, por lo que surge la necesidad de crear un modelo de rendimiento específico para este tipo de escenarios, que se corresponde con el trabajo realizado en la tesis.

El trabajo presentado por Zhang et al. [106] divide las fases de *Map* y *Reduce* en seis subfases genéricas (read, collect, spill, merge, shuffle y write), y emplea una regresión para estimar la duración de cada una de las mismas. Los valores estimados se emplean posteriormente en el modelo analítico presentado por Verma et al. [92] para estimar el tiempo total de finalización de un trabajo. El mismo modelo empleado por Verma et al. [92] es utilizado por Zhang et al. [107], pero para entornos heterogéneos. Hasta donde tenemos conocimiento, el trabajo presentado en esta tesis es el primero en introducir un modelo de rendimiento especializado para *cloud-bursting* y aplicaciones MapReduce iterativas.

1.3 Objetivos

El objetivo principal de esta tesis se centra en el **diseño y validación de una metodología que permita estimar el tiempo de finalización de una aplicación MapReduce iterativa ejecutada sobre un despliegue de *cloud bursting***. Para conseguir este propósito principal, a continuación se definen una serie de objetivos específicos:

- Realizar un análisis detallado del comportamiento de la ejecución de aplicaciones MapReduce iterativas sobre un despliegue de *cloud bursting*. Se debe tener en cuenta que, tanto las aplicaciones como la plataforma sobre la que se ejecuta tienen unos condicionantes específicos

que requieren un estudio detallado. Los más inmediatos son, por parte de la aplicación, la reutilización de datos en distintas iteraciones de la ejecución de la aplicación; por parte de la plataforma, la existencia de un enlace de comunicación de bajas prestaciones entre la infraestructura *on-premise* y la infraestructura *off-premise*, por el cual debe pasar todo el tráfico de red.

- Con el propósito de realizar el análisis indicado en el punto anterior es fundamental disponer de información detallada que nos permita extraer los distintos comportamientos de la ejecución de una aplicación MapReduce en este tipo de entornos. Por tanto, se plantea como otro objetivo el diseño e implementación de una herramienta que permita extraer, de forma semiautomática, toda la información relevante de la ejecución de una aplicación MapReduce iterativa sobre una nube híbrida. Esta herramienta debe permitir la visualización de la planificación de tareas que realiza Hadoop en dicho entorno, así como la extracción de información estadística, tanto del sistema de almacenamiento como del sistema de procesamiento de Hadoop.
- Analizar las políticas de planificación utilizadas por Hadoop para adaptarlas a la ejecución de aplicaciones MapReduce iterativas en despliegues de *cloud bursting*. En base a este análisis, proponer y evaluar distintas estrategias que permitan mejorar el rendimiento de las aplicaciones en este tipo de entornos en base a la localidad de los datos. Para ello, se deberá tener en cuenta la migración inicial de los datos de la nube privada a la nube pública.
- Obtener un modelo de coste económico que permita obtener el coste que conlleva la utilización de un despliegue de *cloud bursting* para aplicaciones MapReduce iterativas. Este modelo servirá, entre otras cosas, para comparar las distintas estrategias propuestas de mejora de rendimiento en términos de coste económico.
- Diseñar y validar una metodología de predicción del tiempo de finalización de una aplicación MapReduce iterativa sobre una plataforma de nube híbrida, desplegada mediante un proceso de *cloud bursting*. Esta metodología estará compuesta por tres partes principales:
  - Obtención de un modelo de la plataforma. Con él, deberemos modelizar aquellos parámetros que sean independientes de la aplicación y que tengan repercusión sobre el tiempo de finalización. Este modelo de plataforma, al ser de aspectos independientes de la aplicación, debe ser único y válido para cualquier aplicación que se vaya a ejecutar sobre dicha plataforma.
  - Obtención de un modelo de la aplicación. Esta parte del modelo tendrá información acerca de aquellos aspectos específicos de cada aplicación que deban tenerse en cuenta en la metodología de predicción.
  - Obtención de una expresión matemática que modele todas las fases de una aplicación MapReduce y que, a partir del modelo de la plataforma y de la aplicación, permita estimar su correspondiente tiempo de finalización.

1.4 Estructura de la tesis

Esta tesis está estructurada en seis capítulos. El Capítulo 1 presenta los retos que supone el análisis de Big Data sobre plataformas híbridas de computación en la nube. Además, describe la motivación, estado del arte, objetivos principales, y la estructura de este documento.

El Capítulo 2 describe todos los conceptos básicos acerca de la tecnología utilizada en este documento. Este capítulo se divide en dos partes en las que primero se describen los conceptos relacionados con la computación en la nube, como el concepto de nube híbrida o el de *cloud bursting*, y posteriormente los conceptos relacionados con Big Data, como MapReduce o Hadoop. Además, en este capítulo se describe el entorno experimental utilizado en todos los experimentos realizados en esta tesis.

El Capítulo 3 presenta el entorno de extracción estadística para Hadoop llamado H-Stat. En este capítulo se describe su arquitectura, sus cuatro herramientas de análisis y extracción de datos, y finalmente ejemplos de funcionamiento con datos reales de una aplicación MapReduce iterativa.

En el Capítulo 4 se realiza un análisis del impacto de la localidad de los datos de una aplicación MapReduce iterativa en un entorno de nube híbrida. En este capítulo se abordan los distintos problemas que conlleva la migración inicial de datos desde una nube privada a una pública a través de un *enlace débil*, así como la planificación de tareas MapReduce en este tipo de despliegues. Para ello, se presenta un análisis comparativo de distintas estrategias de localidad de datos en cuanto a rendimiento en distintos escenarios. Además, también se presenta un modelo de coste económico que permite cuantificar dicha mejora no solo en términos de rendimiento, sino también en cuanto a coste económico.

El Capítulo 5 presenta una metodología para la predicción del tiempo de finalización de una aplicación MapReduce iterativa en un despliegue de *cloud bursting*. Esta metodología consta de varios pasos compuestos por un proceso de calibración de la nube híbrida, una caracterización de aplicaciones, y finalmente la aplicación de una expresión matemática que estime el tiempo de finalización de cada aplicación. Todo el proceso de calibración, caracterización de la aplicación y formulación matemática se describe con detalle en este capítulo. Además, se muestra la eficacia de esta metodología a través de su evaluación mediante distintos experimentos que abarcan los distintos tipos de aplicaciones MapReduce.

Para finalizar, en el Capítulo 6 se presentan las conclusiones principales de esta investigación. Además, se muestran todas las contribuciones que se han generado durante su transcurso. Posteriormente, se proponen una serie de líneas abiertas de investigación relacionadas con esta investigación.

CAPÍTULO 2

Entorno Tecnológico

En este capítulo se presentan las dos tecnologías principales que han sido utilizados durante el desarrollo de esta tesis: la computación en la nube y el modelo de programación MapReduce. La finalidad de esta presentación es proporcionar una visión completa y homogénea del entorno de desarrollo utilizado durante la investigación, haciendo hincapié en aquellos aspectos que más influencia tienen sobre el trabajo realizado.

En primer lugar se describe la computación en la nube, donde se presenta una breve introducción de sus modelos, características y tipos de nube, además de la descripción de OpenStack [33]. OpenStack es uno de los proyectos de software libre más extendidos para la creación de infraestructuras como servicio (IaaS, *Infrastructure as a Service*) de computación en la nube. Durante la investigación de esta tesis se ha utilizado OpenStack como base para la creación de nubes de tipo IaaS.

En segundo lugar se profundiza en uno de los modelos de programación por excelencia para el análisis de Big Data como es MapReduce [34], junto con su implementación de software libre Apache Hadoop [8].

Finalmente, se describe el diseño y configuración de todo el entorno experimental utilizado en esta tesis. Para ello, se describe el equipamiento utilizado y todo el detalle de la configuración realizada para el despliegue de una nube híbrida con OpenStack y Hadoop.

2.1 Computación en la nube

La computación en la nube o *cloud computing* es un modelo de computación que permite ofrecer prácticamente cualquier recurso de cómputo a través de una red, normalmente Internet. Se trata de una capa de abstracción de hardware y software que permite ofrecer dichos recursos como servicios bajo demanda a través de una infraestructura de red, con independencia de su ubicación, y ocultando todos los aspectos de mantenimiento o gestión. Esto permite la generación de infraestructuras adaptadas a cada necesidad, y la posibilidad de pago por uso.

Actualmente, la computación en la nube se ha adoptado en prácticamente todos los sectores de TIC (Tecnologías de la Información y la Comunicación) gracias a su elasticidad y flexibilidad de adaptación a prácticamente cualquier necesidad de cómputo en cada momento, y gracias al modelo de negocio de pago por uso, el cual permite reducir el coste total de propiedad o TCO (*Total Cost of Ownership*).

2.1.1 Modelos de servicio

La computación en la nube ofrece tres modelos de servicios distintos. La diferencia entre ellos consiste en el grado de control y personalización del servicio que ofrece al usuario. Los tres modelos son los siguientes:

Software como Servicio (SaaS). Corresponde a la capa de abstracción más alta y se caracteriza por ofrecer una aplicación completa, junto con todo su despliegue software y hardware, como un servicio bajo demanda y *multitenancy*, es decir, una sola instancia de la aplicación ejecutada en la infraestructura del proveedor permite el acceso a múltiples clientes. El proveedor de nube ofrece a los clientes todo lo necesario para utilizar su aplicación a través de una interfaz web o una aplicación nativa conectada a Internet. En este caso, el usuario no tiene ningún tipo de control sobre la aplicación; únicamente se le permite realizar algunos ajustes, pero a cambio puede despreocuparse del mantenimiento, actualizaciones, posibles errores, etc. Este modelo es el más común entre el consumidor medio, ya que interactúa con aplicaciones como Netflix [73], Spotify [36] o Google Apps [44].

Plataforma como Servicio (PaaS). La Plataforma como Servicio se ubica en una capa intermedia de abstracción. El proveedor de servicios de nube ofrece al cliente un entorno de desarrollo de aplicaciones, y el empaquetamiento de una serie de módulos o complementos que proporcionan, normalmente, una funcionalidad horizontal. En este caso, el cliente tiene mayor control del entorno sobre el que desea desarrollar, pero el proveedor se encarga del mantenimiento, asistencia y escalabilidad del entorno. El usuario no debe preocuparse de la infraestructura hardware que hay en la capa inferior, pudiendo centrarse únicamente en todo el ciclo de desarrollo de aplicaciones. Ejemplos de Plataforma como Servicio son Google App Engine [45] o Microsoft Azure [70].

Infraestructura como Servicio (IaaS). La Infraestructura como Servicio se encuentra en la capa de abstracción más baja en la que el proveedor de servicios ofrece directamente recursos de cómputo, almacenamiento o red al cliente a través de, normalmente, tecnologías de virtualización. En este caso, el cliente toma control de los recursos que se le ofrecen, la arquitectura del sistema, arquitectura de red, etc., pero debe ser él quien realice el mantenimiento y actualizaciones. Algunos de los ejemplos de Infraestructura como Servicio son Amazon Web Services [5] o SoftLayer de IBM [85].

2.1.2 Características

Según el Instituto Nacional de Estándares y Tecnología (NIST) de los EE.UU., un modelo de computación en la nube debe presentar cinco características esenciales [69] :

1. **Autoservicio bajo demanda.** El usuario puede acceder a los recursos de cómputo que le ofrezca su proveedor o proveedores de servicio en la nube a través de la red a medida que lo necesite, sin necesidad de interacción humana.

2. **Múltiples formas de acceso.** Los recursos que se ofrecen deben ser accesibles a través de la red por cualquiera de los mecanismos estándares utilizados, desde teléfonos móviles a ordenadores portátiles o PDAs.

3. **Compartición de recursos.** Los recursos de los proveedores (almacenamiento, memoria, ancho de banda, capacidad de procesamiento, máquinas virtuales, etc.) se agrupan para servir múltiples usuarios, de forma que éstos se van asignando y reasignando de forma dinámica según sus peticiones. Los usuarios pueden ignorar el origen y la ubicación de los recursos a los que acceden, aunque sí es posible que sean conscientes de su situación a determinado nivel, como el de centro de procesamiento de datos (CPD) o el de país.

4. **Elasticidad.** Los recursos se pueden asignar y liberar rápidamente, muchas veces de forma automática, de forma que permita al usuario tener la impresión de que los recursos a su alcance son ilimitados y están siempre disponibles.

5. **Servicio medido**. El proveedor debe ser capaz de medir, a determinado nivel, el servicio efectivamente entregado a cada usuario, de forma que tanto el proveedor como el usuario tengan un acceso transparente al consumo real de los recursos para posibilitar el pago por el uso efectivo de los servicios.

2.1.3 Tipos de Nube

Existen actualmente cuatro tipos de nube según el NIST para ofrecer distintos tipos de servicios de computación en la nube según las necesidades y posibilidades del usuario u organización. Cada tipo de nube corresponde con un tipo distinto de despliegue de la infraestructura de la nube dependiendo del grado de privacidad que se necesite. En la Figura 2.1 se muestra un esquema de los tipos de nube existentes.

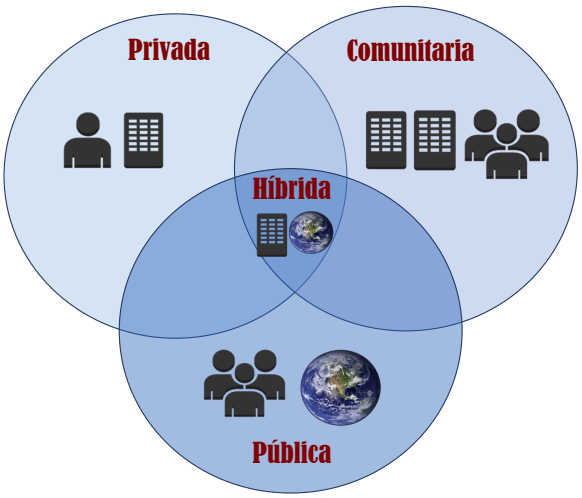


Figura 2.1: Computación en la nube. Tipos de nube.

Nube Privada. La infraestructura de esta nube pertenece a una única organización o compañía privada, la cual controla y gestiona todos los recursos que le ofrece a sus usuarios internos. De esta forma, posee un control total sobre la privacidad de los datos que manejan, así como los recursos y usuarios a los que se les permite el acceso. La organización o compañía es la única responsable del despliegue y administración de la nube, a través del uso de sus propios recursos de cómputo.

Nube Comunitaria. Este tipo de nube es muy similar a una nube privada pero con la diferencia de que la administración de toda la infraestructura de la nube es compartida y gestionada por varias organizaciones o compañías que controlan todos los recursos que ofrecen, así como los usuarios que tienen acceso. En este caso, las dos entidades deben suministrar conjuntamente los recursos de cómputo, así como negociar las políticas de prestación de servicio a sus usuarios.

Nube Pública. En una nube pública la infraestructura de la nube está disponible para el público en general a través de un proveedor de servicios de nube. Es decir, la administración y el control de esta nube pertenece a un tercero que ofrece sus recursos de cómputo o red al público en general, a través de un modelo de negocio de pago por uso. En este tipo de nubes tanto los datos como los procesos de cálculo pueden verse afectados por el uso de otros clientes, ya que el almacenamiento, la red y otras infraestructuras son normalmente compartidas por todos los clientes. En este caso, el acceso a los datos del cliente sólo puede realizarse de forma remota, normalmente a través de Internet.

Nube Híbrida. Una nube híbrida consta de la combinación de dos o más tipos de nube, normalmente del tipo pública y privada, que permanecen como entidades únicas pero que poseen la misma tecnología que les permite compartir datos y procesos. Una nube híbrida permite ampliar los límites de recursos que pueda poseer una nube privada al expandir sus recursos con recursos de una nube pública, dando a la entidad privada la posibilidad de llegar a un compromiso entre la privacidad de sus datos y la escalabilidad de los recursos.

2.1.4 Nube Híbrida: Cloud Bursting

Muchas organizaciones experimentan niveles variables de carga de trabajo con aplicaciones que suelen fluctuar durante su uso. La mayoría de empresas necesitan obtener altos rendimientos a costa de la escalabilidad de sus aplicaciones, pero para ello deben proporcionar recursos TI suficientes para superar picos de trabajo sin perjudicar su rendimiento. Sin embargo, resulta muy costoso e ineficiente proporcionar capacidad de infraestructura TI basándose únicamente en superar dichos picos de trabajo.

El *cloud bursting* [39] es un tipo de despliegue en forma de nube híbrida en el cual la nube privada extiende su capacidad de TI, de forma temporal, con recursos de una nube pública para superar las limitaciones físicas de una nube privada durante la duración de picos de trabajo. El *cloud bursting* permite romper las limitaciones físicas que puede tener una infraestructura de nube privada de una empresa, de forma que una aplicación que necesite un pico de recursos extra puede aprovechar los recursos adicionales disponibles para cumplir con la calidad de servicio demandante, de una forma flexible a través de un pago por uso.

2.2 OpenStack

OpenStack [33] es un proyecto de software libre, gestionado por la Fundación OpenStack, para el despliegue y administración de nubes de tipo infraestructura como servicio (IaaS).

OpenStack permite el control de grandes cantidades de recursos de cómputo, almacenamiento y red de un centro de datos, gestionados a través de una interfaz de control o *dashboard*, o vía la interfaz de programación de aplicaciones (API, *Application Programming Interface*) de OpenStack. Además, es capaz de funcionar con la mayoría de tecnologías de virtualización y software libre, haciéndolo ideal para su uso en infraestructuras heterogéneas.

Con más de 20 millones de líneas de código, y 66.685 desarrolladores en 181 países, OpenStack es una de las comunidades de desarrollo de software libre que más ha crecido en los últimos años. Iniciado por RackSpace y NASA, actualmente son más de 648 empresas las que están colaborando con el desarrollo y soporte de OpenStack bajo la supervisión de la Fundación OpenStack, entre las que destacan AT&T, Canonical, HP, IBM, Intel, RedHat, SUSE, Cisco, Nec o Huawei entre otras. Muchas compañías y organizaciones como CERN, Volkswagen, Snapdeal, China Mobile, AT&T, Comcast o Enter CloudSuite han adoptado el uso de este proyecto para sus necesidades de negocio.

OpenStack se implementa de forma modular a través de una serie de proyectos relacionados entre sí que controlan el procesamiento, almacenamiento y recursos de red, todo ello a través de un panel de control que permite a los administradores manejar todo su potencial, a la vez que los usuarios gestionan sus recursos.

Actualmente consta de más de 20 proyectos, entre los cuales destacan seis proyectos principales que conforman el núcleo, junto con el proyecto de interfaz gráfica de usuario.

En la Figura 2.2 se muestra la arquitectura modular de los servicios principales que ofrece OpenStack.

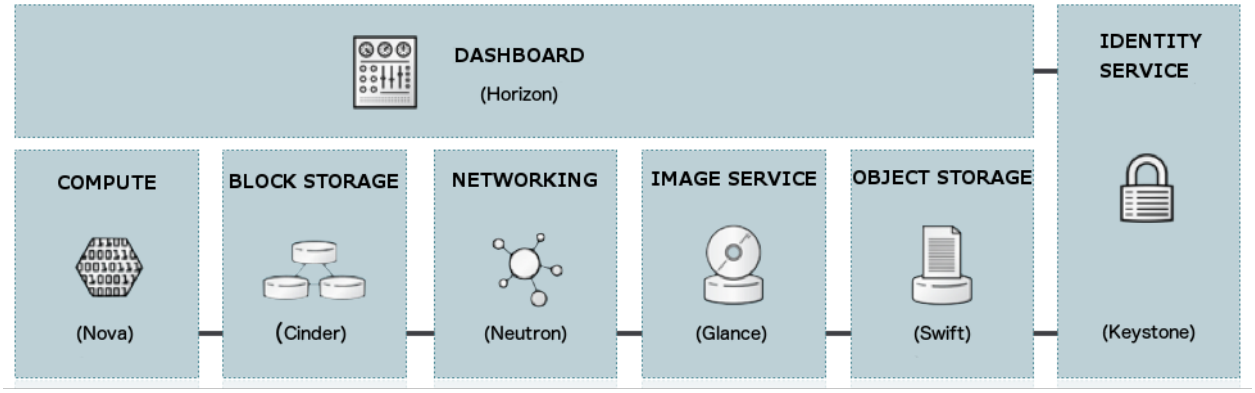


Figura 2.2: Arquitectura modular de los servicios principales de OpenStack. [1]

[1]Fuente: `https://docs.openstack.org/security-guide/introduction/introduction-to-openstack.html`

2.2.1 Nova

OpenStack Compute (Nova) es un controlador de recursos de cómputo y de la estructura de OpenStack, siendo la parte principal de un sistema de IaaS. Es el encargado de gestionar el ciclo de vida de las instancias de MV de un entorno OpenStack. Está diseñado para automatizar y gestionar los recursos de cómputo de la infraestructura de nube, soportando distintas tecnologías de virtualización como KVM, Xen, Hyper-V, VMware o incluso con la tecnología de contenedores Linux LXC. Entre sus funciones se incluye el despliegue, planificación y retirada de máquinas virtuales bajo demanda.

2.2.2 Cinder

OpenStack Block Storage (Cinder) es un servicio de almacenamiento de bloques para OpenStack. Está diseñado para proporcionar dispositivos de almacenamiento persistente a nivel de bloque para ser usado con las instancias de OpenStack Compute (Nova). Cinder utiliza la tecnología de LVM (*Logical Volume Manager*) u otros controladores para ofrecer el servicio. Se encarga de virtualizar la administración de los dispositivos de almacenamiento de bloque, y de proporcionar a los usuarios finales una API de autoservicio para solicitar y consumir esos recursos, sin requerir ningún conocimiento sobre dónde o en qué tipo de dispositivo se despliega su almacenamiento. Los volúmenes de almacenamiento de bloque se integran plenamente en otros servicios de OpenStack como Nova u Horizon, lo que permite a los usuarios de la nube gestionar sus propias necesidades de almacenamiento. Además del almacenamiento local de un servidor Linux, Cinder soporta el uso de otras plataformas de almacenamiento incluyendo Ceph, CloudByte, Coraid, EMC (VMAX y VNX), GlusterFS, Hitachi Data Systems, IBM Storage, Linux LIO, NetApp, Nexenta, Scality, SolidFire y HP.

2.2.3 Neutron

OpenStack Networking (Neutron) es un sistema de gestión de redes y direccionamiento IP. Proporciona Red como Servicio (NaaS, *Netwoking as a Service*) entre interfaces de red virtuales (vNICs) gestionadas por otros servicios de OpenStack (p. ej. Nova). Permite a los usuarios crear sus propias arquitecturas de red, controlar el tráfico, y conectar los servidores y dispositivos a una o más redes. Además, permite la extensión de servicios de red adicionales, como el balanceo de carga, cortafuegos o redes privadas virtuales (VPN). Es compatible con la mayoría de sistemas de virtualización de redes como Open vSwitch, Cisco UCS, Linux Bridge, Nicira NVP, Ryu OpenFlow Controller o Cloudbse-Hyper-V, entre otros.

2.2.4 Glance

OpenStack Image Service (Glance) es un proyecto que proporciona servicios y bibliotecas asociadas a almacenar, examinar, compartir, distribuir y gestionar imágenes de disco de arranque, además de otros datos estrechamente asociados con la inicialización de recursos de cómputo y definición de metadatos. También se puede utilizar para almacenar y catalogar un número ilimitado de copias de seguridad.

2.2.5 Swift

OpenStack Object Storage (Swift) es un sistema de almacenamiento de objetos escalable y tolerante a fallos. Ofrece software de almacenamiento en la nube para que los usuarios puedan almacenar y recuperar grandes cantidades de datos a través de su API. Está diseñado para ofrecer escalabilidad, durabilidad y redundancia de datos. Swift es ideal para almacenar datos no estructurados que puedan crecer sin límite.

2.2.6 Keystone

OpenStack Identity Service (Keystone) es el servicio de identidad utilizado por OpenStack para la autenticación (*authN*) y la autorización de alto nivel (*authZ*) de toda la infraestructura de la nube. Keystone es compatible con múltiples formas de autenticación. Recientemente se ha reestructurado para soportar servicios externos de proxy y mecanismos $AuthN/AuthZ$ como *oAuth*, *SAML* y *openID* en futuras versiones. Ofrece un directorio central de usuarios, asignados a los servicios de OpenStack a los que pueden acceder. Actúa como un sistema de autenticación común para todo el sistema operativo de la nube, y se puede integrar con servicios de directorio existentes como LDAP (*Lightweight Directory Access Protocol*).

2.2.7 Horizon

Horizon ofrece un panel de control o *dashboard* a través de una interfaz gráfica basada en web, tanto para usuarios como administradores de nube. A través de esta interfaz los administradores y usuarios pueden aprovisionar, gestionar, y monitorizar los recursos de la nube. Horizon se despliega comúnmente para ser utilizado por el público en general, por tanto implementa todas las características de seguridad habituales de los portales web.

2.2.8 Servicios Opcionales

Además de los servicios principales, la comunidad de OpenStack sigue desarrollando nuevos proyectos para ofrecer servicios adicionales que permitan enriquecer el potencial de esta tecnología. A continuación se muestran los proyectos opcionales que están actualmente disponibles:

Ceilometer. Telemetría. Monitor y medidor para facturación, benchmarking, escalabilidad y propósitos estadísticos.

Heat. Orquestación. Permite desplegar y gestionar múltiples aplicaciones o recursos de la nube usando el formato nativo HOT o el formato de AWS CloudFormation.

Trove. Base de datos. Base de datos como servicio que proporciona motores de bases de datos relacionales y no relacionales.

Sahara. MapReduce. Sahara tiene como objetivo proporcionar a los usuarios medios sencillos para proporcionar clústeres Hadoop especificando varios parámetros como la versión de Hadoop, topología del clúster, detalles de hardware de nodos y algunos parámetros más. Después de que un usuario rellene todos los parámetros, Sahara despliega el clúster en unos minutos. Sahara también

proporciona medios para escalar el clúster ya provisto añadiendo y eliminando nodos de trabajo bajo petición.

Ironic. Aprovisionamiento *Bare-metal* o de nodos físicos. Ironic tiene como objetivo el aprovisionamiento de máquinas físicas en lugar de máquinas virtuales.

Zaqar. Servicio de mensajería. Zaqar es un servicio de mensajería multiusuario para desarrolladores de web y telefonía móvil. Combina las ideas promovidas por el producto SQS (*Simple Queue Service*) de Amazon con semántica adicional para soportar la difusión de eventos.

Manila. Sistema de compartición de archivos. Proporciona un sistema de compartición de archivos para las instancias de OpenStack.

Designate. Servicio DNS. Proporciona DNS como servicio de forma integrada en el sistema de autenticación.

Barbican. Gestión de claves. Barbican proporciona una REST API diseñada para el almacenamiento seguro, el aprovisionamiento y la gestión de información privada como contraseñas, claves de cifrado y Certificados X.509.

Magnum. Contenedores. Es un servicio de la API de OpenStack para ofrecer contenedores como Docker y Kubernetes como recursos de primera clase.

Murano. Catálogo de aplicaciones. Ofrece un catálogo de aplicaciones para OpenStack, permitiendo a los desarrolladores de aplicaciones y a los administradores de la nube publicar aplicaciones listas para la nube en un catálogo navegable categorizado.

Congress. Sistema de gestión. Proporciona políticas como un servicio a través de cualquier colección de servicios en la nube, con el fin de ofrecer gobernabilidad y elasticidad a infraestructuras dinámicas.

2.3 Análisis de Big Data

Vivimos en un mundo impulsado por los datos. Las personas generamos y almacenamos información constantemente de forma cada vez más abundante. Esta contribución a la acumulación masiva de datos la podemos encontrar en diversos sectores de la industria, como por ejemplo en empresas que almacenan grandes cantidades de datos transaccionales para reunir información acerca de sus clientes, proveedores, ventas, etc. En el sector público, por ejemplo, se almacenan enormes bases de datos sobre censo de población, registros médicos, impuestos, etc. Si analizamos los datos que se generan a través de las redes sociales podemos encontrarnos con más de 12 Terabytes de *tweets* creados diariamente en Twitter o el almacén de 100 Petabytes de información de vídeos y fotos de Facebook [15]. Por otra parte, todas las actividades que realizamos con nuestros teléfonos móviles, ubicación geográfica mediante coordenadas GPS, correo electrónico, etc., son también almacenadas y analizadas para ofrecernos publicidad cada vez más personalizada.

El término Big Data hace referencia al almacenamiento de grandes volúmenes de datos de una gran variedad (estructurados y no estructurados), para su posterior procesamiento y análisis a gran velocidad con el fin de obtener información de valor intrínseco. Hoy en día el Big Data se ha convertido en un elemento esencial para la mayoría de empresas, ya que su análisis les permite obtener información muy valiosa para la toma de decisiones y movimientos estratégicos de negocio.

El Big Data se define generalmente por tres dimensiones (3 Vs) [64]:

- **Volumen.** El volumen de datos se puede medir por la cantidad de transacciones, eventos, o cantidad de historial que se produce. Las organizaciones recopilan datos de una variedad de fuentes, incluyendo transacciones comerciales, redes sociales, e información de sensores, produciendo grandes volúmenes de datos.
- **Velocidad.** El término velocidad se refiere a la velocidad a la que se crean, acumulan, inician y procesan los datos. El incesante ritmo de crecimiento de las empresas demanda el procesamiento de los datos en prácticamente tiempo real para la toma de decisiones de negocio. Esto requiere un tratamiento de datos de alta velocidad que permita extraer la información necesaria para tomar decisiones en tiempo real o en velocidades muy próximas al tiempo real.
- **Variedad.** Los datos son recopilados en todo tipo de formatos: desde datos estructurados y numéricos en bases de datos tradicionales, hasta documentos de texto no estructurados, correo electrónico, vídeo, audio, datos de cotización y transacciones financieras.

El análisis de Big Data requiere de modelos de programación que se adapten a las necesidades que implican las 3 Vs definidas anteriormente. Todos los modelos de programación diseñados para el análisis de Big Data se basan necesariamente en técnicas de paralelismo masivo, adoptando distintas estrategias dependiendo de las necesidades de procesamiento. Entre ellas se pueden destacar los modelos DAG (*Directed Acyclic Graph*), BSP (*Bulk Synchronous Parallel*), Stream processing, y uno de los más utilizados, MapReduce [40].

Una de las implementaciones más populares de MapReduce es Hadoop, ya que permite procesar conjuntos de datos de Big Data de gran tamaño, llegando a alcanzar los Petabytes de datos. Actualmente han surgido nuevos proyectos alrededor de Hadoop para dar soluciones más eficientes a los distintos tipos de conjuntos de datos como *Avro*, *Parquet*, *Flume*, *Sqoop*, *Pig*, *Crunch*, *Spark*, *HBase* o *ZooKeeper*. En esta investigación se ha utilizado Hadoop MapReduce por su gran popularidad y adopción en todo el entorno empresarial [98].

2.3.1 MapReduce

MapReduce [34] es un modelo de programación desarrollado por Google para procesar y generar grandes volúmenes de datos de forma paralela. Los usuarios especifican una función de mapeado o *Map*, que procesa un tipo de datos de clave/valor para generar un conjunto de datos intermedio de tipo clave/valor, y una función de reducción o *Reduce*, que combina todos los valores intermedios asociados con la misma clave intermedia para obtener un resultado. Los programas escritos en este estilo funcional se paralelizan y ejecutan automáticamente de forma distribuida en un clúster.

MapReduce ha sido ampliamente adoptado gracias a su implementación de software libre Apache Hadoop.

Función Map()

La función de mapeado o *Map* es una función escrita por el usuario que toma como entrada una lista de pares clave/valor, y produce una salida de un conjunto de datos intermedios de tipo

clave/valor. La biblioteca de MapReduce se encarga de asociar todos los valores intermedios producidos y agruparlos para cada clave I, para posteriormente enviarlos a la función de reducción o *Reduce*.

$$Map(k_1,v_1) \rightarrow list(k_2, v_2)$$

Cuando se inicia un proceso MapReduce, el conjunto de datos de entrada es dividido, en tantas partes como especifique el usuario, en listas clave/valor (k_1,v_1), para que el conjunto de tareas *Map* pueda procesar en paralelo los datos, aplicando la función *Map* a cada una de la divisiones para producir un conjunto de listas de datos intermedios de clave/valor (k_2,v_2).

Función Reduce()

La función de reducción o *Reduce* también es implementada por el usuario para indicar qué operaciones se deben realizar con las listas de pares clave/valor intermedias producidas por la fase *Map*. Cada función *Reduce* acepta una o varias claves I y un conjunto de valores de dicha clave $(k_2, list(v_2))$. Estos datos son combinados en cada *Reduce*, y se les aplica la función escrita por el usuario para producir un resultado final (v_3).

$$Reduce(k_2, list(v_2)) \rightarrow list(v_3)$$

Flujo de trabajo

A continuación se muestra el flujo de trabajo que utiliza MapReduce para procesar un conjunto de datos de entrada:

1. La primera fase se compone de una serie de tareas *Map*. Cada una de estas tareas se encarga de gestionar (tratar) una de las partes en las que se ha dividido el conjunto de datos de entrada, a través de la ejecución de la función *Map* implementada por el usuario.
2. Los datos intermedios obtenidos en la fase *Map* son distribuidos según su clave a las distintas tareas *Reduce* configuradas. Este proceso de transferencia de datos se denomina *Shuffle* y forma parte de la tarea *Reduce*.
3. Una vez distribuidos los datos, éstos son ordenados en una fase denominada *Sort* antes de ser procesados por la función *Reduce*.
4. Finalmente, cada tarea *Reduce* ejecuta su función *Reduce* implementada por el usuario para producir un resultado final.

El modelo de programación MapReduce implica la ejecución en paralelo de todas las fases, respetando siempre las dependencias de datos que puedan existir entre ellas. En concreto, la fase *Reduce* no puede empezar hasta que se complete la fase *Map*, ya que la entrada de datos de la fase *Reduce* requiere de todos los datos de salida de la fase *Map*.

En la Figura 2.3 se muestra el flujo de trabajo que produciría un ejemplo de un contador de palabras implementado en MapReduce. En esta figura se muestra cada una de las fases descritas anteriormente, así como un ejemplo de salida de los datos producidos en cada fase.

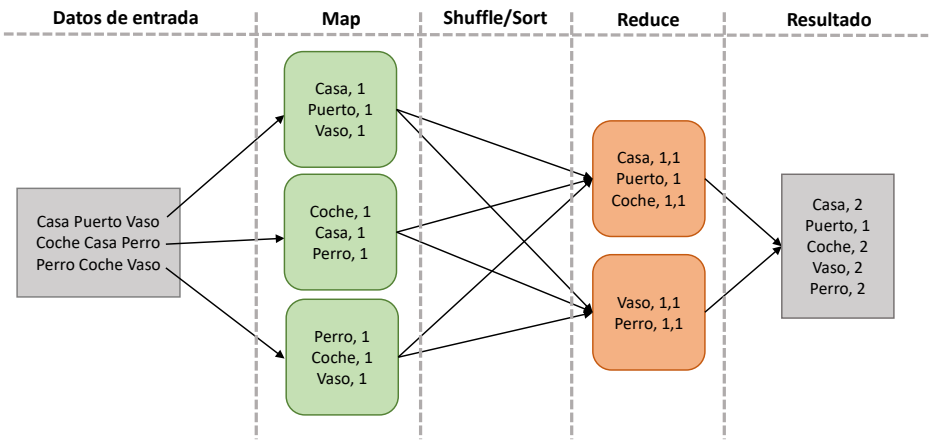


Figura 2.3: Ejemplo de flujo de trabajo de MapReduce. Aplicación WordCount.

2.3.2 Hadoop

Hadoop [8] es un proyecto de software libre, gestionado inicialmente por Yahoo y actualmente por el proyecto Apache, que permite procesar grandes volúmenes de datos de forma distribuida, fiable y escalable. Permite a las aplicaciones trabajar con miles de nodos y Petabytes de datos. Actualmente, Hadoop está siendo construido y usado por una comunidad global de contribuyentes [10], a través del lenguaje de programación Java. Yahoo! es uno de sus mayores contribuyentes y usa Hadoop en todo su modelo de negocio.

La biblioteca de software de Apache Hadoop es un *framework* que permite el procesamiento distribuido de grandes conjuntos de datos a través de clústeres de computadores y del uso de modelos de programación sencillos. Hadoop está diseñado para poder escalar desde servidores individuales a miles de nodos, cada uno de ellos ofreciendo capacidad de cómputo y almacenamiento local. Proporciona alta disponibilidad a través de su diseño de detección de fallos en la capa de aplicación en lugar de confiar en el hardware subyacente, por lo que ofrece un servicio altamente disponible a través de una capa lógica superior al clúster de computadores, los cuales pueden ser propensos a fallos.

Hadoop implementa el modelo de programación MapReduce basándose en la investigación que publicó Google sobre MapReduce y Google File System (GFS) en 2004 [34].

Hoy en día, Hadoop es mundialmente utilizado por innumerables empresas para ejecutar cómputos sobre enormes cantidades de datos de forma distribuida a través de miles de nodos. Algunos de estos ejemplos son A9.com, eBay, Facebook, Fox Interactive Media, IBM, ImageShack, Last.fm, LinkedIn, Twitter, MercadoLibre, The New York Times, Rackspace o 1&1, entre otros.

El proyecto está implementado en varios módulos principales [8]:

- **Hadoop Common:** son las utilidades más comunes que utilizan el resto de módulos de Hadoop.
- **Hadoop Distributed File System (HDFS):** es un sistema distribuido de ficheros que proporciona acceso de alto rendimiento a los datos de aplicación.

- **Hadoop YARN:** es un *framework* de planificación de tareas y gestión de recursos de un clúster.
- **Hadoop MapReduce:** es la implementación del modelo de programación MapReduce para el procesamiento paralelo de grandes conjuntos de datos basado en YARN.

Otros proyectos relacionados con Hadoop en Apache incluyen:

- **Ambari**: es una herramienta basada en web para el aprovisionamiento, gestión y monitorización de clústeres Apache Hadoop que incluye soporte para Hadoop HDFS, Hadoop MapReduce, Hive, HCatalog, HBase, ZooKeeper, Oozie, Pig y Sqoop.
- **Avro**: es un sistema de serialización de datos.
- **Cassandra**: es una base de datos multi-master escalable sin puntos de fallo único.
- **Chukwa**: es un sistema de recopilación de datos para la gestión de grandes sistemas distribuidos.
- **HBase**: es una base de datos distribuida y escalable que admite el almacenamiento estructurado de datos de grandes tablas.
- **Hive**: es una infraestructura de datos *warehouse* que proporciona resúmenes de datos y consultas *ad hoc*.
- **Mahout**: es una biblioteca escalable de aprendizaje automático y de minería de datos.
- **Pig**: es un lenguaje de alto nivel de flujo de datos y un *framework* de ejecución para el cálculo paralelo.
- **Spark**: es un motor rápido de cálculo general para datos Hadoop. Spark proporciona un modelo de programación simple y expresivo que soporta una amplia gama de aplicaciones, incluyendo ETL, aprendizaje automático, procesamiento de flujos y computación gráfica.
- **Tez**: es un *framework* de programación de flujo de datos generalizado, basado en Hadoop YARN, que proporciona un motor potente y flexible para ejecutar un DAG (*Directed Acyclic Graph*) arbitrario de tareas para procesar datos, tanto para casos de uso discontinuos como interactivos. Tez está siendo adoptado por Hive, Pig y otros *frameworks* en el ecosistema de Hadoop, y también por otro software comercial (por ejemplo, herramientas ETL (*Extract, Transform and Load*)), para reemplazar a Hadoop MapReduce como motor de ejecución subyacente.
- **ZooKeeper**: es un servicio de coordinación de alto rendimiento para aplicaciones distribuidas.

Hadoop Distributed File System (HDFS)

Hadoop Distributed File System (HDFS) [9] es el sistema de ficheros distribuidos principal de Hadoop. HDFS tiene muchas similitudes con los sistemas de ficheros distribuidos existentes, sin embargo sus diferencias son significativas. Es altamente tolerante a fallos y está diseñado para ser desplegado en hardware de bajas prestaciones. HDFS proporciona un alto rendimiento de acceso a

los datos de aplicación, y es adecuado para aplicaciones que requieren grandes volúmenes de datos. Originalmente fue construido como infraestructura para el motor de búsqueda web Apache Nutch. Actualmente HDFS es parte del núcleo del proyecto Apache Hadoop.

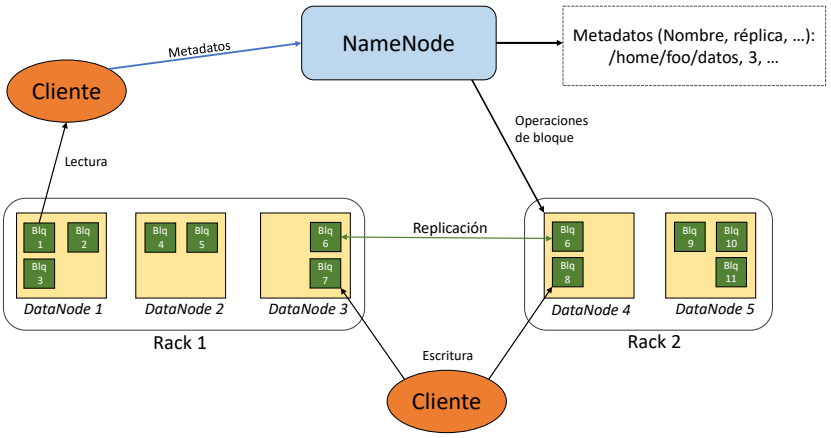


Figura 2.4: Arquitectura de HDFS.

HDFS posee una arquitectura maestro/esclavo. Un clúster HDFS consta de un único *NameNode*: servidor maestro que gestiona el espacio de nombres del sistema de archivos y regula el acceso a los archivos por parte de los clientes. Además, hay un número de *DataNodes*, por lo general uno por nodo en el clúster, que gestionan el almacenamiento adjunto a los nodos en los que se ejecutan. HDFS expone un espacio de nombres del sistema de ficheros. Internamente, un archivo se divide en uno o más bloques, de un tamaño prefijado por el usuario, los cuales se almacenan en un conjunto de *DataNodes*. El *NameNode* se encarga de ejecutar todas las operaciones del espacio de nombres del sistema de ficheros como abrir, cerrar y renombrar archivos y directorios. También determina la asignación de bloques a *DataNodes*. Los *DataNodes* son responsables de servir las solicitudes de lectura y escritura de los clientes del sistema de ficheros. Además también realizan la creación, eliminación y replicación de bloques a partir de las instrucciones del *NameNode*. En la Figura 2.4 se muestra la arquitectura de HDFS.

HDFS está diseñado para ofrecer tolerancia a fallos a través de la replicación de todos sus bloques entre los nodos. Tanto el tamaño de bloque como el factor de replicación son configurables por el usuario según sus necesidades. La colocación de estas réplicas es fundamental para la fiabilidad y el rendimiento de HDFS, y es lo que le distingue de la mayoría de sistemas de ficheros distribuidos. HDFS presupone su ejecución sobre un clúster de computadores compuesto por varios racks, con múltiples nodos por rack. Por defecto, HDFS genera tres réplicas de cada bloque, asegurando que cada de ellas se encuentra en un nodo diferente, y al menos en dos racks diferentes. De esta forma garantiza fiabilidad y disponibilidad de datos ante fallos del sistema, y además permite reducir la comunicación inter-rack que puede afectar al rendimiento de E/S de datos.

YARN

YARN (Yet Another Resource Negotiator) [11] es esencialmente un sistema para administrar aplicaciones distribuidas. YARN está compuesto por un componente central llamado *ResourceManager*, que arbitra todos los recursos disponibles del clúster, y un *NodeManager* por nodo que recibe

y toma las decisiones recibidas por el *ResourceManager*, el cual es el responsable de gestionar los recursos disponibles del nodo donde se ejecuta. Además de estos dos componentes principales, YARN lanza un tercer componente denominado *ApplicationMaster* para cada una de la aplicaciones que ejecuta, el cual es responsable de negociar los recursos que necesita su aplicación y monitorizar su estado.

A continuación se describe con más detalle cada uno de los componentes de la arquitectura de YARN:

- **ResourceManager:** en YARN, el *ResourceManager* es, principalmente, un planificador puro. En esencia, se limita estrictamente a arbitrar los recursos disponibles en el sistema entre las aplicaciones. Optimiza el uso del clúster (mantiene todos los recursos en uso todo el tiempo) para garantizar calidad de servicio y cumplir con restricciones de capacidad, imparcialidad y SLA. Para permitir diferentes políticas de restricción, el *ResourceManager* tiene un planificador configurable que permite utilizar diferentes políticas según las necesidades.
- **ApplicationMaster:** se instancia para cada una de la aplicaciones que se ejecutan en un clúster Hadoop. Se encarga de gestionar los recursos que necesita cada aplicación con el *ResourceManager* y de trabajar con el componente *NodeManager* para ejecutar y monitorizar la aplicación. Este componente ofrece escalabilidad al sistema, ya que permite delegar parte de las funciones del *ResourceManager* de forma distribuida por los nodos del clúster, para evitar así cuellos de botella. De esta forma, el *ResourceManager* solo debe responder a las peticiones de recursos que le solicita cada *ApplicationMaster*, pero no debe planificar ni gestionar todo su proceso de ejecución en los diferentes nodos.
- **NodeManager:** dentro de la arquitectura maestro/esclavo de YARN este componente se enmarca como el esclavo. Se trata de un gestor a nivel de nodo que controla y determina los recursos disponibles de un nodo, además de monitorizar su estado e informar al *ResourceManager* de dicha gestión cada cierta frecuencia. El *NodeManager* se encarga de recibir y ejecutar todas la peticiones que recibe del *ResourceManager*.
- **ResourceRequest y Containers:** YARN está diseñado para permitir que aplicaciones individuales utilicen recursos de un clúster de una manera compartida, segura y multi-usuario (a través del *ApplicationMaster*). Además, es consciente de la topología del clúster con el fin de planificar y optimizar eficientemente el acceso a los datos. Para cumplir con estos objetivos, el planificador central (en el *ResourceManager*) posee toda la información sobre las necesidades de recursos de una aplicación, lo que le permite tomar mejores decisiones de planificación en todas las aplicaciones del clúster. Básicamente, una aplicación realiza una petición de recursos o *ResourceRequest* a través del *ApplicationMaster* para satisfacer sus necesidades, y el planificador principal le responde con uno o varios contenedores o *Containers*. Un *Container* contiene la localización del nodo en el cual se le permite ejecutar su solicitud y los permisos necesarios para hacer uso de sus recursos físicos (CPU, memoria, disco, etc). En la ejecución de una aplicación MapReduce un contenedor puede albergar o bien una tarea *Map* o bien una *Reduce*.

En YARN, la unidad lógica de datos que utiliza cada contenedor en su lectura y escritura de datos se llama *Split*. Un *Split* suele corresponder con la unidad física de datos de HDFS, que es un bloque de datos. Sin embargo, puede ocurrir que para completar una unidad lógica (*Split*) sea necesaria la lectura de un bloque completo de datos más una pequeña parte de otro o más bloques

de datos (HDFS realiza particiones fijas de los datos). Esto se explica por el hecho de que la unidad lógica realiza su partición de datos teniendo en cuenta el contenido de la información (por ejemplo, líneas de texto). Por tanto, en caso de que esta información quede incompleta a la hora de leer un bloque completo (una línea o palabra quede incompleta), el contenedor deberá leer pequeñas partes de otros bloques para completar su información y procesar correctamente dicho *Split*.

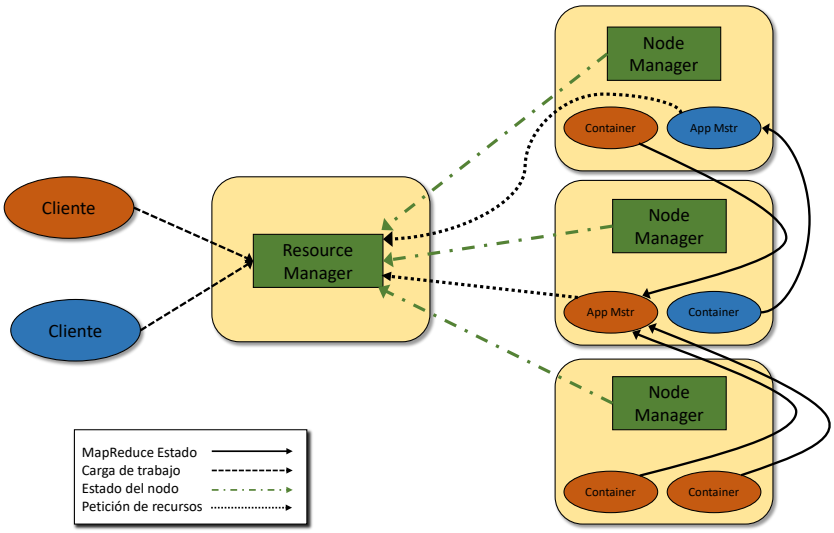


Figura 2.5: Arquitectura y flujo de trabajo de YARN.

En la Figura 2.5 se muestra la arquitectura y el flujo de trabajo de YARN para un ejemplo en el que se atienden dos clientes. En este ejemplo se muestran cuatro nodos físicos, de los cuales uno de ellos ejerce la función de maestro a través de la ejecución del *ResourceManager*, y los otros tres la función de esclavos a través de la ejecución del componente *NodeManager*. Cada tipo de flecha que aparece representa un flujo distinto de datos descrito en la leyenda de la figura. A continuación se muestran los pasos que sigue el flujo de trabajo en este ejemplo:

1. Inicialmente, ambos clientes realizan su petición al *ResourceManager*, que es el encargado de generar un *ApplicationMaster* para cada aplicación.

2. Cada *ApplicationMaster* chequea los recursos que necesita su aplicación y solicita dichos recursos al *ResourceManager*.

3. Cada *NodeManager* informa al *ResourceManager* de su disponibilidad de recursos.

4. El *ResourceManager* procesa las solicitudes de cada *ApplicationMaster* y realiza la planificación de recursos en los nodos disponibles.

5. Cada *ApplicationMaster* genera una cola de *Containers* necesarios para procesar los datos de entrada, y lanza cada *Container* en el nodo planificado.

6. Cada *Container* ejecuta su tarea e informa al *ApplicationMaster* sobre su estado.

7. El proceso se repite desde el paso tres hasta que se ejecutan todos los *Containers* requeridos.

2.4 Entorno de experimentación

En esta sección se presenta el entorno de experimentación utilizado a lo largo de toda la investigación. Para ello, se detalla la configuración y el despliegue de cada tecnología empleada.

La realización de los experimentos de esta tesis ha requerido el despliegue de las dos tecnologías anteriormente descritas, además de toda la configuración hardware implícita. Por una parte, se ha desplegado un entorno de computación en la nube a través de la creación de dos nubes privadas conectadas por una red de comunicación configurable entre ambas partes, simulando un entorno de nube híbrida lo más próximo posible a la realidad. Por otra parte, sobre dicho entorno de nube híbrida se ha desplegado el *framework* Hadoop para la ejecución de aplicaciones MapReduce, haciendo uso de las máquinas virtuales (MVs) subyacentes.

La descripción del entorno de experimentación se ha dividido en tres partes principales. Primero se describe la capa física en la que se presenta todo el equipamiento hardware utilizado, así como la visión general de la arquitectura del sistema. Posteriormente se detalla la configuración realizada en la primera capa lógica, en la que se realiza el despliegue de dos nubes privadas para la formación de una nube híbrida a través de OpenStack. Finalmente, se describe la configuración realizada en la segunda capa lógica, en la que se detalla el despliegue del *framework* Hadoop MapReduce sobre la nube híbrida desplegada, tal y como se muestra en la Figura 2.6.

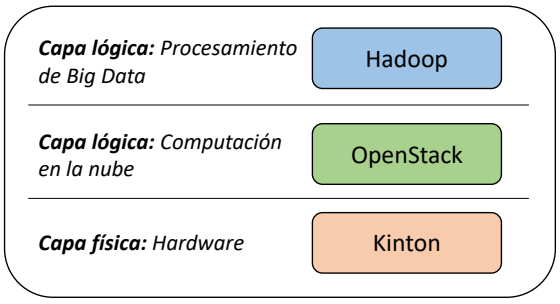


Figura 2.6: Capas del entorno de experimentación.

2.4.1 Equipamiento

Todos los experimentos realizados en esta tesis han utilizado el mismo entorno de experimentación. Para ello, se ha hecho uso del clúster Kinton del grupo HPC&A de la Universidad Jaume I.

Este clúster está formado por un total de 9 nodos de cómputo divididos en dos grupos según sus características. El primer grupo consta de un total de 5 nodos de menos capacidad, que hemos denominado *thin*, dedicados a tareas de gestión. El segundo grupo consta de 4 nodos más potentes, que hemos denominado *fat*, con capacidad y potencia suficiente para albergar varias máquinas virtuales al mismo tiempo. Todos los nodos del clúster están conectados a dos conmutadores de red de tipo Gigabit de 24 puertos. En las Tablas 2.1 y 2.2 se muestran las características técnicas del grupo *thin* y *fat*, respectivamente.

Nodos *thin*	
Fabricante	HP
Modelo	Proliant DL120 G6
Procesador	Intel Xeon X3430 (4 núcleos a 2,67 GHz)
Memoria	4 GB
Tarjeta de red	2 Fast Ethernet (100 Mbps) y 1 Gigabit Ethernet (1 Gbps)
Disco duro	500 GB 7200 rpm

Tabla 2.1: Características técnicas de los nodos *thin*.

Nodos *fat*	
Fabricante	Supermicro
Modelo	Twin 2U
Procesador	Dual Xeon E5-2630v3 (2×8 núcleos a 2,4 GHz)
Memoria	64 GB DDR4 2133 MHz
Tarjeta de red	2 Gigabit Ethernet (1 Gbps)
Disco duro	480 GB SSD

Tabla 2.2: Características técnicas de los nodos *fat*.

El sistema operativo (SO) que utiliza Kinton es la distribución de Linux Rocks 6.1.1, un SO basado en CentOS para el despliegue y gestión de clústeres de computadores. En este caso, uno de los nodos *thin* se ha utilizado como *frontend* del clúster para acceder al resto de nodos, quedando por tanto 4 nodos *thin* disponibles para cualquier propósito requerido.

2.4.2 Arquitectura

En la Figura 2.7 se muestra la arquitectura general del entorno de experimentación. En ella se detalla el despliegue tanto de la capa física, como de las dos capas lógicas.

Como se puede observar en la Figura 2.7, cada nodo físico se ha designado para ejercer un rol dentro del despliegue de OpenStack, donde 2 nodos *thin* y 1 *fat* se han designado para la creación de una primera nube privada a la que denominamos *on-premise*, y los otros 2 nodos *thin* y 3 *fat* para la formación de una segunda nube privada que simula una nube pública a la que denominamos *off-premise*. Esta nomenclatura se utilizará para el resto de la tesis.

Cada despliegue de OpenStack para la formación de una nube privada requiere de una arquitectura en la que exista al menos un nodo de control de la nube (*Controller*), un nodo para la gestión de la red de las MVs (*Network*) y uno o múltiples nodos de cómputo (*Compute*) para albergar las MVs. Dado que las tareas de gestión de la nube y de la red no requieren de nodos de gran capacidad, se han utilizado los nodos *thin* para dicho propósito en cada una de las nubes, y el resto de nodos *fat* de más capacidad para albergar las MVs.

OpenStack también requiere de una configuración de red específica (que se explica con más detalle en la siguiente sección) para separar el tráfico de red de cada servicio. Por tanto, en todos

los nodos de cada nube se ha conectado su interfaz de red *NIC0* al correspondiente conmutador de 1 Gbps, formando una subred que OpenStack utiliza para administración. Posteriormente, en cada nodo *Compute* y *Network* se ha utilizado un segundo interfaz de red *NIC1* formando una segunda subred dedicado a aislar el tráfico interno de las MVs de cada nube. Finalmente, se ha conectado una tercera interfaz de red *NIC2* de los nodos *Network* a su correspondiente conmutador de 1 Gbps para crear una subred que OpenStack utiliza para redirigir todo el tráfico externo de las MVs. En este último caso, OpenStack crea un enrutador virtual en cada nube para gestionar dicho tráfico, el cual está físicamente mapeado a un puerto del nodo *Network* para ofrecer una comunicación entre nubes a través de este enlace. En la nube *on-premise* se han desplegado un total de 4 MVs, y 12 MVs en la nube *off-premise*.

Sobre el despliegue de MVs *on-premise* se ha instalado el *framework* Hadoop MapReduce. Hadoop requiere de una arquitectura maestro/esclavo, por lo que una de las MVs *on-premise* se ha utilizado como maestro y las tres restantes como esclavos. Este despliegue de Hadoop se ha complementado con 12 MVs *off-premise*, configuradas como esclavos, como parte del despliegue de Hadoop *on-premise*. De esta forma, cualquier aplicación MapReduce ejecutada en este entorno utiliza tanto las MVs *on-premise* como *off-premise*, lo que da lugar a una nube híbrida.

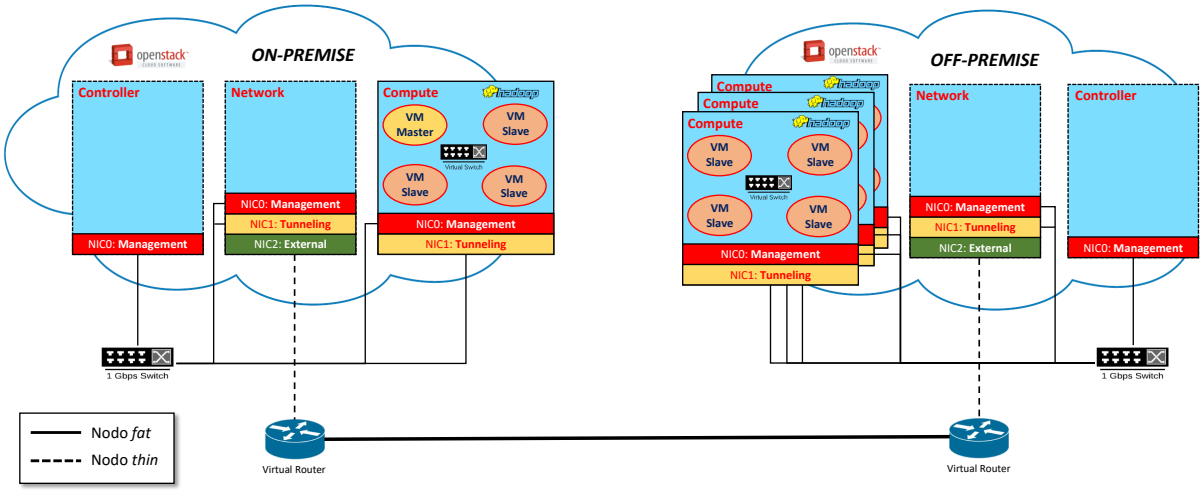


Figura 2.7: Arquitectura general del entorno de experimentación.

2.4.3 Despliegue de OpenStack

Tal y como se ha descrito en la sección anterior, una arquitectura básica de OpenStack está formada por al menos un nodo de control o *Controller*, que se encarga de gestionar y orquestar todos los servicios de una nube de OpenStack; un nodo de red o *Network*, que gestiona toda la topología de red virtual creada en OpenStack para las MVs y otros servicios de red; y uno o múltiples nodos de cómputo o *Compute*, que permiten el despliegue de MVs a través de un hipervisor.

En este caso, se ha instalado la versión Icehouse de OpenStack junto con el hipervisor de máquinas virtuales QEMU/KVM 0.12.1. En cada nodo de control se han instalado los servicios de gestión de MVs (Nova), red (Neutron), autenticación (Keystone) y gestión de imágenes (Glance). En cada nodo de red se ha instalado el servicio principal de red, Neutron. Finalmente, en los nodos

de cómputo se han instalado los servicios de gestión de MVs y de red, Nova y Neutron, para la creación de MVs y configuración de su red virtual. En la Figura 2.8 se muestran los servicios de OpenStack utilizados en cada caso.

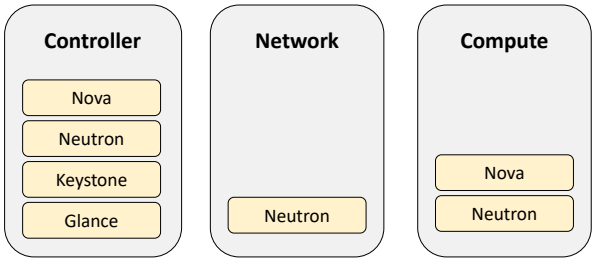


Figura 2.8: Servicios de OpenStack utilizados para cada tipo de nodo.

En una configuración típica basada en *Neutron* (el servicio de gestión de red OpenStack estándar), hay tres dominios de comunicación separados conceptualmente: la red de gestión o *Management* (es decir, utilizada para mensajes de control y tráfico administrativo), la red interna o *Tunneling* (es decir, el tráfico entre las instancias de MVs, que utilizan direcciones IP privadas), y la red externa o *External* (es decir, el tráfico entre las instancias de MV y el exterior de la nube). Como se muestra en la Figura 2.7 de la arquitectura general, en cada nodo de cómputo se crea un conmutador virtual, gestionado por Neutron y mapeado a la interfaz física *NIC1* de *Tunneling*, que permite comunicar todas las MVs de una nube a través de una subred virtual independiente. Sin embargo, toda la comunicación con el exterior de la nube se encamina a través del nodo de red, utilizando un enrutador virtual creado con este fin y mapeado a la interfaz física de red *NIC2*. Por lo tanto, en una configuración de nube híbrida que involucre dos despliegues de OpenStack, todo el tráfico producido entre MVs *on-premise* y *off-premise* debe pasar a través de la red externa (interfaz de red *NIC2*) de cada nodo de red, lo que le convierte en un *enlace débil*, ya debe soportar el rendimiento agregado de todas las MVs *on-premise* y *off-premise*.

Para nuestros experimentos se ha creado un nuevo tipo de MV similar a la instancia *m4.xlarge* de Amazon AWS para simular un escenario más próximo a la realidad, con 4 CPUs virtuales (o vCPUs), almacenamiento local HDD de 100 GB y 16 GB de RAM. Con estas características, cada nodo de cómputo tiene la capacidad de alojar 4 MV simultáneamente. Dado que dos MVs ubicadas conjuntamente pueden comunicarse mucho más rápido que dos MV remotas (comunicación *loopback*), se ha limitado el ancho de banda de red de cada MV a un 1 Gbps para obtener un entorno uniforme, donde todas las máquinas virtuales puedan comunicarse entre sí a la misma velocidad, sin importar dónde se encuentren ubicadas. Además, dado que el disco duro es compartido por 4 MVs en cada nodo físico, se ha limitado también el rendimiento del disco de cada MV para obtener los mismos rendimientos de acceso a disco, independientemente del número de MVs desplegadas en cada caso.

En cada MV se ha instalado el sistema operativo Ubuntu 14.04 LTS Server y se ha generado una clave de seguridad RSA, donde cada clave pública se ha distribuido por el resto de MVs para que la comunicación SSH entre MVs no requiera de autenticación a través de usuario y contraseña.

Por último, la red externa que conecta las dos nubes y que está gestionada por ambos nodos de red se ha configurado para poder limitar su capacidad a 1 Gbps o a 100 Mbps, de forma que permita realizar distintos experimentos en los que se simule la contratación de un proveedor de servicios de Internet (ISP) de alta velocidad u otro de una velocidad más estándar.

2.4.4 Despliegue de Hadoop

Sobre la configuración de dos nubes OpenStack en el clúster Kinton se ha desplegado el *framework* Hadoop MapReduce, versión 2.6.0, en todas las MVs. La instalación de Hadoop consta de cuatro módulos principales que se han instalado en cada MV: el núcleo del sistema Hadoop, el módulo de almacenamiento distribuido HDFS, el módulo YARN de planificación y ejecución de tareas, y el módulo que implementa el modelo de programación MapReduce. Cada uno de estos módulos posee su propia configuración, en la cual se pueden parametrizar cientos de aspectos a través de los siguientes ficheros:

- `core-site.xml`: fichero de configuración del núcleo de Hadoop.
- `hdfs-site.xml`: fichero de configuración de HDFS.
- `yarn-site.xml`: fichero de configuración de YARN.
- `mapred-site.xml`: fichero de configuración MapReduce.

La mayoría de estos parámetros tiene un valor por defecto para agilizar el proceso de configuración. Sin embargo, hay ciertos parámetros que dependen tanto de la arquitectura sobre la que se instala Hadoop, como del tipo de aplicación que se desee optimizar, y éstos deben configurarse manualmente para obtener el máximo rendimiento. En nuestro caso, estos parámetros se han configurado para que cada una de las MVs tenga disponible 4 slots (espacio de memoria para contenedores de tareas *Map* o *Reduce*) para tareas *Map*, o 2 slots para tareas *Reduce*. A continuación se detallan algunos de los parámetros principales configurados en cada fichero de configuración:

Parámetros del núcleo de Hadoop:

En el núcleo de Hadoop es conveniente cambiar algunos parámetros como el *buffer* de E/S por defecto para obtener un mejor rendimiento. Es recomendable utilizar un valor entre 64 y 128 KB (Kilobytes). Además, en este fichero es donde se especifica el tipo de sistema de almacenamiento que se desea utilizar, en nuestro caso HDFS.

A continuación se detallan los parámetros específicos que se han configurado en el fichero `core-site.xml`:

- Tamaño del buffer de E/S de ficheros: 128 KB.
- Especificación del uso de HDFS como sistema de almacenamiento.

Parámetros de HDFS:

En HDFS se trabaja en pequeñas cantidades de información a las que llama bloques. Éstos están normalmente comprendidos entre valores de 64 y 128 MB (Megabytes) debido a que se trabaja con ficheros de gran tamaño (Gibabytes, Petabytes, etc). HDFS tiene como objetivo dividir estos ficheros en bloques de un tamaño fijo y distribuirlos por todos los nodos del clúster. Debido a la cantidad de tráfico que se produce al distribuir todos estos bloques, en nuestro caso se ha optado por utilizar

un tamaño de bloque relativamente grande de 128 MB. Además, HDFS replica todos los bloques en un factor determinado para conseguir tolerancia a fallos. En este caso se ha configurado para que realice 3 réplicas de cada bloque.

A continuación se muestran los parámetros que se han configurado en el fichero `hdfs-site.xml`:

- Tamaño de bloque: 128 MB.
- Réplica: 3

Parámetros de YARN:

YARN se encarga de planificar y ejecutar *containers* o contenedores, los cuales contienen toda la información de la tarea que deben ejecutar, la ubicación de su bloque de datos, así como el nodo donde se deben ejecutar. Cada uno de estos contenedores utiliza la memoria del nodo donde se ejecuta para obtener un mejor rendimiento. Por tanto, dicha memoria se debe reservar y liberar una vez completada su tarea. En este fichero se especifica el tamaño mínimo y máximo que puede tener un contenedor para poder ejecutarse en un nodo. Estos parámetros dependen de la cantidad de memoria que se desee utilizar en cada nodo. En nuestro caso, cada MV posee 16 GB (Gigabytes) de memoria, de los cuales 2 GB se han reservado para el sistema operativo. Los 14 GB restantes se han especificado en este fichero de configuración para que Hadoop los utilice como más le convenga. Concretamente, se ha especificado un tamaño mínimo de contenedor de 1.792 MB y un máximo de 14.336 MB, con lo cual YARN puede lanzar entre 1 y 8 contenedores en cada MV.

Finalmente, se debe indicar cuál es el nodo maestro de Hadoop a través de la dirección IP de éste. A continuación se muestran los parámetros que se han configurado en el fichero `yarn-site.xml`:

- Cantidad de memoria disponible: 14.336 MB
- Tamaño mínimo de contenedor: 1.792 MB
- Tamaño máximo de contenedor: 14.336 MB
- Dirección IP del nodo maestro.

Parámetros de MapReduce:

Cada contenedor creado por YARN puede contener un tipo de tarea distinta. En este caso, al utilizar el modelo de programación MapReduce, un contenedor puede poseer una tarea *Map* o una tarea *Reduce*. Sin embargo, cada tipo de tarea necesita una cantidad mínima de memoria para poder ejecutarse correctamente, la cual debe acoplarse al tamaño de contenedor configurado. En este caso se ha configurado este fichero para que cada tarea *Map* disponga de 3.584 MB, y cada tarea *Reduce* de 7.168 MB. Con estas decisiones, YARN es capaz de lanzar hasta 4 contenedores de tipo *Map* o hasta 2 contenedores de tipo *Reduce* en cada MV. Además, como se ha descrito anteriormente, cuando YARN inicia una aplicación se ejecuta un contenedor especial llamado *ApplicationMaster* que realiza la función de gestión de la aplicación. Este contenedor se ha configurado con un tamaño de 7.168 MB, de forma que una vez planificado quede espacio disponible en dicho nodo para otros contenedores.

Como se puede observar, el concepto de slot en Hadoop se configura a través de esta gestión de memoria, por lo que podemos hablar de que cada MV posee 4 slots disponibles para tareas *Map* o 2 slots para tareas *Reduce*, o bien la combinación de ambos. A continuación se muestran los parámetros configurados en el fichero `mapred-site.xml`:

- Tamaño máximo para una tarea *Map*: 3.584 MB
- Tamaño máximo para una tarea *Reduce*: 7.168 MB
- Tamaño máximo del (*ApplicationMaster*): 7.168 MB

Una vez configurados todos los parámetros en todas las MVs, se ha especificado qué MVs van ejercer el rol de esclavos y qué MV realizará la gestión de nodo maestro. Para ello, se ha utilizado el fichero `slaves` de la MV designada como maestro y ubicada en la nube *on-premise*.

Cabe destacar que, aunque todas las MVs, tanto *on-premise* como *off-premise*, contengan toda la instalación y configuración de Hadoop, éstas solo formarán parte del sistema si se lanzan todos los servicios de Hadoop necesarios y se agregan al fichero `slaves` de la MV maestro. De este modo, podemos partir de un clúster Hadoop de una sola MV como esclavo hasta 15 MVs como esclavos de Hadoop, utilizando ambas nubes de forma híbrida.

CAPÍTULO 3

H-Stat: Entorno de extracción estadística para Hadoop

Como parte de la investigación y experimentación realizada en esta tesis surge la necesidad de analizar los detalles relativos a la ejecución de una aplicación MapReduce en una nube híbrida. Esto incluye información relativa a la planificación de tareas que realiza MapReduce en dicho entorno, extracción de estadísticas de cada una de las tareas, información sobre la utilización de los recursos de la infraestructura, así como información relativa tanto al uso del sistema de almacenamiento HDFS como al tráfico de datos producido entre ambas nubes.

Cada módulo de Hadoop permite activar su depuración para un posterior análisis de su ejecución. Sin embargo, la extracción de información relevante a partir de los ficheros de registro que produce Hadoop no es una tarea sencilla. La mayoría de los ficheros que se generan están orientados a la depuración de la implementación de Hadoop; por tanto, suelen ser ficheros sin formatear y de un gran volumen.

Con el fin de extraer toda esta información en un formato sencillo y fácil de analizar, se ha desarrollado un entorno de extracción de datos para Hadoop, que permite obtener y combinar todos los ficheros de registro que se producen, junto con información adicional acerca del uso de la infraestructura utilizada. Esto permite realizar un completo análisis estadístico mediante una serie de herramientas incorporadas en dicho entorno, e implementadas para este fin.

En este capítulo se presenta dicho entorno, llamado H-Stat, compuesto por una colección de herramientas desarrolladas durante esta tesis para la extracción de información estadística de la ejecución de una aplicación Hadoop MapReduce. En primer lugar se presenta el entorno de extracción de datos, para posteriormente introducir las herramientas que componen el núcleo de H-Stat, que son: H-Parser, H-Profile, H-HDFS y H-Traffic.

3.1 Entorno de extracción

En esta sección se presenta la visión general del entorno H-Stat. El uso de H-Stat requiere una configuración previa del entorno, así como el uso de algunas herramientas complementarias. Por

tanto, a continuación se describen todos los requisitos y la configuración necesaria para utilizar H-Stat. Además, se presenta su arquitectura y flujo de trabajo.

3.1.1 Requisitos previos

H-Stat requiere una configuración de Hadoop específica que asegure la generación de todos los ficheros de registro que será necesario analizar tras una ejecución de una aplicación MapReduce. Además, H-Stat requiere el uso de una serie de herramientas complementarias para monitorizar los recursos físicos donde se ejecuta Hadoop MapReduce (*Sysstat*), para ayudar a la extracción de información sobre tareas MapReduce (*Rumen*) o para visualizar las trazas (*Paraver*) creadas con nuestra herramienta. Todos los datos deben, además, estar perfectamente sincronizados en una línea de tiempo, de cara a poder establecer posteriormente relaciones entre ellos. En los siguientes puntos se describen las herramientas complementarias utilizadas y la configuración necesaria para el correcto uso de H-Stat.

Configuración de Hadoop

H-Stat hace uso de los datos de registro (ficheros de *log*) de Hadoop a partir de la ejecución de una aplicación MapReduce, teniendo en cuenta los servicios implicados en la ejecución, tales como YARN, HDFS o MapReduce. Para ello, se deben activar una serie de parámetros en el fichero de configuración de cada módulo de Hadoop, de forma que asegure la generación de toda la información necesaria.

En primer lugar, en el fichero de configuración de MapReduce se debe activar el uso del historial de trabajos MapReduce, de forma que recopile información acerca de las tareas y acciones MapReduce ocurridas durante todo el transcurso de su ejecución. Para ello, se debe indicar la dirección IP del servidor de historial de trabajos (*Job History Server*), y los directorios temporales sobre los que escribirán los ficheros intermedios y los completados.

- `mapreduce.jobhistory.address`
- `mapreduce.jobhistory.intermediate-done-dir`
- `mapreduce.jobhistory.done-dir`

En segundo lugar, se debe activar la depuración en la configuración de HDFS para que recopile información con respecto al tráfico de bloques de datos entre los nodos participantes:

- `log4j.logger.org.apache.hadoop.hdfs.server.datanode.DataNode.clienttrace = DEBUG`

Finalmente, se debe iniciar el demonio de registro de historial de trabajos de Hadoop para que recopile información acerca de la ejecución de una aplicación MapReduce en los directorios indicados en el primer paso:

- `mr-jobhistory-daemon.sh`

Rumen

Apache *Rumen* [7] es una herramienta de extracción de información histórica de tareas MapReduce, a partir de un fichero de registro de un trabajo MapReduce ejecutado.

En nuestro caso, H-Stat hace uso de *Rumen* para generar dos ficheros en formato *json*, `job-trace.json` y `job-topology.json`, a partir de los ficheros de registro extraídos de una ejecución. El fichero `job-trace.json` contiene la información relativa al inicio, finalización, estado y otros parámetros de todas las tareas MapReduce ejecutadas para una aplicación. Y el fichero `job-topology.json` contiene la información acerca de los nodos físicos implicados para su ejecución, en nuestro caso MVs.

Sysstat

Sysstat [43] es un conjunto de utilidades para la monitorización del rendimiento y de la actividad de sistemas Linux. *Sysstat* contiene varias utilidades y herramientas que permiten planificar de forma automática la recogida de datos de actividad y rendimiento del sistema. Algunos de los parámetros que permite monitorizar son estadísticas de utilización de CPU, E/S de disco, E/S de red y utilización de memoria, entre otras.

H-Stat hace uso de *Sysstat* para extraer la información relativa a los nodos implicados en una ejecución de una aplicación MapReduce. De esta forma, se obtiene información sobre CPU, E/S de disco, E/S de red y memoria de cada uno de los nodos con una frecuencia de muestreo específica (en nuestro caso de 5 segundos).

TakTuk

TakTuk [22] es un software que permite la ejecución paralela de comandos en un clúster. Es un administrador de nodos, que intenta minimizar al máximo el tiempo requerido para lanzar un comando a los nodos de un clúster, ejecutándolo de forma eficiente y paralela. *TakTuk* permite enviar comandos, además de copiar, enviar o recibir datos de todos los nodos del clúster con un solo comando.

En un entorno como el presentado en nuestro caso, H-Stat hace uso de *TakTuk* como herramienta para la obtención de información de los nodos de Hadoop, enviar y recibir datos, así como comprobar el estado de todos ellos en cada caso.

Paraver

Paraver [77] es una herramienta de visualización de trazas desarrollada en el Centro de Supercomputación de Barcelona (BSC), que permite la visualización gráfica y análisis de códigos paralelos basados en herramientas paralelas como OpenMP, MPI, OpenMP+MPI, etc. Paraver proporciona un potente entorno para inspeccionar el paralelismo y la escalabilidad de una aplicación, así como para obtener métricas para caracterizar un programa y su rendimiento.

En nuestro caso, H-Stat permite crear trazas adaptadas al formato de Paraver, que contienen la información relativa al inicio, finalización y estado de cada una de las tareas MapReduce ejecutadas,

así como la información relativa a los nodos utilizados. De este modo, Paraver permite sincronizar y visualizar esta información en una línea temporal para el posterior análisis de dichos resultados.

3.1.2 Arquitectura

En la Figura 3.1 se muestra la arquitectura de H-Stat en un ejemplo de ejecución de una aplicación Hadoop MapReduce, en este caso PageRank [18], sobre un escenario de nube híbrida compuesto por 4 MVs *on-premise* (*mr-onpremise-0* ejerce de MV de administración y el resto de cómputo de Hadoop) y 3 MVs *off-premise* (como MVs de cómputo de Hadoop).

Como se muestra en la Figura 3.1, H-Stat se encarga de recoger los datos de registro almacenados en cada una de las MVs participantes en la ejecución de la aplicación. H-Stat recopila la información de cada MV de cada nube a través de *Sysstat*, los registros de HDFS (*HDFS Logs*), el directorio *Job History* y los dos ficheros generados por *Rumen* (`job-trace.json` y `job-topology.json`). Estos datos son almacenados en una base de datos para su posterior explotación con la colección de herramientas de H-Stat descritas en la siguiente sección.

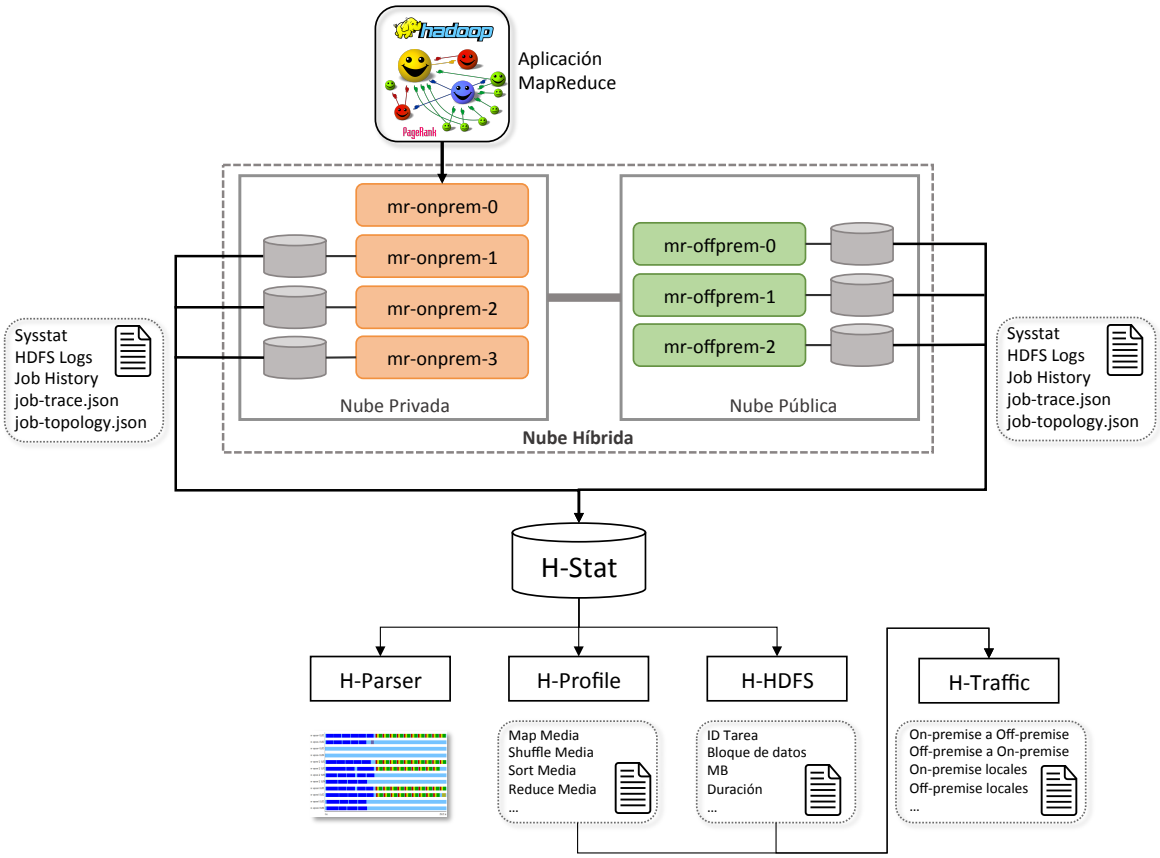


Figura 3.1: Arquitectura del entorno de extracción de datos H-Stat.

3.2 Herramientas de extracción

En esta sección se presentan las herramientas de H-Stat implementadas durante esta tesis para la extracción y el análisis, tanto a nivel de evolución temporal como estadístico, de cada una de las ejecuciones de las aplicaciones MapReduce en los diferentes escenarios de ejecución. En concreto, se presentan cuatro herramientas como parte del entorno de H-Stat, cada una de ellas encargada de una parte del análisis de los datos.

Para mostrar un ejemplo de salida de cada una de las herramientas presentadas en esta sección, se ha utilizado una arquitectura como la que se muestra en la Figura 3.1, en la que se ha ejecutado la aplicación PageRank sobre 3 MVs *on-premise* (las 3 MVs destinadas a cómputo) y 3 MVs *off-premise* de una nube híbrida, utilizando un *enlace débil* de 100 Mbps. Por claridad, y para facilitar la lectura de este documento, solo se mostrará parte de la información obtenida en cada caso.

Una de las características de H-Stat es su capacidad de extracción y análisis de datos de aplicaciones MapReduce iterativas, las cuales constan de varios trabajos MapReduce para completar la totalidad de su ejecución. Todas las herramientas de H-Stat permiten el análisis de este tipo de aplicaciones, consiguiendo obtener estadísticas globales de toda la ejecución completa.

La aplicación PageRank es un tipo de aplicación MapReduce iterativa, que ejecuta varias iteraciones de trabajos MapReduce hasta converger. La funcionalidad de esta aplicación queda fuera del contenido de este capítulo, ya que ésta únicamente sirve como ejemplo de uso del entorno H-Stat y de la información que permiten extraer cada una de las herramientas de H-Stat presentadas a continuación.

3.2.1 H-Parser

H-Parser es una herramienta de análisis y creación de trazas a partir de los datos históricos de la ejecución de una aplicación MapReduce. Es capaz de clasificar y sincronizar las etapas de cada una de las tareas MapReduce ejecutadas en cada nodo, y crear una traza de información que permite visualizar su planificación de tareas en una línea temporal. Es decir, extrae información de inicio y finalización de cada una de las tareas MapReduce (*Map*, *Shuffle*, *Sort* y *Reduce*) y genera un fichero de trazas con el formato adecuado para poder ser visualizado por Paraver. Además, combina esta información con información relativa a la utilización de recursos de la infraestructura empleada, es decir, parámetros relativos a la CPU, E/S de disco, E/S de red y memoria utilizada en cada nodo físico (en nuestro caso MVs), todo ello integrado también en el fichero de trazas. De esta forma, permite tener una visión general de la distribución de tareas realizada en una determinada ejecución, así como su aprovechamiento de recursos físicos.

H-Parser utiliza como datos de entrada:

- Los ficheros `job-topology.json` y `job-trace.json` generados por la herramienta *Rumen*, que contienen el detalle de cada una de las tareas MapReduce ejecutadas, además de la información acerca de la topología física utilizada.
- La información obtenida por *Sysstat* acerca de la utilización de los recursos de cómputo empleados en la ejecución de la aplicación, en cuanto a utilización de CPU, E/S de red, E/S de disco y memoria.

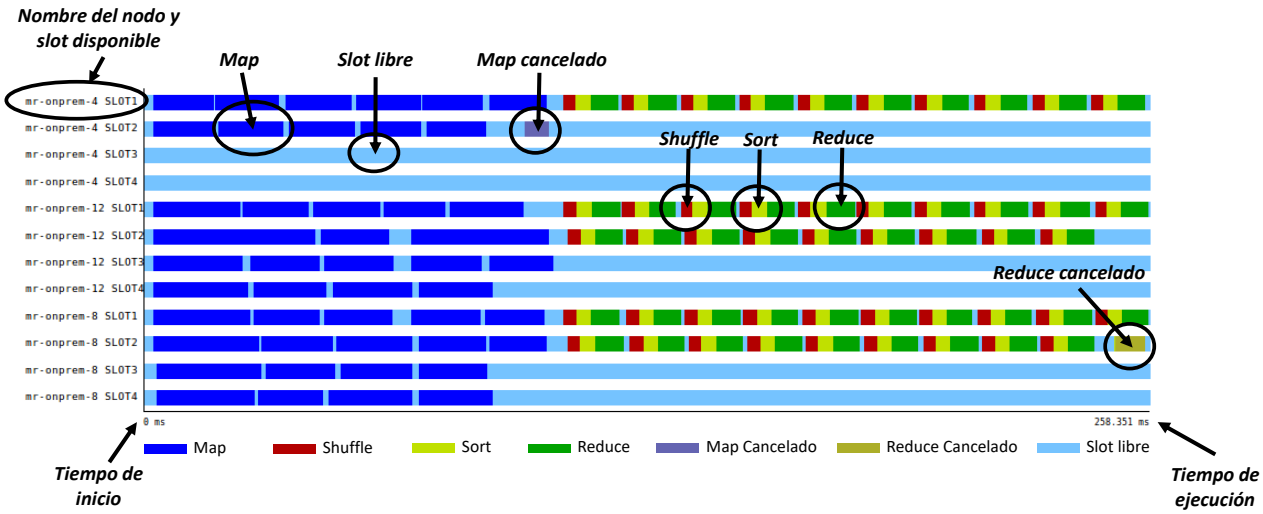


Figura 3.2: Descripción de los elementos de visualización obtenidos por H-Parser. Visor: Paraver.

En la Figura 3.2 se describen los elementos, relativos a las etapas MapReduce, que aparecen en la visualización de una traza creada con esta herramienta, haciendo uso de Paraver para su visualización. Como se puede observar, se trata de una línea temporal en la que cada fila corresponde a un slot de un determinado nodo físico o MV. En cada slot aparece el detalle de la planificación que ha realizado Hadoop en cada etapa MapReduce. A continuación se describe cada elemento:

- **Nombre del nodo y slot disponible:** en el eje de ordenadas se representan los slots de cada uno de los nodos de la infraestructura; por ejemplo, `mr-onprem-4 SLOT1` indica, en este caso, el slot número 1 de la MV mr-onpremise-4.
- **Slot libre:** en azul claro se representa el tiempo en el que el slot correspondiente no está siendo utilizado.
- **Tiempo de inicio:** corresponde con el tiempo inicial desde el cual empieza la ejecución de la aplicación y su correspondiente monitorización.
- **Tiempo de ejecución:** representa el tiempo que ha tardado la aplicación en terminar su ejecución.
- **Map**: en azul oscuro se representa el tiempo de ejecución de una tarea *Map* en un determinado slot de un nodo.
- **Reduce:** en verde oscuro se representa el tiempo de ejecución de una tarea *Reduce* en un determinado slot de un nodo.
- **Shuffle:** en rojo se representa el tiempo de ejecución de la fase *Shuffle* en un determinado slot de un nodo.
- **Sort:** en amarillo se representa el tiempo de ejecución de la fase *Sort* en un determinado slot de un nodo.
- **Map cancelado**: las tareas *Map* que han sido canceladas por Hadoop debido a cualquier tipo de error se muestran en la línea de tiempo como franjas de color morado.

- **Reduce cancelado:** las tareas *Reduce* que han sido canceladas por Hadoop debido a algún tipo de error se muestran en la línea de tiempo como franjas de color verde claro.

Finalmente, en la Figura 3.3 se muestra un ejemplo de salida completa de la ejecución de la primera iteración de PageRank sobre la nube híbrida descrita al inicio de esta sección. En esta figura se muestran los elementos completos sincronizados con la utilización de recursos de CPU, E/S de red, E/S de disco y memoria. En este caso concreto se muestra únicamente una MV *on-premise* (*mr-onprem-1*) y una MV *off-premise* (*mr-offprem-0*) por una cuestión de espacio, pero en la salida real se mostrarían las seis MVs completas. En la parte superior de la figura se puede apreciar la línea de tiempo de planificación de tareas MapReduce, en la que cada MV dispone de cuatro slots disponibles (cuatro para tareas *Map* y dos para tareas *Reduce*). En la parte inferior de la figura se muestra la utilización de los nodos, sincronizados con la línea de tiempo, cada uno con su escala correspondiente. Se presenta, en orden desde arriba hacia abajo en la figura, el porcentaje de utilización de CPU, la cantidad de información que se transfiere por la red, la cantidad de información leída o escrita en el disco y el porcentaje de utilización de memoria.

Realizando un pequeño análisis de la figura, se observa una alta actividad de la CPU correspondiente a la fase *Map*, lo que muestra una gran eficiencia en cuanto a la utilización de los recursos en dicha fase, cercana al 100 %. Respecto a las transferencias por la red, se observa en la gráfica de la red (NetRx) que existen 4 picos de comunicación que coinciden con la fase *Shuffle* de cada una de las tareas MapReduce, momento que se corresponde con la transferencia de la salida producida por las tareas *Map* a las tareas *Reduce* a través de dicha red. También se observa que las lecturas de disco (DiscRd) se realizan exclusivamente en las tareas *Map*, ya que es cuando se realiza la lectura de los datos de entrada desde HDFS. En cambio, en las tareas *Reduce* se observa más actividad en la escritura en disco (DiscWd) y en las transferencias en la red, debido a que las tareas *Reduce* escriben su resultado final en HDFS y éste se replica en diferentes MVs. Por tanto, con un reducido ancho de banda del *enlace débil* (en este caso de 100 Mbps) se produce un cuello de botella de la red, repercutiendo en una infrautilización de la CPU en dichas tareas *Reduce*, aspecto que se observa en la gráfica de CPU en toda la fase *Reduce*.

Aunque la inspección visual nos permite entender fácilmente el comportamiento de las aplicaciones, necesitamos también datos estadísticos fiables para utilizar en nuestros modelos. De la extracción de dichos datos estadísticos se encargan las herramientas H-Profile, H-HDFS y H-Traffic, que se presentan a continuación.

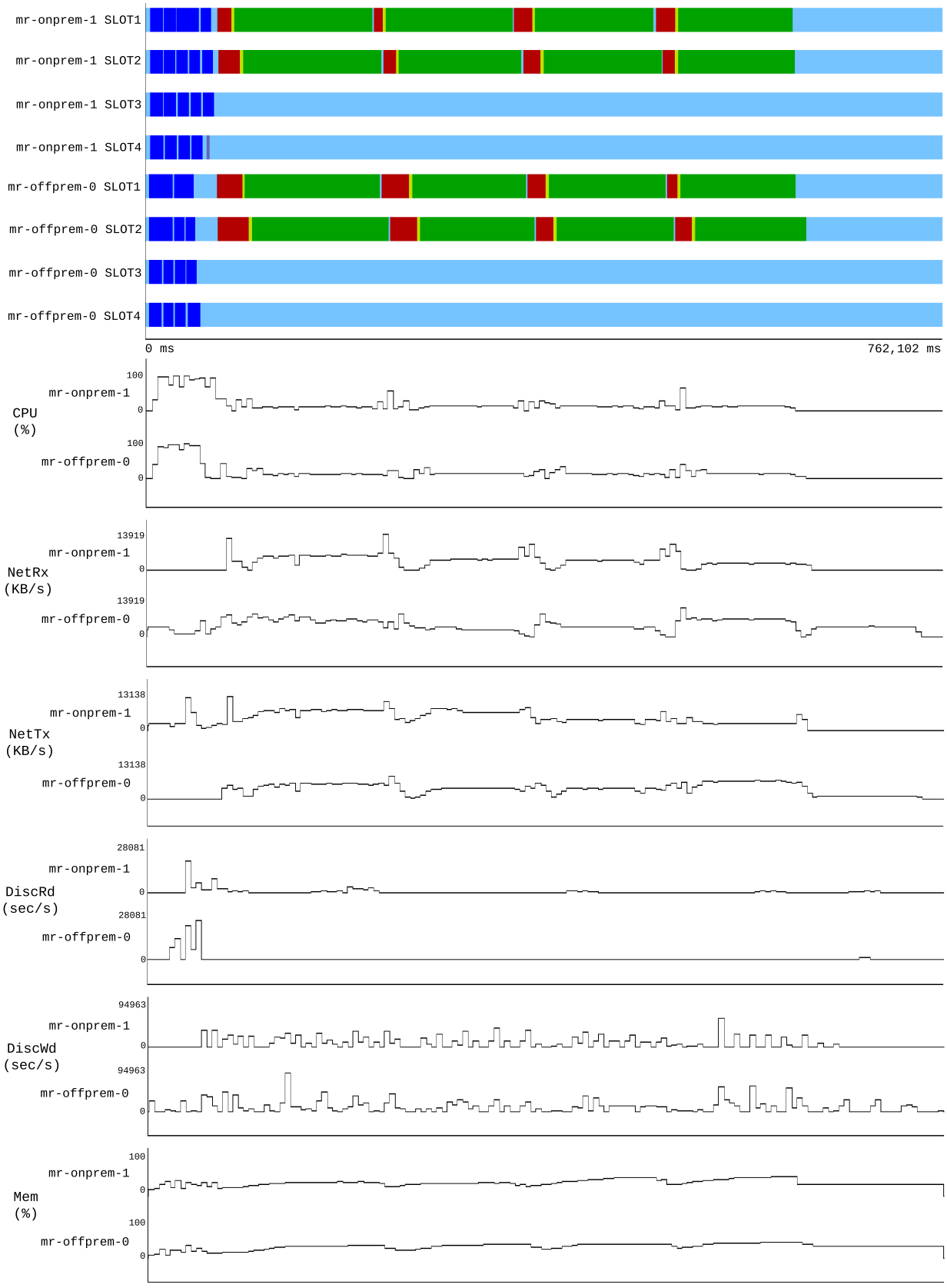


Figura 3.3: Ejemplo de salida de H-Parser para la aplicación PageRank. Visor: Paraver.

3.2.2 H-Profile

Esta herramienta se encarga de extraer información estadística de cada uno de los trabajos MapReduce ejecutados en un determinado escenario. Partiendo de la misma información proporcionada por los ficheros `job-trace.json` y `job-topology.json`, ésta permite extraer los parámetros que se muestran en la Tabla 3.1 para cada uno de los trabajos Hadoop.

Parámetro	**Descripción**
Nº Trabajo	Número de trabajo de una aplicación MapReduce
Tiempo Ejecución (s)	Tiempo de ejecución del trabajo
Tiempo Lanzamiento (s)	Tiempo transcurrido desde la planificación hasta la ejecución del trabajo
Nombre Trabajo	Nombre asignado al trabajo.
Total Maps	Número total de tareas Map.
Total Reduce	Número total de tareas Reduce.
Map Media (s)	Tiempo medio de ejecución de las tareas Map.
Map Desv. Típ. (s)	Desviación típica del tiempo de ejecución de las tareas Map.
Map Máximo (s)	Tiempo más alto de la ejecución total de tareas Map.
Shuffle Media (s)	Tiempo medio de ejecución de las tareas Shuffle.
Shuffle Desv. Típ. (s)	Desviación típica del tiempo de ejecución de las tareas Shuffle.
Shuffle Máximo (s)	Tiempo más alto de la ejecución total de las tareas Shuffle.
Sort Media (s)	Tiempo medio de ejecución de las tareas Sort.
Sort Desv. Típ. (s)	Desviación típica del tiempo de ejecución de las tareas Sort.
Sort Máximo (s)	Tiempo más alto de la ejecución total de las tareas Sort.
Reduce Media (s)	Tiempo medio de ejecución de las tareas Reduce.
Reduce Desv. Típ. (s)	Desviación típica del tiempo de ejecución de las tareas Reduce.
Reduce Máximo (s)	Tiempo más alto de la ejecución total de las tareas Reduce.
Planificación Media (s)	Tiempo medio de planificación entre tareas.
Planificación Desv. Típ. (s)	Desviación típica del tiempo de planificación entre tareas.
Planificación Máximo (s)	Tiempo más alto de planificación entre tareas.
Shuffle Datos Media (MB)	Cantidad de datos media que recibe cada tarea Shuffle.
Reduce Datos Media (MB)	Cantidad de datos media que escribe cada tarea Reduce.

Tabla 3.1: Descripción de los parámetros obtenidos por H-Profile.

Puesto que uno de los objetivos principales de esta tesis es la predicción del tiempo de finalización de una aplicación MapReduce iterativa en una nube híbrida, es necesario crear un perfil previo de la aplicación que se desea modelar. Esta herramienta permite extraer aquellos parámetros relevantes con el fin de crear un perfil concreto de la aplicación. Algunos de los parámetros obtenidos, tales como el tiempo medio de las tareas *Map*, la planificación media entre tareas, la cantidad de datos medios de *Shuffle* y *Reduce* o el total de tareas *Map* y *Reduce*, son factores invariantes de una aplicación, que tras una ejecución en una arquitectura con un número diferente de nodos permanecen constantes. Éstos últimos son muy relevantes para el perfil, ya que, como se verá en los próximo capítulos, permiten ampliar o disminuir la cantidad de nodos homogéneos de un clúster sin que se vean modificados. El resto de parámetros sirven como análisis del comportamiento de la aplicación en un escenario concreto.

La información que obtiene esta herramienta es tratada previamente a través de un proceso estadístico de eliminación de picos para obtener información más precisa, por el cual se eliminan aquellos puntos que están fuera de un determinado intervalo definido por la suma de la media y la desviación típica.

A continuación, y siguiendo con el ejemplo experimental de la aplicación PageRank descrita al inicio de la sección, se muestra un ejemplo de salida de H-Profile para los primeros cuatro trabajos de PageRank en la Tabla 3.2.

Parámetro	**Valor**			
Nº Trabajo	1	2	3	4
Tiempo Ejecución (s)	762,102	276,97	722,039	278,16
Tiempo Lanzamiento (s)	8,15	2,89	6,31	3,81
Nombre Trabajo	Pagerank_Stage1	Pagerank_Stage2	Pagerank_Stage1	Pagerank_Stage2
Total Maps	98	48	144	48
Total Reduce	48	48	48	48
Map Media (s)	10,27	17,75	8,09	17,82
Map Desv. Típ. (s)	0,55	0,48	1,71	0,75
Map Máximo (s)	11,51	18,81	10,33	19,71
Shuffle Media (s)	17,32	33,57	14,56	30,41
Shuffle Desv. Típ. (s)	2,63	2,01	2,96	3,06
Shuffle Máximo (s)	32,95	44,2	25,41	38,08
Sort Media (s)	2,43	2,86	2,38	2,81
Sort Desv. Típ. (s)	0,08	0,06	0,096	0,072
Sort Máximo (s)	2,71	3,11	3,29	3,41
Reduce Media (s)	112,65	9,97	108,86	11,043
Reduce Desv. Típ. (s)	4,74	0,88	1,85	0,84
Reduce Máximo (s)	132,37	17,28	114,63	21,98
Planificación Media (s)	1,78	1,85	1,73	1,87
Planificación Desv. Típ. (s)	0,28	0,45	0,32	0,39
Planificación Máximo (s)	22,96	20,79	10,21	20,02
Shuffle Datos Media (MB)	56,81	116,31	56,7	115,94
Reduce Datos Media (MB)	122,02	4,48	121,66	4,47

Tabla 3.2: Ejemplo de salida de H-Profile de cuatro trabajos de PageRank.

3.2.3 H-HDFS

H-HDFS se encarga de la extracción de los datos relativos a la lectura/escritura de bloques de datos del sistema distribuido de almacenamiento HDFS durante el intervalo de ejecución de una aplicación MapReduce. En la Tabla 3.3 se muestran los parámetros que se extraen a partir de los datos de registro (*HDFS Logs*).

La información que proporciona H-HDFS nos permiten realizar un análisis detallado de las acciones y transferencias de datos que se producen en HDFS. Por cada tarea *Map* o *Reduce* que se ejecuta, se producen una serie de lecturas y/o escrituras en HDFS de los bloques de datos solicitados por YARN. Todo ello nos permite analizar el comportamiento y preferencias de lectura/escritura que produce HDFS en un entorno de nube híbrida como el que se analiza en esta tesis.

Parámetro	**Descripción**
Id. Tarea	Identificador de la tarea MapReduce (M *Map*, R *Reduce*).
IP Origen	IP de origen de la transferencia.
IP Destino	IP de destino de la transferencia.
Tiempo Inicial	Tiempo inicial de la transferencia con precisión de segundos.
Tiempo Final	Tiempo final de la transferencia con precisión de segundos.
Duración (s)	Duración de la transferencia en segundos.
Tipo Operación	Tipo de operación (lectura/escritura).
Tamaño (MB)	Cantidad de datos transferida en MB.
Bloque	Bloque de datos implicado.
Transferencia	Indica si la transferencia ha sido local o remota.
Ancho Banda (MB/s)	Ancho de banda de la transferencia en MB/s.
Nodo	Nodo implicado en la lectura o escritura.

Tabla 3.3: Descripción de los parámetros obtenidos por H-HDFS.

Como ejemplo de salida, en la Tabla 3.4 se muestra el resultado que produciría H-HDFS en la aplicación PageRank sobre la arquitectura descrita en la sección 3.1, mostrando únicamente una tarea *Map* y una *Reduce*. Haciendo un pequeño análisis del ejemplo se puede observar que en este caso la tarea *Map* necesita leer de tres bloques de datos distintos del HDFS para completar el *Split* de datos solicitado por YARN (caso explicado en la sección 2.3.2). Como se muestra en la fila etiquetada como Transferencia, estas lecturas son locales, ya que dichos bloques se encuentran en el mismo nodo (*mr-offprem-1*).

En el caso de la tarea *Reduce*, se puede observar que escribe tres bloques de datos iguales pero en tres MVs distintas, dos en remoto y uno en local, lo cual nos indica que HDFS tiene configurada una réplica de un factor tres para proporcionar tolerancia a fallos. Hay que resaltar que, cuando se muestra la duración de la operación de escritura, dicho tiempo corresponde a la duración total de las tres operaciones producidas en paralelo, ya que HDFS considera que el inicio y finalización de escritura de un bloque incluye la escritura de todas su réplicas, por tanto se muestra la misma duración en los tres bloques.

En la Figura 3.4 se muestra el flujo de datos de este ejemplo concreto. Como se ha descrito anteriormente, los tres bloques que lee la tarea *Map* se encuentran en la misma MV (flechas gruesas). Por contra, mediante flechas discontinuas se representan las escrituras que realiza la tarea *Reduce*

en la MV local (*mr-onprem-1*) y en las MVs remotas (*mr-offprem-1* y *mr-offprem-0*). Este flujo de información se observa en las filas etiquetadas como *IP Origen* e *IP Destino* de la Tabla 3.4.

Parámetro	**Valor**					
Id. Tarea	M_000000	M_000000	M_000000	R_000000	R_000000	R_000000
IP Origen	192.168.2.17	192.168.2.17	192.168.2.17	192.168.1.13	192.168.2.16	192.168.1.13
IP Destino	192.168.2.17	192.168.2.17	192.168.2.17	192.168.2.16	192.168.2.17	192.168.1.13
Tpo. Inicial	12:16:25.995	12:16:27.005	12:16:30.57	12:17:45.86	12:17:45.86	12:17:45.86
Tpo Final	12:16:26	12:16:31	12:16:38	12:19:56	12:19:56	12:19:56
Duración (s)	0,0044	3,99	7,43	130,14	130,14	130,14
Tipo Op.	READ	READ	READ	WRITE	WRITE	WRITE
Tamaño (MB)	0,011	129	8,57	122,02	122,02	122,02
Bloque	blk_1148	blk_1145	blk_1146	blk_1153	blk_1153	blk_1153
Transferencia	Local	Local	Local	Remota	Remota	Local
Ancho B. (MB/s)	2,52	32,29	1,15	0,94	0,94	0,94
Nodo	mr-offprem-1	mr-offprem-1	mr-offprem-1	mr-offprem-0	mr-offprem-1	mr-onprem-1

Tabla 3.4: Ejemplo de salida de H-HDFS para una tarea *Map* y una *Reduce* de PageRank.

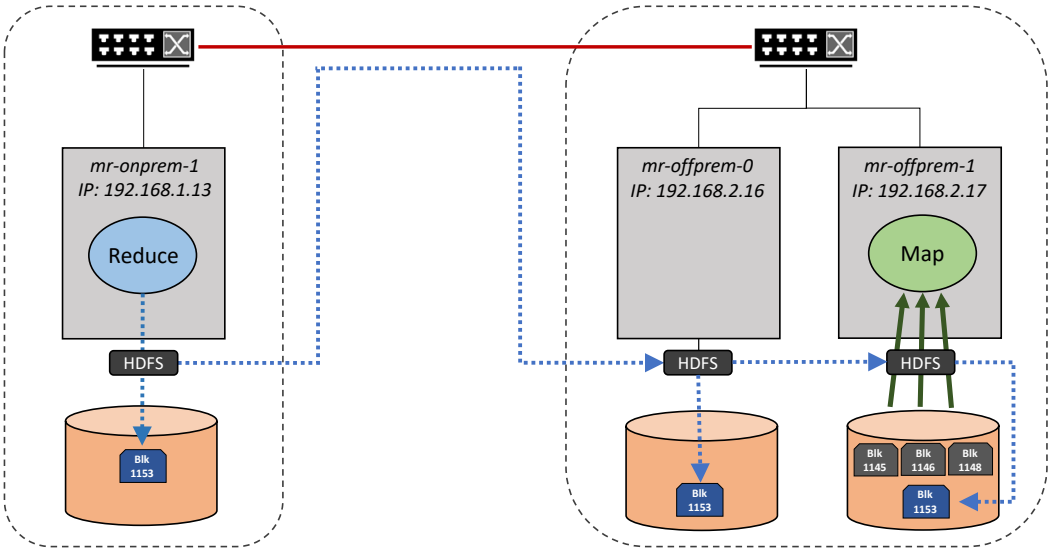


Figura 3.4: Flujo de datos del ejemplo de salida de H-HDFS para una tarea *Map* y una *Reduce* de PageRank.

3.2.4 H-Traffic

H-Traffic es una herramienta que permite realizar un análisis del tráfico de datos que produce una aplicación en un entorno de nube híbrida como el estudiado en esta tesis.

En la Tabla 3.5 se muestran todos los parámetros que se obtienen a través de esta herramienta.

Parámetro	**Descripción**
Nº Trabajo	Número de iteración o trabajo dentro de una aplicación
Tiempo Ejecución (s)	Tiempo de ejecución del trabajo
	HDFS LECTURAS (MB)
Off-premise desde On-premise	Total de datos leídos Off-premise desde los nodos On-premise
On-premise desde Off-premise	Total de datos leídos On-premise desde los nodos Off-premise
Local On-premise	Total de datos leídos en los nodos On-premise en local
Local Off-premise	Total de datos leídos en los nodos Off-premise en local
Total Datos Leídos	Total de datos leídos de HDFS
	HDFS ESCRITURAS (MB)
On-premise hacia Off-premise	Total de datos escritos Off-premise desde los nodos On-premise
Off-premise hacia On-premise	Total de datos escritos On-premise desde los nodos Off-premise
Local On-premise	Total de datos escritos en los nodos On-premise en local
Local Off-premise	Total de datos escritos en los nodos Off-premise en local
Total Datos Escritos	Total de datos escritos en HDFS
	SHUFFLE (MB)
Shuffle On-premise hacia Off-premise	Datos Shuffle transferidos desde On-premise hacia Off-premise
Shuffle Off-premise hacia On-premise	Datos Shuffle transferidos desde Off-premise hacia On-premise
Shuffle On-Premise	Datos Shuffle transferidos On-premise
Shuffle Off-Premise	Datos Shuffle transferidos Off-premise
Total Shuffle	Total de datos Shuffle transferidos
	ESTADÍSTICAS
Maps On-premise	Total de tareas Map ejecutadas On-premise
Reduces On-Premise	Total de tareas Reduce ejecutadas On-premise
Maps Off-premise	Total de tareas Map ejecutadas Off-premise
Reduces Off-premise	Total de tareas Reduce ejecutadas Off-premise
Maps Cancelados On-premise	Total de tareas Map canceladas On-premise
Reduces Cancelados On-premise	Total de tareas Reduce canceladas On-premise
Maps Cancelados Off-premise	Total de tareas Map canceladas Off-premise
Reduces Cancelados Off-premise	Total de tareas Reduce canceladas Off-premise
Total Maps Satisfactorios	Total de tareas Map ejecutadas de forma satisfactoria
Total Reduces Satisfactorios	Total de tareas Reduce ejecutadas de forma satisfactoria

Tabla 3.5: Descripción de los parámetros obtenidos por H-Traffic.

Esta herramienta parte de los datos extraídos de H-Profile y H-HDFS para generar su salida. Se encarga de realizar un análisis de todas las transferencias generadas en HDFS, así como de analizar el tráfico generado por la fase *Shuffle*. De esta forma, permite tratar y combinar estos datos para extraer información acerca de la cantidad de tráfico generado por cada nube, y entre ambas nubes.

Clasificar este tráfico permite estudiar el comportamiento de HDFS desde otra perspectiva, ya que, como se verá más adelante, el tráfico producido entre nubes suele conllevar un coste económico para una nube pública, y por tanto es importante cuantificarlo.

Parámetro	**Valor**			
Nº Trabajo	1	2	3	4
Tiempo Ejecución (s)	736,38	276,34	696,14	277,59
HDFS LECTURAS (MB)				
Off-premise desde On-premise	30,40	0,00	0,00	0,00
On-premise desde Off-premise	0,00	0,00	0,00	0,00
Local On-premise	1772,05	3064,79	1575,23	2819,98
Local Off-premise	1306,26	2951,56	1519,94	3144,38
Total Datos Leídos	3108,70	6016,35	3095,18	5964,36
HDFS ESCRITURAS (MB)				
On-premise hacia Off-premise	3294,66	98,58	3041,62	94,23
Off-premise hacia On-premise	2562,41	116,51	2798,12	120,95
Local On-premise	5857,07	215,09	5839,73	215,19
Local Off-premise	5857,07	215,09	5839,73	215,19
Total Datos Escritos	17571,21	645,28	17519,20	645,57
SHUFFLE (MB)				
Shuffle On-premise hacia Off-premise	718,32	1512,56	661,10	1500,88
Shuffle Off-premise hacia On-premise	610,51	1317,42	698,95	1359,24
Shuffle On-Premise	923,60	1317,42	718,64	1250,50
Shuffle Off-Premise	474,82	1512,56	642,99	1631,39
Total Shuffle	2727,26	5659,95	2721,69	5742,00
ESTADÍSTICAS				
Maps On-premise	60	25	73	23
Reduces On-Premise	27	23	25	23
Maps Off-premise	39	24	71	26
Reduces Off-premise	21	26	23	27
Maps Cancelados On-premise	1	1	0	0
Reduces Cancelados On-premise	0	1	0	2
Maps Cancelados Off-premise	0	0	0	1
Reduces Cancelados Off-premise	0	0	0	0
Total Maps Satisfactorios	98	48	144	48
Total Reduces Satisfactorios	48	48	48	48

Tabla 3.6: Ejemplo de salida de H-Traffic para cuatro trabajos de PageRank.

Para entender mejor el tráfico generado, H-Traffic extrae una serie de estadísticas con respecto a las tareas MapReduce ejecutadas. Las tareas MapReduce no siempre consiguen ejecutarse de forma correcta, y éstas deben ser canceladas y planificadas de nuevo. Esto es debido a que MapReduce configura un tiempo límite de respuesta de estado de una tarea para comprobar su correcto funcionamiento. Si se supera este límite ésta se cancela y se planifica de nuevo, ya que MapReduce considera que ha ocurrido algún error o no se ha ejecutado correctamente. Si una nube híbrida

dispone de un *enlace débil* con un ancho de banda limitado, se puede producir una congestión en la red que produzca que algunas de las tareas MapReduce no puedan informar a tiempo de su estado, y sean canceladas y planificadas de nuevo. En este último caso, la tarea cancelada también consume un tráfico de datos, que en muchos casos no es despreciable, al tener que leer o escribir parte de los datos antes de ser cancelada. Por tanto, H-Traffic permite extraer también esta información para realizar un análisis más completo del tráfico generado por una aplicación MapReduce.

Siguiendo con el mismo ejemplo empleado para el resto de herramientas, en la Tabla 3.6 se muestra la salida de los cuatro primeros trabajos de PageRank. Como se puede observar, en este caso no se realizan lecturas entre nubes *on-premise* y *off-premise* salvo en el primer trabajo, ya que los datos se encuentran inicialmente *on-premise* pero se migran rápidamente. Por tanto el resto de lecturas se hacen localmente. Por el contrario, se puede apreciar una gran actividad de tráfico entre nubes en la escritura, ya que al menos un bloque de datos debe ser escrito en ambas nubes para equilibrar el sistema. También se aprecia un tráfico de *Shuffle* más o menos balanceado entre nubes, lo que indica que el número de MVs *on-premise* y *off-premise* es similar. Finalmente, observando las estadísticas podemos apreciar que se han cancelado pocas tareas *Map* o *Reduce*, ya que todas las lecturas se han hecho prácticamente en local, dejando libre la red para el tráfico de escritura. Esto permite evitar una gran congestión de la red y la consecuente cancelación de tareas.

3.3 Resumen del capítulo

En este capítulo se ha presentado H-Stat, un entorno de extracción y análisis estadístico de Hadoop, enfocado al análisis de aplicaciones MapReduce sobre un entorno de nube híbrida. H-Stat está compuesto por una colección de herramientas de extracción de datos que permiten analizar distintos aspectos de la ejecución de una aplicación MapReduce, tales como su planificación de tareas, estadísticas sobre las tareas, la E/S de bloques de datos de HDFS, así como el tráfico de datos generado entre nubes. Además, con la ayuda de herramientas como Paraver, permite visualizar gráficamente la planificación de tareas MapReduce realizada en un determinado entorno, así como la utilización de los recursos físicos empleados.

Otra de las características a destacar de H-Stat es que permite su uso en el análisis de aplicaciones MapReduce iterativas, de forma que una vez finalizadas las iteraciones de dicha aplicación, es capaz de analizar la información de forma global, dando una visión más general de su ejecución.

Este entorno se ha implementado íntegramente durante la investigación de esta tesis, y su utilización ha permitido la extracción y el análisis de información esencial para el desarrollo de esta investigación.

CAPÍTULO 4

Análisis de la gestión de datos y planificación de tareas

En una arquitectura de nube híbrida la comunicación de datos entre ambas partes, *on-premise* y *off-premise*, se produce a través de un único canal de comunicación de alta latencia y bajo rendimiento, normalmente un orden de magnitud menor de ancho de banda, con respecto a las redes propias de cada nube. Para un entorno como Hadoop, donde la localidad de los datos es un factor clave para obtener un alto rendimiento, la ejecución de una aplicación MapReduce iterativa sobre un entorno de nube híbrida puede mermar su rendimiento de forma muy considerable sin una correcta gestión de los datos de entrada y planificación de tareas. Además, en una nube híbrida los recursos de la nube pública conllevan un coste económico en función del tiempo utilizado y de la cantidad de tráfico de datos producido de entrada y salida a la nube pública. Por tanto, es necesario estudiar y analizar distintas estrategias de gestión de los datos y políticas de planificación de tareas que eviten generar un coste económico innecesario y mejoren el rendimiento de este tipo de aplicaciones.

En este capítulo se presenta un análisis de distintas alternativas para la gestión de los datos y planificación de tareas de un entorno Hadoop MapReduce sobre una nube híbrida. Concretamente, este análisis se centra en un tipo de despliegue de nube híbrida denominado *cloud bursting*, en el cual un conjunto nuevo de MVs *off-premise* se adhieren de forma puntual a las MVs *on-premise* para contribuir al cálculo. Sin embargo, todos los datos de entrada se encuentran inicialmente en las MVs *on-premise*. Por tanto, se deben buscar y evaluar distintas estrategias que permitan el uso de este tipo de despliegues de nube híbrida de una forma eficiente. El objetivo de este análisis es determinar las ventajas e inconvenientes de cada una de las alternativas en relación al aprovechamiento de la localidad de los datos. Este análisis está alineado con el objetivo de esta tesis, centrado en maximizar la eficiencia del uso del modelo de programación MapReduce sobre una nube híbrida.

4.1 Implicaciones de la localidad de los datos

El paradigma de MapReduce está específicamente diseñado para facilitar un alto grado de paralelismo de datos, en donde cantidades masivas de datos de entrada se transforman de forma

paralela mediante una fase *Map* en un tipo de datos *(clave-valor)*, después de lo cual se agregan y procesan de forma paralela para obtener una salida mediante una fase *Reduce*. Toda esta cantidad de datos se encuentra normalmente distribuida a lo largo de los nodos participantes de la ejecución de Hadoop, y por tanto muchos de los datos deben ser leídos y/o escritos desde/hacia nodos remotos en la mayoría de los casos. Como es de esperar, esto conduce a un patrón de acceso intensivo de E/S altamente concurrente, que puede abrumar rápidamente la infraestructura de red con transferencias de datos remotas entre los nodos.

Para abordar este problema, Hadoop MapReduce implementa un alto grado de conocimiento de la localidad de los datos como característica clave. La capa de almacenamiento de Hadoop (HDFS) está diseñada para exponer la ubicación de los bloques de datos a la capa de procesamiento de datos (YARN). A través de este conocimiento, el planificador de Hadoop es capaz de mapear las tareas MapReduce lo más cerca posible a los datos que se van a utilizar, de forma que se minimice el tráfico de red relacionado con el almacenamiento. Sin embargo, en un escenario de *cloud-bursting* aparece una nueva vertiente en la gestión de los datos. En concreto, los datos de entrada se encuentran inicialmente en las MVs *on-premise*, por lo que éstos deben enviarse a las MVs *off-premise* antes de que puedan contribuir al cálculo. Un tipo de aplicaciones que podrían beneficiarse de esta nueva gestión de datos son las aplicaciones iterativas de análisis de datos, ya que éstas reutilizan los datos de entrada para refinar su resultado en las distintas iteraciones, y pueden aprovechar el potencial de una nube híbrida una vez enviados los datos de entrada a las MVs *off-premise*.

Por otra parte, las características intrínsecas del modelo MapReduce solo permiten aprovechar la localidad de los datos en la fase *Map*, ya que es la que generalmente realiza la lectura de los datos y puede ser planificada cerca de ellos. Por el contrario, cada tarea *Reduce* (compuesta por una fase *Shuffle*, *Sort* y *Reduce*) es responsable de la agregación y procesamiento de los resultados intermedios. Con lo cual, se enfrenta a una sobrecarga significativa en su primera fase previa de sincronización y recopilación de todos los datos intermedios obtenidos de la fase *Map* (es decir, la fase *Shuffle*). A gran escala, la fase *Shuffle* implica la transferencia simultánea de cantidades significativas de datos remotos entre todos los nodos, que a priori no puede evitarse, y conduce a la congestión de la infraestructura de red. Además, la capa de almacenamiento de Hadoop proporciona tolerancia a fallos mediante un sistema de replicación y distribución de bloques de datos, por tanto, la escritura de los resultados en la última fase *Reduce* implica una replicación de bloques de datos en nodos remotos, contribuyendo también a la congestión de la infraestructura de red.

Dado que las nubes híbridas se enfrentan a un problema de comunicación debido a un *enlace débil* de alta latencia y bajo rendimiento que separa las MVs *on-premise* y *off-premise*, se debe realizar un análisis detallado de aquellos aspectos que se ven afectados en la ejecución de una aplicación Hadoop MapReduce. Para ello, a continuación se analizan estos aspectos de forma detallada para cada una de las fases de ejecución de MapReduce en un despliegue de *cloud bursting*. Además, se proponen y analizan distintas alternativas para la gestión de los datos de entrada, enfocadas a la ejecución de aplicaciones MapReduce iterativas.

4.1.1 Fase Map

Por defecto, Hadoop intenta aprovechar la localidad de datos tanto como sea posible. Para ello, primero intenta planificar una tarea en el mismo nodo donde se encuentran los datos (*Node Local*). Cuando no hay slots disponibles para ejecutar una tarea *Map* en el mismo nodo que los datos de entrada, Hadoop planifica esa tarea *Map* en el mismo rack de nodos (*Rack Local*) de forma

preferente; y en caso de tampoco poder ser planificado, la tarea se planifica en cualquier otro nodo libre fuera del rack donde se encuentra el dato (*Off Switch*). En la Figura 4.1 se muestran los tres tipos de localidad de datos que intenta explotar Hadoop.

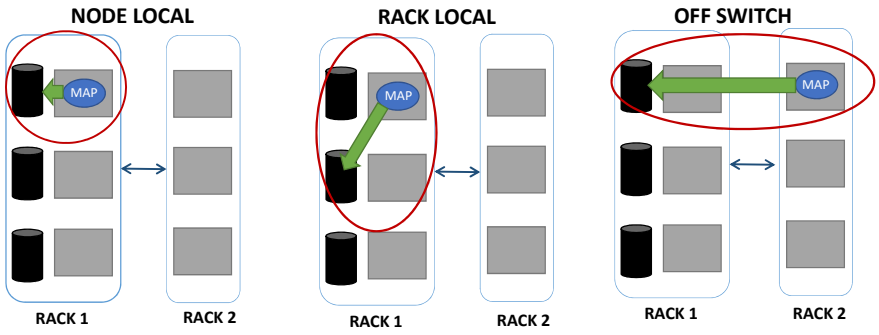


Figura 4.1: Tipos de planificación de tareas *Map* en Hadoop.

En una infraestructura de nube estándar, esta estrategia de planificación es adecuada asumiendo una penalización relativamente pequeña al ejecutar una tarea *Map* de forma remota: los enlaces de red a menudo se mantienen al mismo nivel que el rendimiento del disco, y hay relativamente pocas transferencias de datos remotas simultáneas a través de la red, ya que la mayoría de las tareas *Map* pueden beneficiarse de la localidad de los datos. Sin embargo, en un despliegue de *cloud bursting*, dado que los datos de entrada están presentes inicialmente solo en las MVs *on-premise*, cualquier tarea *Map* que se haya planificado para su ejecución *off-premise* debe tener acceso a los datos *on-premise*, lo que implica una transferencia de datos a través del *enlace débil*. Para ello, existen múltiples alternativas de acceso a estos datos.

Una opción obvia es simplemente dejar los datos de entrada en las MVs *on-premise* y transferirlos bajo demanda a las MVs *off-premise*. Con esta opción, si hay una cantidad considerable de MVs *off-premise*, entonces se planifican todas las tareas *Map* posibles cerca de los datos en las MVs *on-premise* hasta cubrir su capacidad total, pero el resto de tareas *Map* se planifican *Off Switch* para equilibrar el sistema, es decir, en la parte *off-premise* de la infraestructura (independientemente de la iteración). Por tanto, dichas tareas no podrán beneficiarse de la localidad de datos, teniendo que acceder siempre a los mismos a través del *enlace débil*. Esto provoca una fuerte degradación del rendimiento cuando los datos de entrada son de un tamaño considerable, creando un fuerte desequilibrio en las ondas de ejecución y una alta congestión del *enlace débil*, lo que a su vez tiene un impacto global negativo puesto que la fase *Reduce* necesita esperar la finalización de la fase *Map*.

Otra opción es migrar una réplica de los datos de entrada de las MVs *on-premise* a las MVs *off-premise* antes de ejecutar la aplicación MapReduce, de forma que éstos se encuentren presentes *off-premise* en tiempo de ejecución de la aplicación. En este caso, la estrategia del planificador original se comportará de manera similar a como si se ejecutara en un único clúster, ya que la mayoría de las tareas *Map* se beneficiarán de la localidad de los datos, y el *enlace débil* no tendrá sobrecarga en dicha fase al evitar el problema de enviar repetidamente el mismo bloque de datos a través de dicho enlace. Sin embargo, el proceso de migración de datos a las MVs *off-premise* es un proceso que consume un tiempo que se suma al tiempo de ejecución general. Además, también conduce a una infrautilización de los recursos *on-premise* y *off-premise* durante dicho proceso.

Para evitar esperar a que finalice el proceso de migración, una tercera opción es utilizar una migración asíncrona. Utilizando este enfoque, los datos de entrada se envían a las MVs *off-premise* al mismo tiempo que se ejecutan las tareas *Map* y *Reduce*. En este caso, la estrategia de planificación por defecto tiene una mayor probabilidad de aprovechar la localidad de los datos en las tareas *Map* planificadas *off-premise*, ya que algunos bloques de datos de entrada ya pueden estar presentes *off-premise* cuando se planifica una tarea *Map*. Sin embargo, durante el proceso de migración todavía puede darse la situación de que tareas *Map* planificadas *off-premise* no dispongan localmente del dato. Entonces surge una situación similar a la primera de las opciones, en la que los datos están únicamente *on-premise* aunque en menor grado. Por esta razón, es importante explorar estrategias de planificación alternativas cuando se utiliza la migración asíncrona para poder evitar este último comportamiento.

4.1.2 Fase Shuffle, Sort y Reduce

Una vez finalizada una tarea *Map*, su resultado se escribe en un archivo temporal en el disco local del nodo donde se ejecutó. Estos datos intermedios se recogen de todos los nodos a través de las tareas *Reduce* durante la fase *Shuffle* (fase intermedia entre las fases *Map* y *Reduce*). A gran escala, esto conduce a un patrón de comunicación todos a todos, como se muestra en la Figura 4.2, de tareas *Map* a tareas *Reduce*, que puede sobrecargar la red y en particular el *enlace débil*. Esta sobrecarga no puede ser evitada a priori, ya que los datos intermedios están presentes tanto *on-premise* como *off-premise*. Sin embargo, la planificación de las tareas *Map* afecta a la distribución de los datos intermedios, lo que a su vez afecta en la forma en la que se sobrecarga el *enlace débil*.

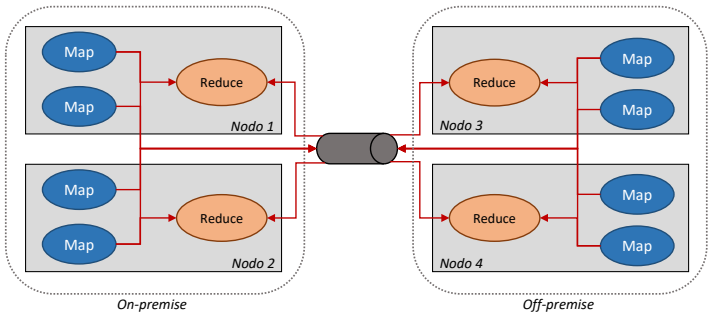


Figura 4.2: Patrón de comunicación de tareas *Map* a tareas *Reduce*.

El aspecto computacional de una tarea *Reduce*, es decir, la clasificación (fase *Sort*) y aplicación de la agregación definida por el usuario (fase *Reduce*), no se ve afectado por el *enlace débil*. Estos procesos se realizan localmente en los nodos implicados donde se recogen los datos intermedios de las tareas *Map*. Sin embargo, una vez que se genera la salida de la fase *Reduce* con éxito, ésta se escribe normalmente en la capa de almacenamiento de Hadoop (HDFS). Si HDFS se despliega únicamente en las MVs *on-premise*, entonces se produce de nuevo una saturación del *enlace débil*, ya que cada tarea *Reduce* planificada *off-premise* tendrá que transmitir toda su salida a través del *enlace débil*. Por el contrario, si HDFS se despliega tanto *on-premise* como *off-premise*, cada bloque de datos escrito en HDFS deberá ser replicado en ambas partes para garantizar la característica de tolerancia a fallos implementada en HDFS, con lo que tampoco se podrá evitar una posible congestión del *enlace débil*. Por tanto, la extensión de HDFS a la parte *off-premise* tampoco ofrece grandes ventajas.

4.2 Rack Awareness como característica clave

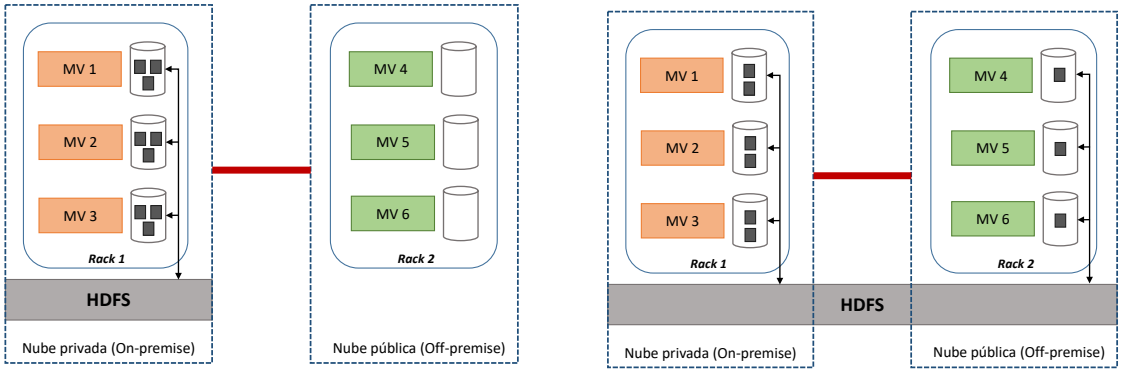
HDFS utiliza la replicación de datos como una característica esencial para proporcionar tolerancia a fallos y facilitar la planificación de tareas cerca de los datos que se van a utilizar en las mismas. Para implementar esta replicación de forma más eficiente, HDFS integra actualmente una característica denominada *Rack Awareness* [49], que se basa en el conocimiento previo de la jerarquía de nodos y racks en la que se va a ejecutar Hadoop, partiendo del supuesto de que los nodos que se hallan dentro de un mismo rack tienen un ancho de banda de red un orden de magnitud superior al que puedan tener nodos situados en racks distintos. Por tanto, se trata de crear grupos lógicos de nodos pertenecientes al mismo rack, para que HDFS sea consciente de la arquitectura de red y pueda realizar una distribución más equilibrada de los datos. De esta forma, el planificador de Hadoop puede realizar una planificación más adecuada y eficiente de las tareas con el fin de obtener un mejor rendimiento y menor congestión de la red. Además, una de las características clave de *Rack Awareness* es que garantiza que al menos una réplica de los datos se hallará en cada grupo lógico de nodos (racks), obteniendo así una mayor tolerancia a fallos en caso de caída de un rack entero de nodos.

Cuando un nuevo nodo se une a HDFS en un determinado rack, posee un espacio de almacenamiento inicialmente vacío. Esto conduce a una distribución desequilibrada de datos, donde los nodos más antiguos poseen más carga de datos que los nodos agregados más recientemente. Esto provoca una mayor utilización de los nodos más cargados, produciendo una utilización ineficiente de la localidad de datos. Para ello, HDFS emplea un mecanismo de **reequilibrio** que intenta redistribuir las réplicas de bloques de datos de acuerdo con la configuración impuesta por *Rack Awareness*, con la restricción de que al menos debe haber una réplica de los datos en cada uno de los racks.

Esta característica es la que aprovechamos para el contexto de las nubes híbridas en un despliegue de *cloud bursting*. Inicialmente definimos dos grupos de racks en HDFS, uno para las MVs *on-premise* y otro para las MVs *off-premise*. Después, cuando iniciamos el proceso de *cloud bursting* añadiendo más MVs *off-premise* al despliegue de Hadoop, extendemos la capa de almacenamiento HDFS a las MVs *off-premise*, provocando una alerta en el sistema HDFS dado que no se cumple la condición de haber al menos una réplica de todos los bloques de datos en cada uno de los racks definidos en el fichero de configuración de *Rack Awareness*. Esto conduce efectivamente a la migración automática y transparente de una réplica de cada bloque de datos almacenado *on-premise* hacia las MVs *off-premise*. Este reequilibrio puede ser de dos tipos: bloqueante (lo que provoca que no se inicie la ejecución de la aplicación MapReduce hasta que haya finalizado el proceso de migración) o asíncrono (que permite la ejecución de la aplicación al mismo tiempo que se realiza la replicación).

En la Figura 4.3 se muestra un ejemplo de extensión de la capa de almacenamiento HDFS utilizando la característica *Rack Awareness* con tres bloques de datos y una configuración de réplica tres (situación inicial), en la cual se realizaría la migración automática de al menos una réplica de los bloques *on-premise* a los nodos *off-premise* (situación final). En este ejemplo la configuración de *Rack Awareness* sería la siguiente:

```
Rack1: MV1, MV2, MV3
Rack2: MV4, MV5, MV6
```



(a) Situación inicial.

(b) Situación final.

Figura 4.3: Ejemplo de evolución de *Rack Awareness* para una nube híbrida con 3 bloques de datos y réplica 3.

La principal ventaja de este enfoque es que minimiza la cantidad de datos transferidos *off-premise* (una única réplica) para lograr el máximo potencial de explotación de la localidad de datos. Por otra parte, es una solución no invasiva que funciona con la versión original de HDFS.

4.3 Estrategias para gestionar la localidad de los datos

A partir del análisis presentado en la sección anterior, en esta sección se proponen una serie de estrategias que implementan las distintas alternativas para la gestión de datos de entrada, presentadas en la sección 4.1.1, a través de la utilización de la característica *Rack Awareness*. El objetivo de estas estrategias es el análisis y evaluación comparativa sobre cuál es la gestión más adecuada de localidad de datos para un escenario de *cloud bursting*. Para cada estrategia se describe el detalle de su implementación en Hadoop y HDFS, para su posterior evaluación experimental y análisis.

Se parte de una estrategia inicial simple, que servirá como base comparativa para el resto de estrategias, en la cual no se extiende el sistema de almacenamiento HDFS a las MVs *off-premise*. Las estrategias que se plantean para mejorar la localidad de datos se basan en la extensión de la capa de almacenamiento HDFS a la parte *off-premise*, haciendo uso de la característica *Rack Awareness*, con el objetivo de que en algún momento exista una copia completa de los datos en cada una de las partes de la infraestructura híbrida. De esta forma, *Rack Awareness* obliga al sistema a reequilibrar los bloques de datos iniciando un proceso de migración de al menos una réplica de todos los bloques de datos a las recién adquiridas MVs *off-premise*. En este caso se plantean tres estrategias diferentes: dos de ellas hacen uso de las políticas por defecto del planificador de Hadoop, mientras que para la última se ha creado una nueva política de planificación a través de la modificación del planificador de Hadoop.

Así pues, en esta sección va a presentar:

- Una estrategia simple basada en no extender HDFS a las MVs *off-premise*:
  - *Sin HDFS off-premise*.

- Tres estrategias enfocadas en mejorar la localidad de datos:
  - *Reequilibrio bloqueante*.
  - *Reequilibrio asíncrono*.
  - *Reequilibrio asíncrono con rack local* (que implica el diseño de una nueva política de planificación).

4.3.1 Sin HDFS off-premise

Esta estrategia es la más simple y servirá como base comparativa para las demás. Consiste en un despliegue de *cloud bursting* en el que se añaden nuevas MVs de cómputo *off-premise* a la configuración actual de Hadoop ubicada *on-premise*, pero sin la extensión de HDFS a la parte *off-premise*, únicamente la capa de procesamiento YARN de Hadoop. Es decir, las nuevas MVs adquiridas a través de una núbe pública solo servirán para cálculo y no para almacenamiento. Por tanto, cada vez que una tarea sea planificada *off-premise*, ésta deberá leer y escribir en HDFS desde las MVs *on-premise* a través del *enlace débil*. Sin embargo, la aplicación puede empezar de inmediato su ejecución nada más termine el despliegue de *cloud bursting*. El tiempo de finalización global coincide en este caso con el tiempo de ejecución de la aplicación, ya que no hay que esperar a ningún proceso inicial para lanzar la aplicación.

Las ventajas que presenta esta estrategia son:

- La aplicación MapReduce puede ejecutarse inmediatamente después de un despliegue de *cloud bursting*.
- Permite solapar la E/S con el cálculo.
- No requiere ninguna modificación de Hadoop.

Por contra, como inconveniente nos encontramos con que todas las tareas *Map* planificadas en las MVs *off-premise* deben leer repetidamente los datos de entrada desde las MVs *on-premise* a través del *enlace débil*, así como escribir el resultado de cada tarea *Reduce* planificada *off-premise* en las MVs *on-premise*.

4.3.2 Reequilibrio bloqueante

Esta estrategia corresponde a un escenario de *cloud bursting* en el que, tanto la capa de almacenamiento HDFS como su capa de procesamiento YARN de Hadoop se extienden a las nuevas MVs *off-premise*. De esta forma, dada la característica de *Rack Awareness*, todos los bloques de datos de entrada son migrados a las MVs *off-premise*, de forma que se realiza un reequilibrio del sistema. La característica propia de esta estrategia es que la aplicación debe esperar a que todos los bloques de datos de entrada hayan sido migrados antes de empezar su ejecución. Por tanto, una vez finalizado el proceso, ambas nubes dispondrán de una localidad de datos muy equilibrada que podrá ser aprovechada por la aplicación. La principal desventaja de esta estrategia es que el tiempo inicial de espera debido a la migración de datos depende del tamaño de los datos de entrada y del ancho de banda del *enlace débil*. Por tanto, puede que este tiempo haga mermar el rendimiento con

respecto al tiempo global de ejecución de la aplicación. En este caso, al tiempo de ejecución de la aplicación hay que sumarle el tiempo de espera del proceso de migración, ya que forma parte del tiempo de finalización global.

Esta estrategia presenta como ventajas el hecho de que la localidad de los datos puede ser explotada desde el inicio de la aplicación y además no requiere ninguna modificación de Hadoop. Como inconveniente, se debe esperar a tener una copia completa de los datos en las MVs *off-premise* antes de iniciar la ejecución de la aplicación.

4.3.3 Reequilibrio asíncrono

En esta estrategia, al igual que en la de *Reequilibrio bloqueante*, también se extiende el sistema HDFS y YARN a las MVs *off-premise*. Sin embargo, a diferencia de la estrategia anterior, ahora la aplicación MapReduce se inicia al mismo tiempo que el proceso de migración de los datos de entrada. Esta estrategia conduce a un escenario inicial en donde el proceso de migración de los datos, así como el tráfico propio de la ejecución de la aplicación, compiten por el *enlace débil*. En este caso, durante el proceso de migración puede que ocurran dos situaciones en cuanto a la localidad de los datos *off-premise*. Por una parte puede que se planifique una tarea *Map off-premise* con su bloque de datos necesario ya migrado. Por el contrario, puede que se planifique sin que se disponga de su bloque de datos de entrada. En esta última situación se presenta un escenario similar al de la estrategia *Sin HDFS off-premise*, en el cual algunas de las tareas planificadas *off-premise* necesitan leer su bloque de datos de las MVs *on-premise*, creando así un tráfico y una espera innecesaria de ejecución debido a una mala política de planificación, ya que dichos datos serán migrados más adelante. En este caso, dado que el inicio de la ejecución de la aplicación coincide con el de inicio del proceso de migración, el tiempo de finalización global coincide con el tiempo de ejecución de la aplicación.

Las principales ventajas de esta estrategia son las siguientes:

- La aplicación MapReduce puede ejecutarse inmediatamente después de un despliegue de *cloud bursting*.
- Muchas de las tareas *Map* pueden beneficiarse de la localidad de los datos *off-premise* incluso durante el proceso de migración.
- Permite solapar la E/S con el cálculo.
- No requiere ninguna modificación de Hadoop.

Como principal inconveniente cabe destacar que una mala política de planificación puede conducir a un escenario en el que se planifiquen tareas *Map* en las MVs *off-premise* sin que se encuentre su bloque de datos de entrada presente, produciendo una sobrecarga del *enlace débil* y una degradación del rendimiento general.

4.3.4 Reequilibrio asíncrono con rack local

El planificador por defecto de Hadoop utiliza la localidad de los datos solo como un mecanismo de coincidencia preferencial entre tareas *Map* y slots libres. Sin embargo, como se ha descrito

anteriormente, en la estrategia de *Reequilibrio asíncrono* con la política de planificación de tareas *Map* de Hadoop por defecto, se puede producir un escenario donde una tarea *Map* sea planificada *off-premise* antes de migrar la réplica de su correspondiente bloque de datos de entrada. Esto desencadena una solicitud del bloque de datos a las MVs *on-premise* para poder ser procesado *off-premise*, provocando una doble transferencia de datos del mismo bloque. Este comportamiento produce un sobrecoste en la ejecución de las tareas *Map*, que se puede apreciar fácilmente en la Figura 4.4, en donde se muestra una traza de la planificación de las tareas *Map* y *Reduce* de la aplicación MapReduce iterativa K-Means, a través de nuestra herramienta.

(a) Iteración 1. Tiempo de finalización: 620 segundos.

(b) Iteración 10. Tiempo de finalización: 308 segundos.

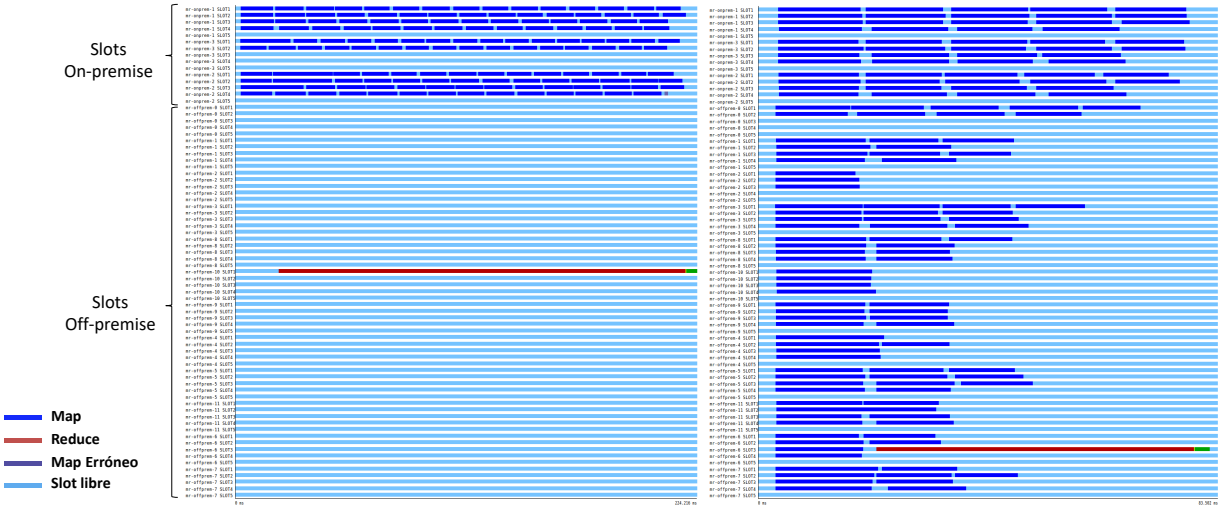
Figura 4.4: Ejemplo de planificación por defecto de Hadoop en una nube híbrida con *enlace débil* de 100 Mbps. Iteraciones 1 y 10 de K-Means.

Como se puede observar, en la primera iteración (Figura 4.4 (a)) las tareas *Map* planificadas en las MVs *off-premise* tienen un tiempo de ejecución mucho mayor que las planificadas *on-premise*. Esto se debe a que las MVs *off-premise* no tienen ningún bloque de datos aún migrado, y por tanto todas las tareas *Map* planificadas *off-premise* deben esperar a la lectura de su bloque a través del *enlace débil* (en este caso de 100 Mbps), produciendo un sobrecoste en el tiempo de finalización de la iteración. En la décima iteración (Figura 4.4 (b)) ya se puede apreciar una mejor utilización de la localidad de los datos al tener gran parte de los datos ya migrados, pero aun así hay ciertas tareas *Map* que no disponen de su bloque de datos y producen una considerable reducción del rendimiento en la iteración.

En este caso, podría ser beneficioso retrasar la planificación de la tareas *Map off-premise* que no dispongan de su bloque de datos migrado, es decir, no planificar ninguna tarea *Map off-premise* hasta que su bloque de datos esté migrado.

Con este fin, proponemos una nueva política de planificación que fuerce a que las tareas solo puedan planificarse si tienen localidad dentro de su rack (tipo de planficación *Rack Local* o *Node Local*), de forma que pueda adaptarse mejor al despliegue de *cloud bursting*: una tarea *Map* nunca se planificará en tipo *Off-Switch* (fuera del rack) si el bloque de datos que necesita procesar no está replicado en el rack de destino. Para ello, se ha implementado esta nueva política en Hadoop realizando una modificación del *ResourceManager* para hacer uso de la opción *relaxLocality* en todos los casos. Esta opción se puede modificar a través del código fuente de una aplicación Hadoop,

sin embargo, la idea es forzar a cualquier aplicación a que utilice esta opción para optimizar su rendimiento en un entorno de nube híbrida. Por lo tanto, se trata de una modificación intrusiva de Hadoop que requiere que el usuario utilice una versión personalizada. Los efectos de esta nueva política se pueden apreciar a través de la Figura 4.5, en donde se muestra el efecto de esta nueva planificación de Hadoop, la cual evita planificar tareas *Map* que no puedan beneficiarse de la localidad de los datos.



(a) Iteración 1. Tiempo de finalización: 224 segundos.

(b) Iteración 10. Tiempo de finalización: 83 segundos.

Figura 4.5: Ejemplo de planificación forzada a rack local de Hadoop en una nube híbrida con *enlace débil* de 100 Mbps. Iteraciones 1 y 10 de K-Means.

El beneficio de esta nueva planificación se obtiene ya desde la primera iteración, puesto que tal y como se observa en las Figuras 4.4 (a) y 4.5 (a) se pasa de un tiempo de ejecución de 620 s a 224 s. Este beneficio se sigue consolidando en las sucesivas iteraciones, ya que la planificacion de tareas permite adaptarse mejor al proceso de migración de datos, tal y como se muestra en la Figura 4.5 (b), donde ya en la décima iteración se han migrado gran parte de los bloques de datos a las MVs *off-premise*. Por tanto, en dicha iteración muchas de las tareas *Map* pueden beneficiarse de la localidad de los datos sin que ninguna tarea *Map* que no disponga de los datos *off-premise* se planifique y perjudique el rendimiento total de la iteración.

En la Figura 4.5 (b) se puede observar que las tareas *Map* planificadas en las MVs *off-premise* tienen un tiempo de ejecución similar a las planificadas *on-premise*. Por otra parte, el tiempo de finalización de dicha iteración se ha reducido de 308 s a 83 s.

En resumen, se propone una nueva estrategia que hemos denominado *Reequilibrio asíncrono con rack local*, similar a la de *Reequilibrio asíncrono* pero con una nueva política de planificación forzada a rack local, que se adapta mejor a un despliegue de *cloud bursting*. Una vez más, el tiempo de finalización global de la aplicación coincide con el tiempo de ejecución de la aplicación.

Esta nueva estrategia ofrece las siguientes ventajas:

- La aplicación MapReduce puede ejecutarse inmediatamente después de un despliegue de *cloud bursting*.

- Todas las tareas *Map* pueden beneficiarse de la localidad de los datos *off-premise*, incluso durante el proceso de migración.
- Permite solapar la E/S con el cálculo.

Por contra, existen una serie de inconvenientes que hay que tener en cuenta:

- Necesita una modificación intrusiva de Hadoop.
- Aunque se ha optimizado la fase *Map*, las tareas *Reduce* no pueden beneficiarse de esta nueva política.

4.4 Evaluación

En esta sección se realiza una evaluación experimental de las cuatro estrategias descritas en la sección anterior, haciendo uso del entorno de experimentación descrito en la sección 2.4. Para ello, se han elegido dos aplicaciones iterativas reales, representativas de MapReduce, como PageRank y K-Means. Ambas aplicaciones han sido extraídas del benchmark para BigData *HiBench* [53] de Intel, utilizado en este ejemplo para la creación de los datos de entrada así como para su ejecución.

Se han elegido K-Means y PageRank para analizar diferentes distribuciones de tareas MapReduce, ya que K-Means es una aplicación en la que el peso de la ejecución se encuentra en la fase *Map*, por lo que denominamos a este tipo de aplicación *Map*-intensiva. Por el contrario, PageRank es una aplicación más equilibrada, que consta de varios trabajos por iteración, algunos de ellos con más peso en la fase *Map*, con lo cual son de tipo *Map*-intensivo, y otros con más peso en la fase *Reduce*, por lo que son de tipo *Reduce*-intensivo. A continuación se realiza una breve descripción de ambas aplicaciones.

K-Means

K-Means [17] es una aplicación ampliamente utilizada para la cuantificación vectorial en el procesamiento de señales, análisis de clústeres en minería de datos, clasificación de patrones y extracción de rasgos para el aprendizaje automático, etc. Se basa en un refinamiento iterativo: cada iteración intenta mejorar el particionamiento de un vector multidimensional en k grupos de tal manera que la suma de los cuadrados de las distancias entre todos los vectores del mismo grupo y su media se minimice. Este proceso se repite hasta que la mejora obtenida durante una iteración es menor que un epsilon predeterminado. K-Means se puede paralelizar de forma eficiente y escalable utilizando MapReduce [109]: cada iteración lee una gran cantidad de datos de entrada (que permanece inmutable durante las iteraciones) mientras que la salida es una pequeña cantidad de datos intermedios y de salida final. Desde una perspectiva de gestión de datos, es un buen ejemplo de aplicación *Map*-intensiva que reutiliza los datos de entrada iniciales en cada iteración.

En este caso, se ha utilizado el benchmark *HiBench* de Intel para la creación de sus datos de entrada. Su implementación MapReduce se encuentra en la distribución *Mahout* [41], una popular colección de algoritmos de aprendizaje automático de código abierto implementada en MapReduce.

PageRank

PageRank [18] es un algoritmo de análisis de enlaces que asigna un peso numérico a cada elemento de un conjunto de documentos con hipervínculos (por ejemplo, WWW) con el propósito de cuantificar su importancia relativa dentro del conjunto. Es ampliamente utilizado en los motores de búsqueda web para calcular el posicionamiento de páginas web en función del número de enlaces de referencia. Su naturaleza iterativa es más compleja e implica dos trabajos sucesivos de MapReduce: (1) una fase de salida intensiva en la que la fase *Reduce* genera el doble de datos que los datos de entrada leídos por la fase *Map*; y (2) una fase *Shuffle* intensiva en donde la salida de las tareas *Map* es igual a la entrada. Por lo tanto, PageRank es una buen ejemplo de aplicación equilibrada, puesto que es a la vez *Map*-intensiva y *Reduce*-intensiva, generando una gran cantidad de datos intermedios que no se reutilizan.

4.4.1 Escenarios

Para realizar una comparación más clara en cuanto a la escalabilidad de las aplicaciones con las diferentes estrategias propuestas, se presentan tres escenarios de experimentación:

1. **On-premise original:** corresponde al caso en el que la aplicación se ejecuta con los recursos *on-premise* originales. Este escenario se utiliza como línea de base para el límite superior de tiempo, mostrando el rendimiento cuando se utiliza una infraestructura reducida sin ningún tipo de recurso remoto extra. Sería el caso en el que la aplicación obtendría su rendimiento más bajo y necesitaría de recursos extra para mejorar su rendimiento.

2. **On-premise extendido:** corresponde a un caso ideal en el que la infraestructura local tuviera una mayor capacidad de recursos para permitir que la aplicación se ejecutara lo más rápido posible. Este escenario se utiliza como un límite inferior de tiempo, es decir, muestra lo que sucedería en un escenario ideal en el que el usuario no tuviera limitaciones de costes y pudiera invertir en recursos adicionales en sus instalaciones para lograr el máximo rendimiento en lugar de adoptar una solución híbrida.

3. **Nube híbrida mediante *cloud bursting*:** corresponde a una solución híbrida en la que las MVs *on-premise* se complementan con MVs *off-premise* a través de un despliegue de *cloud bursting*. En este escenario se utiliza un límite configurable para el ancho de banda del *enlace débil* de forma que nos permita controlar el tráfico entre las nubes, simulando así una situación real. En este caso se experimenta con dos configuraciones representativas de ancho de banda, 1 Gbps y 100 Mbps. Serían correspondientes al caso en el que el usuario decide pagar por un acceso de alta velocidad a la nube (es decir, enlaces rápidos dedicados) frente a un acceso regular. En este escenario, y para cada uno de los dos anchos de banda del *enlace débil*, se evalúan las cuatro estrategias propuestas.

4.4.2 Resultados globales

Para ejecutar las dos aplicaciones descritas en la sección 4.4 se ha usado el generador de datos de entrada incluido en *Mahout* para K-Means y el generador de *HiBench* para PageRank. En el caso de K-Means se han generado 20 GB de datos de entrada y se ha configurado la aplicación para

que se ejecute siempre con 30 iteraciones, de forma que permita repetir la experimentación con otras configuraciones y comparar resultados. En el caso de PageRank, se han generado 2,8 GB de datos de entrada y se ha configurado para que la aplicación termine cuando realice 10 iteraciones (realmente son 5 iteraciones compuestas por 2 trabajos cada una, pero por simplicidad se obvia este detalle en este capítulo).

En primer lugar, ejecutamos las aplicaciones sobre los escenarios de línea de base (*on-premise original* y *on-premise extendido*), para extraer el límite superior e inferior del tiempo de finalización total de cada aplicación en una configuración no híbrida, consistente en un solo despliegue de nube de 3 y 15 MVs, respectivamente. Los resultados obtenidos se resumen en la Tabla 4.1.

Aplicación	MVs On-premise	Tiempo de finalización (s)
KMeans	3	5801,15
	15	1516,20
PageRank	3	3150,08
	15	777,82

Tabla 4.1: Tiempo de finalización para los escenarios de línea de base *on-premise* original (3 MVs) y extendido (15 MVs).

Como se puede observar a partir de los resultados de la Tabla 4.1, ambas aplicaciones poseen un alto grado de escalabilidad. Cuando el número de MVs aumenta de 3 a 15 hay una mejora de rendimiento de un 75 % en ambos casos. Por lo tanto, para que una configuración híbrida sea viable, tiene que obtener un rendimiento al menos tan bueno como el límite superior (3 MVs *on-premise*) e idealmente acercarse lo máximo posible al límite inferior (15 MVs *on-premise*).

A continuación, ejecutamos ambas aplicaciones sobre el escenario de *Nube híbrida mediante cloud bursting*, en el que se parte de 3 MVs *on-premise*, las cuales se complementan con 12 MVs *off-premise* a través de un despliegue de *cloud bursting*. Las Tablas 4.2 y 4.3 muestran una comparativa de los resultados obtenidos utilizando las cuatro estrategias descritas en la sección 4.3, incluyendo los resultados para las dos configuraciones de ancho de banda propuestas para el *enlace débil*: 100 Mbps y 1 Gbps. En las tablas se incluye el tiempo de migración de los datos y el tiempo de ejecución de la aplicación de forma separada. Hay que tener en cuenta que, en el caso de *Reequilibrio bloqueante*, el tiempo de finalización es la suma del tiempo de ejecución de la aplicación y la duración de la migración. Sin embargo, en el caso de *Reequilibrio asíncrono* y de *Reequilibrio asíncrono con rack local*, la migración es un proceso paralelo y transparente al usuario que se superpone con el tiempo de ejecución de la aplicación. Por lo tanto, el tiempo de finalización es idéntico al tiempo de ejecución de la aplicación. En este caso, es interesante observar el tiempo de migración como una indicación de la duración de la interferencia y la competencia entre la aplicación y la migración de datos, que es una causa de degradación del rendimiento.

Analizando los resultados obtenidos en la aplicación K-Means (Tabla 4.2), se puede observar que la mejor estrategia en un *enlace débil* de alta velocidad, en nuestro caso 1 Gbps, es *Reequilibrio asíncrono con rack local*. En este caso, el tiempo total de finalización es muy cercano al límite inferior. Por otra parte, tanto la estrategia de *Reequilibrio bloqueante* como la de *Reequilibrio asíncrono* presentan una degradación del rendimiento del 10 % con respecto al límite inferior, lo que indica que la interferencia es relativamente pequeña pero sigue siendo significativa, lo que deja margen para mejorar al forzar la localidad de rack. En este caso, incluso con la estrategia *Sin HDFS off-premise* se puede obtener una mejora del 40 % con respecto al límite superior del caso base.

Estrategia	Tiempo de migración (s)	Ejecución de la aplicación (s)	Tiempo de finalización (s)
1 Gbps			
Sin HDFS Off-premise	N/A	3485,87	3485,87
Reequilibrio Bloqueante	216,70	1464,97	1681,67
Reequilibrio Asíncrono	396,90	1647,93	1647,93
Reequilibrio Asínc. Rack Local	221,1	1571,47	1571,47
100 Mbps			
Sin HDFS Off-premise	N/A	16667,70	16667,70
Reequilibrio Bloqueante	2507,50	1803,20	4310,70
Reequilibrio Asíncrono	6160,40	6462,23	6462,23
Reequilibrio Asínc. Rack Local	2688,6	2713,86	2713,86

Tabla 4.2: Rendimiento de K-Means en un despliegue de *cloud bursting* (3 MVs *on-premise* y 12 MVs *off-premise*). Comparativa de las cuatro estrategias haciendo uso de *enlaces débiles* de 1 Gbps y 100Mbps. Menor es mejor.

Sin embargo, al cambiar a un ancho de banda reducido de *enlace débil* entre *on-premise* y *off-premise* (en nuestro caso 100 Mbps), la limitación del *enlace débil* tiene un impacto más pronunciado. La gran cantidad de transferencias, así como la cantidad de veces que cada dato se transmite debido a la estrategia de *Sin HDFS off-premise*, conducen a un tiempo de finalización mucho mayor que el límite superior. Esto indica que esta opción no es viable para anchos de banda reducidos del *enlace débil*. En la estrategia de *Reequilibrio asíncrono* se puede observar que se produce una larga duración del proceso de migración de los datos (6160 s concretamente), hasta el punto en que la migración se superpone casi totalmente con el tiempo de ejecución de la aplicación (6462 s), creando una degradación del rendimiento que está por encima del límite superior (5801 s), haciendo por tanto inviable esta opción. En este caso, evitar la interferencia es crítico, como lo demuestra el hecho de que incluso la estrategia de *Reequilibrio bloqueante* funciona mucho mejor (4310 s), pudiendo llegar a mejorar su tiempo de finalización en un 25 % con respecto al límite superior. Sin embargo, la estrategia de *Reequilibrio asíncrono con rack local* que hemos propuesto es la mejor (2713 s): gracias a la restricción de localidad de las tareas *Map off-premise*, la interferencia se reduce mucho, lo que le permite rendir mucho mejor que la estrategia de *Reequilibrio bloqueante* y mejorar su tiempo de finalización en un 55 % con respecto al límite superior.

En el análisis de los resultados de la aplicación PageRank (Tabla 4.3), observamos que con un ancho de banda del *enlace débil* de 1 Gbps obtenemos un tiempo de finalización (875 s) muy cercano al límite inferior, tan solo un 12 % más lento que el límite inferior (777 s) en el mejor de los casos. Como era de esperar, la estrategia *Sin HDFS off-premise* es la que obtiene el peor rendimiento (1232 s). Sin embargo, esta vez la estrategia ganadora es la de *Reequilibrio asíncrono*, seguida por la de *Reequilibrio bloqueante*, y finalmente la de *Reequilibrio asíncrono con rack local*. Este comportamiento puede explicarse por el hecho de que el proceso de migración de datos estresa menos el *enlace débil* (la cantidad de datos de entrada es mucho menor y se migra rápidamente), lo que conduce a una situación en la que no se obtiene ninguna ventaja, ni demorando o limitando la planificación de las tareas *Map* en las MVs *off-premise*, ni esperando a la finalización del proceso de migración para ejecutar la aplicación.

Sin embargo, al cambiar a un ancho de banda reducido del *enlace débil* entre nubes (100 Mbps), la limitación del *enlace débil* sí que tiene un impacto pronunciado al igual que ocurría en K-Means.

Estrategia	Tiempo de reequilibrio (s)	Ejecución de la aplicación (s)	Tiempo de finalización (s)
1 Gbps			
Sin HDFS Off-premise	N/A	1232,77	1232,77
Reequilibrio Bloqueante	86,00	838,95	924,90
Reequilibrio Asíncrono	88,00	875,37	875,37
Reequilibrio Asínc. Rack Local	87,75	944,70	944,70
100 Mbps			
Sin HDFS Off-premise	N/A	5712,00	5712,00
Reequilibrio Bloqueante	450,90	3552,60	4003,50
Reequilibrio Asíncrono	527,40	3582,00	3582,00
Reequilibrio Asínc. Rack Local	569,00	3555,00	3555,00

Tabla 4.3: Rendimiento de PageRank en un despliegue de *cloud bursting* (3 MVs *on-premise* y 12 MVs *off-premise*). Comparativa de las cuatro estrategias haciendo uso de *enlaces débiles* de 1 Gbps y 100Mbps. Menor es mejor.

Sin embargo, aunque prácticamente los tiempos de finalización son muy similares entre las estrategias de *Reequilibrio asíncrono* y la de *Reequilibrio asíncrono con rack local*, la estrategia de *Reequilibrio asíncrono* conduce a un menor tiempo de migración pero a un tiempo de finalización más largo; mientras que la estrategia de *Reequilibrio asíncrono con rack local* mejora el tiempo de finalización, pero conduce a un mayor tiempo de migración. La estrategia de *Reequilibrio bloqueante* es significativamente más lenta en este caso. De todas formas, hay que tener en cuenta que todas las estrategias son inviables como una solución práctica en este caso, ya que están todas por encima del límite superior (3150 s).

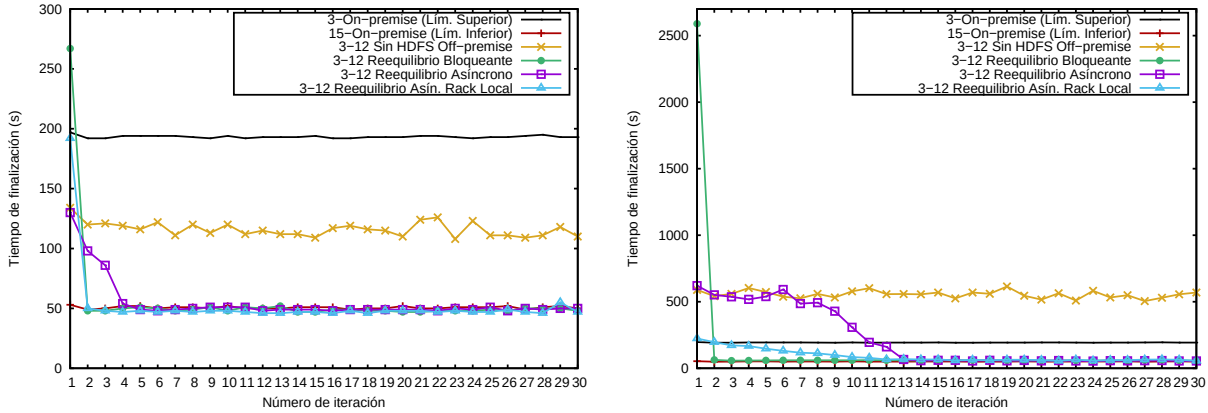
Esto puede ser explicado por el hecho de que PageRank ejecuta dos trabajos distintos por iteración, uno de tipo *Map*-intensivo y otro de *Reduce*-intensivo, lo que significa que en este último la fase *Map* tiene mucho menos impacto general y cualquier optimización en esta fase no afecta en gran medida al rendimiento total de la aplicación. En este tipo de aplicaciones el *enlace débil* crea un cuello de botella importante que no se puede evitar en anchos de banda limitados.

4.4.3 Análisis por iteración

Los resultados obtenidos se pueden comprender mejor al analizar con detalle el tiempo de finalización de cada iteración, o trabajo MapReduce, para cada estrategia con respecto al límite superior e inferior de base. Estos datos se han plasmado en la Figura 4.6 para K-Means y en la Figura 4.8 para PageRank. Cabe destacar que en la estrategia de *Reequilibrio bloqueante* el tiempo de finalización de la primera iteración incluye el tiempo de migración de los datos para realizar un análisis más justo, ya que en el resto de estrategias dicha migración se produce en paralelo o simplemente no se produce.

En el caso de K-Means, el uso de 1 Gbps de *enlace débil* (Figura 4.6(a)) conduce a una situación en la que la estrategia *Sin HDFS off-premise* es incluso más rápida que el límite superior, a pesar de la constante necesidad de solicitar datos a las MVs *off-premise* a través del *enlace débil*. Por otra parte, dado que la estrategia de *Reequilibrio bloqueante* espera a la migración completa de los datos antes de empezar con la ejecución de la aplicación, el tiempo de finalización por iteración

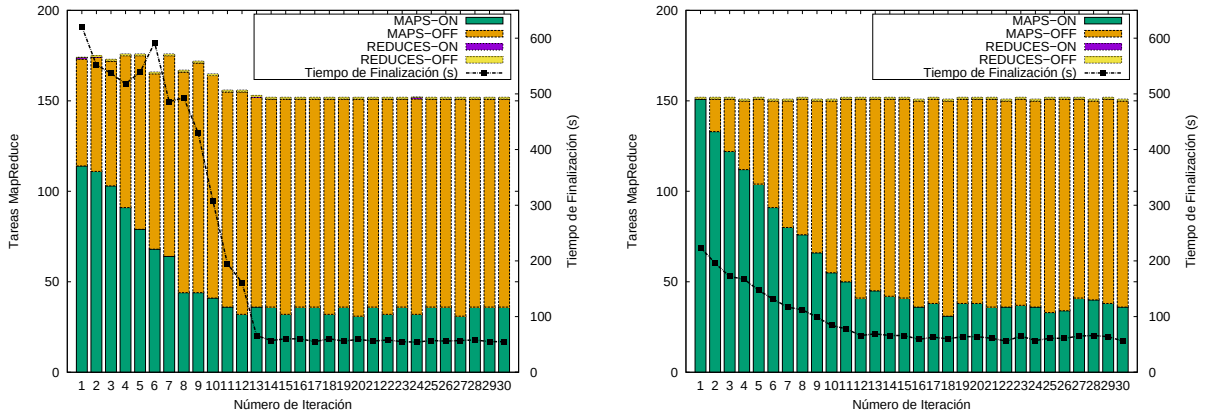
permanece constante excepto en la primera iteración (en la que se incluye el tiempo de migración), y se superpone con el límite inferior, que es el comportamiento esperado. Sin embargo, en el caso de *Reequilibrio asíncrono* y de *Reequilibrio asíncrono con rack local*, la interferencia causada por la migración de datos es claramente visible al principio, donde las iteraciones tardan mucho más en terminar en comparación con el resto del tiempo de ejecución. Es interesante observar que la estrategia de *Reequilibrio asíncrono con rack local* genera más interferencia al principio, pero solo para una única iteración, en comparación con la de *Reequilibrio asíncrono*, en la que la interferencia dura hasta la cuarta iteración. Por lo tanto, dentro de las estrategias de reequilibrio asíncrono, esto conduce a un mejor resultado global para el caso de *Reequilibrio asíncrono con rack local*.



(a) 1 Gbps de *enlace débil*.

(b) 100 Mbps de *enlace débil*.

Figura 4.6: K-Means: análisis comparativo del tiempo de finalización por iteración para las cuatro estrategias, con respecto a las líneas base de límite superior e inferior. Menor es mejor.



(a) Política de planificación de Hadoop por defecto. Estrategia de *Reequilibrio asíncrono*.

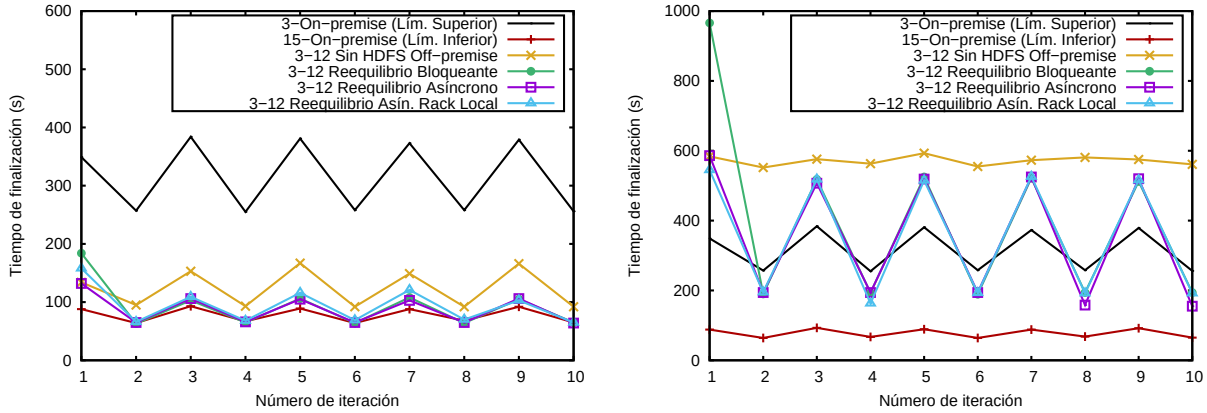
(b) Política de planificación de Hadoop forzada a rack local. Estrategia de *Reequilibrio asíncrono con rack local*.

Figura 4.7: K-Means: tiempo de finalización por iteración con respecto a la distribución de tareas MapReduce planificadas *on-premise* y *off-premise* para 100 Mbps.

Pasando al caso de un *enlace débil* de 100 Mbps (Figura 4.6(b)), se muestra claramente que la estrategia *Sin HDFS off-premise* es más lenta que el límite superior (3-On-premise). Este comportamiento es consistente durante toda la duración del tiempo de ejecución, y se debe a que no

se envía ningún dato *off-premise* para lograr localidad de datos. La estrategia de *Reequilibrio bloqueante* está constantemente cerca del límite inferior, pero empieza con un tiempo de finalización muy alto debido a la espera de migración de los datos, lo que permite que la estrategia de *Reequilibrio asíncrono* y la de *Reequilibrio asíncrono con rack local* la superen a nivel global. En el caso asíncrono, el efecto de la interferencia es más pronunciado, durando en ambos casos hasta la 13^a iteración. Además, el beneficio de *Reequilibrio asíncrono con rack local* también es claramente visible, explicando la diferencia observada en el tiempo de finalización global.

Para entender mejor las diferencias entre las estrategias de *Reequilibrio asíncrono* y de *Reequilibrio asíncrono con rack local* y sus distintas políticas de planificación, en la Figura 4.7 se muestra la distribución de tareas *Map* y *Reduce* que realiza Hadoop en el conjunto de MVs *on-premise* y *off-premise*, para el escenario de la aplicación K-Means con 100 Mbps de *enlace débil*. Nos hemos centrado en este escenario ya que exhibe la diferencia más pronunciada en el tiempo de finalización. Como puede observarse, la estrategia de *Reequilibrio asíncrono* (4.7(a)) planifica una gran cantidad de tareas *Map off-premise* desde el inicio, lo que lleva a una alta congestión del *enlace débil* y a un correspondiente aumento en el tiempo de finalización por iteración. Por el contrario, la estrategia de *Reequilibrio asíncrono con rack local* (4.7(b)) planifica casi todas las tareas *Map on-premise* durante la primera iteración, y gradualmente planifica más tareas *Map off-premise* a medida que progresa la migración de los datos. Gracias a este comportamiento, el tiempo de finalización por iteración converge al ideal mucho más rápido que en el caso de *Reequilibrio asíncrono*.



(a) 1 Gbps de *enlace débil*.

(b) 100 Mbps de *enlace débil*.

Figura 4.8: PageRank: análisis comparativo del tiempo de finalización por iteración para las cuatro estrategias, con respecto a las líneas base de límite superior e inferior. Menor es mejor.

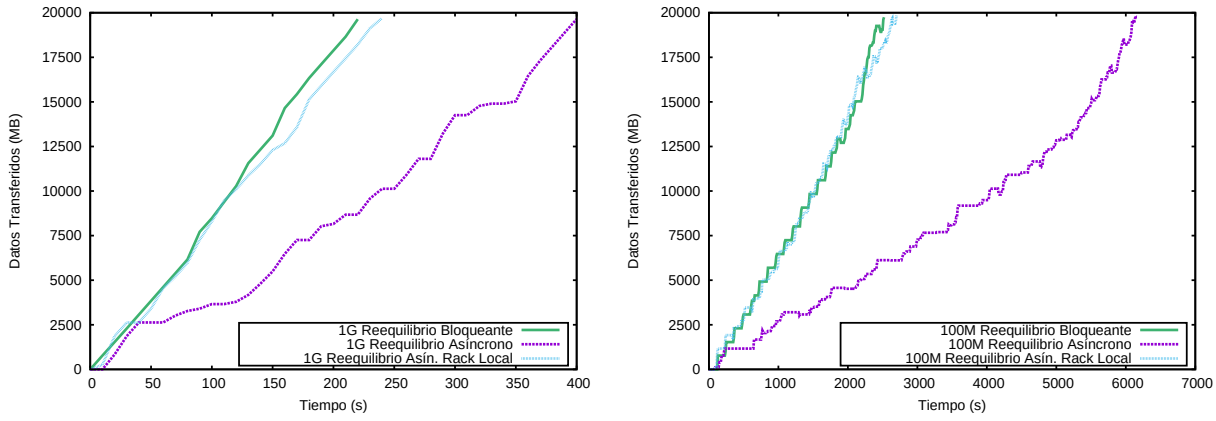
A diferencia de K-Means, en el caso de PageRank observamos un patrón de zigzag que corresponde a los dos tipos de trabajo que se alternan durante las iteraciones. Esto es visible tanto para el caso de 1 Gbps como el de 100 Mbps (Figuras 4.8(a) y 4.8(b)). En el caso de 1 Gbps, el comportamiento del límite inferior, el límite superior, y las estrategias de *Sin HDFS off-premise* y de *Reequilibrio bloqueante* es consistente durante todo el tiempo de ejecución. Tanto la estrategia de *Reequilibrio asíncrono* como la de *Reequilibrio asíncrono con rack local* siguen a la estrategia de *Reequilibrio bloqueante* muy de cerca después de la primera iteración, lo que significa que la diferencia observada se debe principalmente a la interferencia provocada por la migración de los datos de la primera iteración. En el caso de 100 Mbps, todos los enfoques muestran un comportamiento consistente, con poca diferencia entre las estrategias de *Reequilibrio bloqueante* y de *Reequilibrio*

asíncrono (con o sin la implementación de rack local), incluso para la primera iteración. Si bien existe un potencial de aceleración para las iteraciones pares cuando se compara con el límite superior, ya que corresponde al tipo de trabajo *Map*-intensivo de PageRank. Esto no se cumple para las iteraciones impares correspondientes al tipo de trabajo *Reduce*-intensivo, lo que conduce finalmente a un tiempo de finalización global que es peor que el límite superior.

4.4.4 Análisis del proceso de migración de datos

En esta sección se estudia cómo progresa la migración de los datos durante el reequilibrio asíncrono. Esto complementa el análisis por iteración de la sección 4.4.3, ofreciendo más información acerca de la desaceleración observada durante las primeras iteraciones.

En la Figura 4.9 se muestra el progreso de migración para K-Means con 1 Gbps y 100 Mbps de ancho de banda del *enlace débil*, para el fichero de entrada de 20 GB utilizado en estos experimentos. Como se esperaba, la migración más rápida la realiza la estrategia de *Reequilibrio bloqueante* ya que este proceso se realiza de forma aislada sin ningún tipo de interferencia debida a la ejecución de la aplicación. Hay que tener en cuenta que la estrategia de *Reequilibrio asíncrono con rack local* está muy cerca de *Reequilibrio bloqueante*. Esto se explica por que la estrategia de *Reequilibrio asíncrono con rack local* utiliza el *enlace débil* casi únicamente para migrar los datos a las MVs *off-premise*, ya que las tareas *Map* planificadas *off-premise* durante dicho proceso tendrán localidad de datos local y por tanto no necesitarán utilizar el *enlace débil* para leer su bloque. En la estrategia de *Reequilibrio asíncrono* se observa un comportamiento completamente diferente. En este caso, un gran número de tareas *Map* necesitan leer bloques de entrada simultáneamente junto con la migración de los datos a través del *enlace débil*. Esto provoca una saturación del *enlace débil*, que tiene como consecuencia un mayor tiempo de migración de los datos (entre 2 y 3 veces el observado para los otros casos). El comportamiento general para las tres estrategias es muy similar, tanto para los escenarios de 1 Gbps como de 100 Mbps.

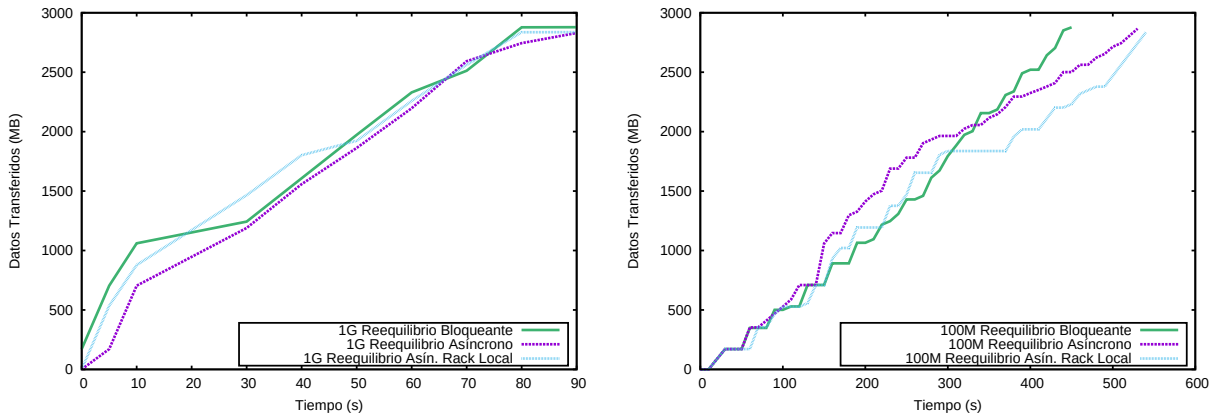


(a) 1 Gbps enlace débil.

(b) 100 Mbps enlace débil.

Figura 4.9: Evolución de la migración para K-Means: cantidad de datos transferidos por tiempo. Menor es mejor.

En el caso de PageRank (Figura 4.10), la similitud entre 1 Gbps y 100 Mbps también es visible; sin embargo, hay una diferencia marcada en comparación con K-Means. La estrategia de *Reequilibrio*



(a) 1 Gbps enlace débil.

(b) 100 Mbps enlace débil

Figura 4.10: Evolución de la migración para Pagerank: cantidad de datos transferidos por tiempo. Menor es mejor.

asíncrono está mucho más cercana a las otras dos estrategias, lo que se puede explicar por el hecho de que el tamaño total de los datos de entrada es mucho menor. Esto se refleja también en el tiempo de migración.

4.5 Metodología de análisis de coste económico

En esta sección se estudia el coste económico de ejecutar aplicaciones MapReduce iterativas sobre nubes híbridas utilizando un despliegue de *cloud bursting*. En este contexto, hemos realizado un estudio comparativo de lo que podría costar lograr las mejoras de rendimiento discutidas en la sección 4.4.2 con cada una de las cuatro estrategias, en un entorno real de nube híbrida.

Con este fin, proponemos una metodología de obtención del coste económico de la ejecución de una aplicación MapReduce sobre un entorno híbrido, de forma que facilite el razonamiento del coste específico para un escenario de *cloud bursting*. En este caso nos centramos en el coste incurrido por el aprovisionamiento de recursos temporales *off-premise* de pago por uso, utilizando un ejemplo real de costes. En este contexto, existen dos fuentes de gastos: (1) el proveedor de la nube, que cobra por la duración de la utilización de la MV (tiempo de actividad) y la cantidad de datos transferidos *on-premise* desde las MVs *off-premise* (tráfico saliente desde la nube pública a la nube privada), y (2) el proveedor de servicios de Internet o ISP, que permite el acceso de red desde/hacia la nube. Normalmente, el coste del ISP se fija en función de la capacidad de ancho de banda y no depende de la cantidad de tráfico. La razón por la que el proveedor de la nube no cobra por cargas de datos que se realizan hacia la nube pública se debe a que tiene como objetivo crear dependencias en la nube para que los usuarios encuentren dificultades para salir de ella (es decir, un "bloqueo"del proveedor). En nuestro estudio, se han evaluado varios proveedores de nube (AWS, Softlayer, y Google, entre otros) y proveedores de servicios de Internet (AT&T, Telefónica, y Verizon, entre otros) para estimar el coste incurrido por una nube pública similar a nuestra configuración (MVs con 4 vCPUs y 16 GB de RAM utilizando una interconexión entre nubes de 100 Mbps y 1 Gbps). En la Tabla 4.4 se resumen los costes promedio típicos que hemos utilizado para nuestro análisis.

Proveedor de nube		Proveedor de ISP	
Utilización de MV	Tráfico de salida	100 Mpbs	1 Gbps
0,239 $/h	0,09 $/GB	0,06 $/h	0,27 $/h

Tabla 4.4: Modelo de coste: deglose típico de gastos

4.5.1 Formulación para el análisis de coste económico

Para llevar a cabo el análisis de coste económico hay que tener en cuenta los costes derivados de la utilización de las máquinas virtuales o instancias (T_c^{vm}), así como del proveedor de ISP (T_c^{isp}), del tráfico de datos de salida (T_c^{out}), y del tráfico de datos de entrada (T_c^{in}). A partir de esta información, presentamos la siguiente fórmula para obtener el coste económico total de una aplicación:

$$T_c = T_c^{vm} + T_c^{isp} + T_c^{out} + T_c^{in}, \tag{4.1}$$

donde,

$$T_c^{vm} = T_{vm} \times C_I \times N_I^{off}$$
$$T_c^{isp} = T_{vm} \times C_{bw}$$
$$T_c^{in} = S_{in} \times C_{in}$$
$$T_c^{out} = S_{out} \times C_{out},$$

siendo T_{vm} el tiempo utilizado por la MV, N_I^{off} el número de MVs empleadas, S_{in} la cantidad de tráfico de entrada y S_{out} la cantidad de tráfico de salida. Además, C_I, C_{in}, C_{out} y C_{bw} corresponden a los costes económicos de la MV o instancia, del tráfico de entrada, del tráfico de salida y del proveedor de ISP, respectivamente.

Por tanto, para realizar un cálculo de los gastos incurridos en las dos aplicaciones estudiadas en este capítulo, es necesario obtener todos los parámetros presentados en la fórmula anterior. La duración de la utilización de las MVs y del ISP viene dada por el tiempo de finalización de la aplicación en concreto, que podemos extraer de la sección 4.4.2. Los costes, tanto de la utilización de las MVs como del ISP, vienen dados por la Tabla 4.4. Sin embargo, la información sobre el tráfico entrante y saliente de las MVs *off-premise*, para cada aplicación y estrategia, ha sido extraída a partir del entorno H-Stat presentado en el capítulo 3. En la siguiente sección se muestra un desglose de todo el tráfico de datos y estadísticas para cada aplicación y estrategia estudiada, de forma que posteriormente se pueda realizar una estimación de los costes de forma detallada.

4.5.2 Tráfico de datos entre nubes

Utilizando nuestra herramienta H-Traffic de H-Stat presentada en la sección 3.2.4, hemos extraído el tráfico de datos para las estrategias y aplicaciones estudiadas, desglosado por iteración. Este tráfico incluye las lecturas y escrituras producidas por HDFS desde/hacia las MVs *on-premise*

y *off-premise*, el tráfico producido por la fase *Shuffle* en ambas partes, y finalmente el tráfico producido por la migración de datos desde las MVs *on-premise* hacia las MVs *off-premise*.

Además, en cada caso se han extraído una serie de estadísticas sobre la planificación de cada una de las tareas MapReduce según la estrategia y aplicación. A partir de estos datos se puede entender mejor el tráfico de datos producido en cada caso, ya que en algunos casos existen tareas MapReduce que finalmente son eliminadas debido a la sobrecarga de la red, consumiendo un tráfico que finalmente no contribuye al procesamiento de la aplicación.

En las Tablas 4.5 y 4.6 se muestra el resumen de todo el tráfico acumulado para las 30 iteraciones de K-Means para cada una de las estrategias, utilizando *enlaces débiles* de 1 Gbps y 100 Mbps respectivamente.

Analizando en ambas tablas el tráfico producido por K-Means, podemos destacar una serie de datos puntuales en cada caso. En la estrategia *Sin HDFS off-premise*, es evidente que en ningún caso se realizan lectura locales de HDFS *off-premise*. Por lo tanto, cualquier tarea *Map* planificada *off-premise* provoca un tráfico desde *on-premise* hacia *off-premise* como se muestra en las tablas ("*Off-premise desde On-premise*"). En este caso, resulta interesante observar la cantidad total de MB leídos desde HDFS, ya que es ligeramente superior al resto de estrategias para ambos enlaces. Esto se puede explicar analizando en las estadísticas la cantidad de tareas *Map* canceladas *off-premise* en ambos enlaces, ya que estas tareas *Map* han sido planificas *off-premise*, han leído parte de su bloque de datos, pero finalmente han sido marcadas como erróneas y planificadas de nuevo. Estas tareas marcadas como erróneas son normalmente la consecuencia de una sobrecarga de la red de interconexión, que impide enviar el estado de la tarea dentro de los tiempos que el planificador admite como válidos. En el caso de 100 Mbps se puede apreciar mejor este comportamiento.

De la misma manera, las Tablas 4.7 y 4.8 muestran el resumen del tráfico acumulado para las 10 iteraciones de PageRank para cada una de las estrategias, utilizando *enlaces débiles* de 1 Gbps y 100 Mbps, respectivamente.

K-Means **1 Gbps de *enlace débil***	**Sin HDFS off-premise**	**Reequilibrio Bloqueante**	**Reequilibrio Asíncrono**	**Reequilibrio Asín. Rack Local**
HDFS LECTURAS (MB)				
Off-premise desde On-premise	339.980	0	20.842	520
On-premise desde Off-premise	0	0	0	0
On-premise Locales	269.171	142.466	150.271	157.346
Off-premise Locales	0	463.434	433.959	449.376
Total Datos Leídos	609.151	605.899	605.072	607.242
HDFS ESCRITURAS (MB)				
On-premise hacia Off-premise	0	0	0	0,005
Off-premise hacia On-premise	0,039	0,078	0,078	0,073
On-premise Locales	0,195	0,078	0,078	0,078
Off-premise Locales	0	0,078	0,078	0,078
Total Datos Escritos	0,234	0,234	0,234	0,234
SHUFFLE (MB)				
Shuffle On-premise a Off-premise	2,95	2,88	3,06	3,07
Shuffle Off-premise a On-premise	3,72	0	0	0,68
Shuffle On-Premise	2,67	0	0	0,17
Shuffle Off-Premise	3,36	9,81	9,64	8,78
Total Shuffle	12,70	12,70	12,70	12,70
ESTADÍSTICAS				
Maps On-premise	2.001	1.025	1.085	1.139
Reduces On-Premise	15	0	0	2
Maps Off-premise	2.567	3.503	3.450	3.381
Reduces Off-premise	15	30	30	28
Maps Cancelados On-premise	6	3	3	10
Reduces Cancelados On-premise	0	0	0	0
Maps Cancelados Off-premise	62	25	32	10
Reduces Cancelados Off-premise	0	0	0	0
Total Maps Satisfactorios	4.500	4.500	4.500	4.500
Total Reduces Satisfactorios	30	30	30	30
REEQUILIBRIO (MB)				
On-premise hacia Off-premise	0	19.152	19.152	19.152

Tabla 4.5: K-Means: transferencias entre nubes en Megabytes (MB), y estadísticas sobre las tareas MapReduce ejecutadas en un *enlace débil* de 1 Gbps.

K-Means **100 Mbps de *enlace débil***	**Sin HDFS off-premise**	**Reequilibrio Bloqueante**	**Reequilibrio Asíncrono**	**Reequilibrio Asín. Rack Local**
HDFS LECTURAS (MB)				
Off-premise desde On-premise	175.648	129	47.012	1.541
On-premise desde Off-premise	0	0	0	0
On-premise Locales	468.090	149.342	202.330	243.395
Off-premise Locales	0	457.308	371.495	366.486
Total Datos Leídos	643.738	606.779	620.837	611.422
HDFS ESCRITURAS (MB)				
On-premise hacia Off-premise	0	0	0,01	0
Off-premise hacia On-premise	0	0,08	0,07	0,08
On-premise Locales	0,23	0,08	0,08	0,08
Off-premise Locales	0	0,08	0,08	0,08
Total Datos Escritos	0,23	0,23	0,23	0,23
SHUFFLE (MB)				
Shuffle On-premise a Off-premise	0	2,94	3,65	4,96
Shuffle Off-premise a On-premise	2,87	0,34	0,45	0
Shuffle On-Premise	9,83	0,09	0,47	0
Shuffle Off-Premise	0	9,34	8,12	7,73
Total Shuffle	12,70	12,70	12,70	12,70
ESTADÍSTICAS				
Maps On-premise	3.479	1.075	1.449	1.757
Reduces On-Premise	30	1	2	0
Maps Off-premise	1.677	3.456	3.290	2.762
Reduces Off-premise	0	29	28	30
Maps Cancelados On-premise	1	4	2	18
Reduces Cancelados On-premise	0	0	0	0
Maps Cancelados Off-premise	655	27	237	1
Reduces Cancelados Off-premise	0	0	0	0
Total Maps Satisfactorios	4.500	4.500	4.500	4.500
Total Reduces Satisfactorios	30	30	30	30
REEQUILIBRIO (MB)				
On-premise hacia Off-premise	0	19.152	19.152	19.152

Tabla 4.6: K-Means: transferencias entre nubes en Megabytes (MB), y estadísticas sobre las tareas MapReduce ejecutadas en un *enlace débil* de 100 Mbps.

PageRank **1 Gbps de *enlace débil***	**Sin HDFS off-premise**	**Reequilibrio Bloqueante**	**Reequilibrio Asíncrono**	**Reequilibrio Asín. Rack Local**
HDFS LECTURAS (MB)				
Off-premise desde On-premise	32.225	0	61	0
On-premise desde Off-premise	0	0	0	0
On-premise Locales	12.868	9.783	10.711	11.054
Off-premise Locales	0	35.153	34.164	33.883
Total Datos Leídos	45.092	44.936	44.937	44.937
HDFS ESCRITURAS (MB)				
On-premise hacia Off-premise	0	7.317	7.046	7.077
Off-premise hacia On-premise	23.321	23.122	23.468	23.309
On-premise Locales	67.886	30.439	30.513	30.386
Off-premise Locales	0	30.439	30.513	30.386
Total Datos Escritos	91.207	91.317	91.540	91.159
SHUFFLE (MB)				
Shuffle On-premise a Off-premise	9.449	6.855	7.592	7.562
Shuffle Off-premise a On-premise	6.649	7.783	6.975	7.589
Shuffle On-Premise	2.805	2.128	2.123	2.322
Shuffle Off-Premise	22.972	25.109	25.185	24.286
Total Shuffle	41.876	41.876	41.876	41.476
ESTADÍSTICAS				
Maps On-premise	326	222	242	251
Reduces On-Premise	112	116	108	115
Maps Off-premise	591	692	674	663
Reduces Off-premise	378	374	382	375
Maps Cancelados On-premise	1	0	2	0
Reduces Cancelados On-premise	1	1	1	1
Maps Cancelados Off-premise	2	0	0	0
Reduces Cancelados Off-premise	9	9	9	9
Total Maps Satisfactorios	914	914	914	914
Total Reduces Satisfactorios	480	480	480	480
REEQUILIBRIO (MB)				
On-premise hacia Off-premise	0	3.076	3.076	3.076

Tabla 4.7: PageRank: transferencias entre nubes en Megabytes (MB), y estadísticas sobre las tareas MapReduce ejecutadas en un *enlace débil* de 1 Gbps.

PageRank **100 Mbps de *enlace débil***	**Sin HDFS off-premise**	**Reequilibrio Bloqueante**	**Reequilibrio Asíncrono**	**Reequilibrio Asín. Rack Local**
HDFS LECTURAS (MB)				
Off-premise desde On-premise	33.124	0	359	0
On-premise desde Off-premise	0	0	0	0
On-premise Locales	12.211	11.837	11.527	11.659
Off-premise Locales	0	33.092	33.043	33.277
Total Datos Leídos	45.335	44.929	44.929	44.937
HDFS ESCRITURAS (MB)				
On-premise hacia Off-premise	0	7.531	7.277	7.261
Off-premise hacia On-premise	14.983	22.779	23.032	23.041
On-premise Locales	75.900	30.309	30.310	30.302
Off-premise Locales	0	30.309	30.310	30.302
Total Datos Escritos	90.883	90.928	90.930	90.906
SHUFFLE (MB)				
Shuffle On-premise a Off-premise	7.079	8.293	8.330	8.529
Shuffle Off-premise a On-premise	9.868	7.729	7.299	7.139
Shuffle On-Premise	3.099	2.599	2.463	2.477
Shuffle Off-Premise	21.777	24.667	25.188	25.074
Total Shuffle	41.824	43.293	43.281	43.220
ESTADÍSTICAS				
Maps On-premise	215	232	244	248
Reduces On-Premise	175	121	114	113
Maps Off-premise	706	684	671	667
Reduces Off-premise	308	376	383	383
Maps Cancelados On-premise	0	0	1	1
Reduces Cancelados On-premise	3	12	11	12
Maps Cancelados Off-premise	7	2	0	0
Reduces Cancelados Off-premise	0	5	6	4
Total Maps Satisfactorios	914	914	914	914
Total Reduces Satisfactorios	480	480	480	480
REEQUILIBRIO (MB)				
On-premise hacia Off-premise	0	3.076	3.076	3.076

Tabla 4.8: PageRank: transferencias entre nubes en Megabytes (MB), y estadísticas sobre las tareas MapReduce ejecutadas en un *enlace débil* de 100 Mbps.

4.5.3 Estimación de costes

Una vez extraídos los datos necesarios para realizar una estimación de costes, se ha realizado un cálculo desglosado de cada uno de los gastos incurridos por la duración de la instancia, el coste del ISP, y el tráfico de entrada y salida, utilizando para ello los costes descritos en la Tabla 4.4 y la fórmula (4.1) propuesta. Hay que tener en cuenta que el proceso de migración realiza el envío de los datos desde las MVs *on-premise* a las MVs *off-premise* y, por lo tanto, no incurre en gasto por los proveedores de la nube.

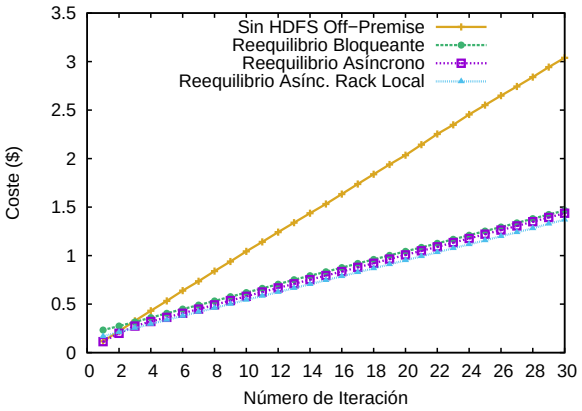
En las Tablas 4.9 y 4.10 se muestra el detalle del coste total de la ejecución completa de K-Means y PageRank para cada estrategia. Además, se ha introducido un método de evaluación que indica qué estrategia es mejor en cada caso desde el punto de vista del tiempo y coste económico. Para ello, se ha evaluado cada estrategia con una puntuación que viene dada por la ecuación *(Tiempo* × *Coste)/1000*, en la cual un valor bajo significa una buena relación tiempo/coste.

Detalle del coste K-Means	Sin HDFS off-premise	Reequilibrio Bloqueante	Reequilibrio Asíncrono	Reequilibrio Asínc. Rack Local
1 Gbps				
Tiempo de ejecución Instancia ($)	2,78 $	1,34 $	1,312855 $	1,25 $
Coste ancho de banda (ISP) ($)	0,26 $	0,13 $	1,12 $	0,12 $
Transf. Off-prem. a On-prem. ($)	0,00033 $	0,000007 $	0,000007 $	0,00007 $
Trans. On-prem. a Off-prem.($)	0 $	0 $	0 $	0 $
Total ($)	**3,04 $**	**1,47 $**	**1,44 $**	**1,37 $**
Puntuación (Tiempo x Coste)	**10,6**	**2,5**	**2,4**	**2,2**
100 Mbps				
Tiempo de ejecución Instancia ($)	13,32 $	3,43 $	5,148246 $	2,16 $
Coste ancho de banda (ISP) ($)	0,28 $	0,07 $	0,11 $	0,05 $
Transf. Off-prem. a On-prem. ($)	0,00025 $	0,000036 $	0,000046 $	0,000007 $
Trans. On-prem. a Off-prem.($)	0 $	0 $	0 $	0 $
Total ($)	**13,6 $**	**3,51 $**	**5,26 $**	**2,21 $**
Puntuación (Tiempo x Coste)	**227,3**	**15,1**	**34**	**6**

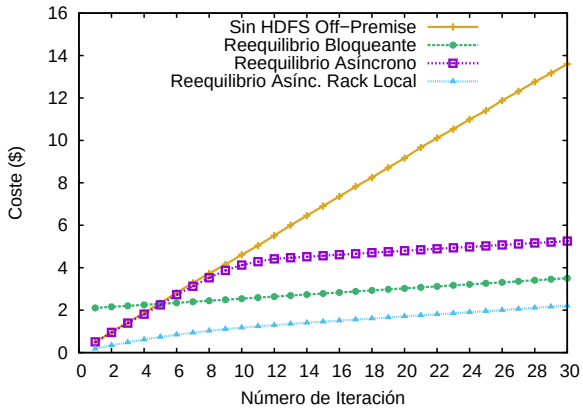
Tabla 4.9: K-Means: desglose de gastos y puntuación obtenida para cada estrategia utilizando *enlaces débiles* de 1 Gbps y 100 Mbps. Menor es mejor.

Toda esta información, combinada con el tiempo de ejecución total por cada iteración, se ha utilizado para calcular la acumulación de costes por iteración (es decir, cómo aumenta el coste total a medida que aumenta el número de iteraciones). En las Figuras 4.11 y 4.12 se muestra el coste acumulado por iteración para K-Means y PageRank, respectivamente.

Como puede observarse, para K-Means la solución más económica y que obtiene mejor puntuación utiliza la estrategia de *Reequilibrio asíncrono con rack local*, tanto para el caso de 1 Gbps como el de 100 Mbps. En el caso de 1 Gbps, todas las estrategias excepto *Sin HDFS off-premise* exhiben un coste económico y puntuación similar. La gran diferencia entre *Sin HDFS off-premise* y el resto es principalmente debida al aumento del tiempo de ejecución de la aplicación. Esto también se mantiene para el caso de 100 Mbps, donde la acumulación de costes también crece de forma lineal pero de forma más rápida. Sin embargo, para el caso de 100 Mbps, surgen grandes diferencias entre



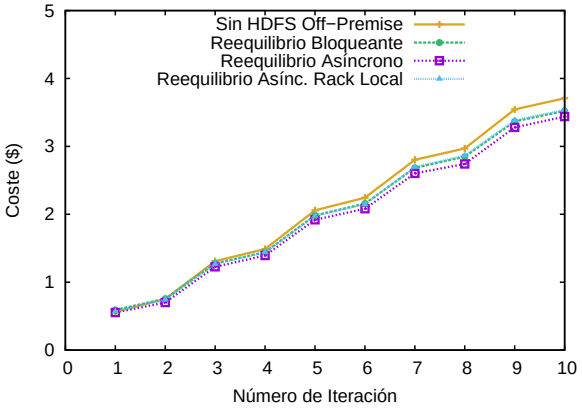
(a) K-Means: 1 Gbps de *enlace débil*.



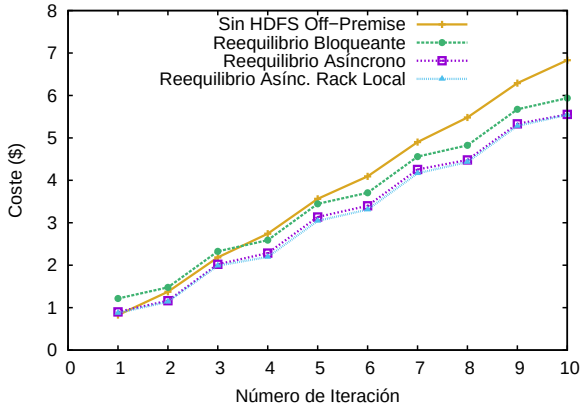
(b) K-Means: 100 Mbps de *enlace débil*.

Figura 4.11: K-Means: Coste económico acumulado por iteración. Menor es mejor.

el resto de estrategias: la de *Reequilibrio bloqueante* es visiblemente más cara que la de *Reequilibrio asíncrono con rack local* debido a la sobrecarga inicial. La estrategia de *Reequilibrio asíncrono* comienza con un coste económico similar a la de *Sin HDFS off-premise*, pero tras completar el proceso de migración mejora su coste económico, sin embargo este coste es superior a las estrategias de *Reequilibrio bloqueante* y de *Reequilibrio asíncrono con rack local*. La estrategia de *Reequilibrio asíncrono con rack local* es inicialmente más económica y mantiene su ventaja independientemente del número de iteraciones. Comparado con el competidor más cercano (*Reequilibrio bloqueante*), es un 37 % más económica y al mismo tiempo un 37 % más rápida. Por tanto, para el caso de K-Means la clara vencedora es la estrategia *Reequilibrio asíncrono con rack local*, además de obtener la mejor puntuación con diferencia.



(a) PageRank: 1 Gbps de *enlace débil*.



(b) PageRank: 100 Mbps de *enlace débil*.

Figura 4.12: PageRank: Coste económico acumulado por iteración. Menor es mejor.

En el caso de PageRank, como se esperaba, la estrategia *Sin HDFS off-premise* genera unos costes mucho más elevados que el resto de estrategias y una relación tiempo/coste mayor que el resto, sobre todo en 100 Mbps. Sin embargo, la diferencia es menor que en el caso de K-Means, debido a que en general se realizan muchas más transferencias a consecuencia de las tareas *Reduce*,

cuyo coste afecta a todas las estrategias. La estrategia de *Reequilibrio asíncrono* sostiene un coste ligeramente menor que la de *Reequilibrio bloqueante* y de *Reequilibrio asíncrono con rack local* para el caso de 1 Gbps. Sin embargo, en el caso de 100 Mbps, tanto la estrategia de *Reequilibrio asíncrono* como la de *Reequilibrio asíncrono con rack local* muestran un coste muy similar, mientras que la estrategia de *Reequilibrio bloqueante* sufre un sobrecoste por el hecho de tener que esperar a que se migren los datos iniciales. Por tanto, para el caso de PageRank, tanto la estrategia de *Reequilibrio asíncrono* como la de *Reequilibrio asíncrono con rack local* serían válidas, obteniendo unos costes muy similares para ambos casos.

Detalle del coste PageRank	Sin HDFS off-premise	Reequilibrio Bloqueante	Reequilibrio Asíncrono	Reequilibrio Asínc. Rack Local
1 Gbps				
Tiempo de ejecución Instancia ($)	0,98 $	0,74 $	0,697382 $	0,75 $
Coste ancho de banda (ISP) ($)	0,09 $	0,07 $	0,065 $	0,07 $
Transf. Off-prem. a On-prem. ($)	2,63 $	2,72 $	2,68 $	2,72 $
Trans. On-prem. a Off-prem.($)	0 $	0 $	0 $	0 $
Total ($)	**3,71 $**	**3,52 $**	**3,44 $**	**3,54 $**
Puntuación (Tiempo x Coste)	**4,6**	**3,26**	**3**	**3,3**
100 Mbps				
Tiempo de ejecución Instancia ($)	4,55 $	3,19 $	2,829079 $	2,83 $
Coste ancho de banda (ISP) ($)	0,10 $	0,07 $	0,06 $	0,06 $
Transf. Off-prem. a On-prem. ($)	2,18 $	2,68 $	2,67 $	2,65 $
Trans. On-prem. a Off-prem.($)	0 $	0 $	0 $	0 $
Total ($)	**6,83 $**	**5,94 $**	**5,55 $**	**5,54 $**
Puntuación (Tiempo x Coste)	**39**	**23,7**	**19,72**	**19,7**

Tabla 4.10: PageRank: desglose de gastos y puntuación obtenida para cada estrategia utilizando *enlaces débiles* de 1 Gbps y 100 Mbps. Menor es mejor.

4.6 Conclusiones

En este capítulo se ha realizado un análisis de la gestión de datos y planificación de tareas que realiza Hadoop MapReduce en un entorno de nube híbrida, mediante un despliegue de *cloud bursting*. Uno de los factores clave en el rendimiento de una aplicación MapReduce en Hadoop es la velocidad de acceso a los datos. Sin embargo, en una nube híbrida existe una red de intercomunicación por la cual debe pasar todo el tráfico entre ambas nubes (*enlace débil*), normalmente con un ancho de banda de un orden de magnitud inferior al de las propias redes de cada nube. Esto provoca que una mala planificación de tareas MapReduce traiga consigo un bajo rendimiento de la aplicación al provocar una congestión del *enlace débil*.

Por tanto, surge la necesidad de explorar distintas estrategias que permitan adaptarse y mejorar la ejecución de una aplicación MapReduce en un despliegue de *cloud bursting*. Para ello, inicialmente se ha realizado un estudio de los distintos aspectos que influyen en la localidad de los datos para cada una de las fases de una aplicación MapReduce. Como resultado, se han planteado varias

estrategias para gestionar la migración inicial de los datos de entrada de las MVs *on-premise* a las MVs *off-premise*.

Mediante el uso de la característica *Rack Awareness* implementada en Hadoop, se ha planteado otro enfoque de su uso para adaptarlo a nubes híbridas, lo cual ha permitido la propuesta de dos estrategias de gestión de localidad de los datos. Sin embargo, tras el análisis de estas estrategias se ha observado que el planificador por defecto de Hadoop no es consciente del uso de un *enlace débil* de bajo rendimiento, lo que provoca una planificación ineficiente de tareas. Por tanto, se ha implementado una nueva política de planificación de Hadoop, que ha dado lugar a una tercera estrategia de gestión de los datos. Además, para comparar estas tres estrategias se ha propuesto una cuarta estrategia en la que no se realiza ninguna acción de migración de los datos a las MVs *off-premise*, lo cual permite tener una perspectiva general a la hora de comparar las distintas estrategias.

Estas cuatro estrategias se han evaluado a través de un entorno de experimentación de nube híbrida controlado, el cual permite limitar el ancho de banda del *enlace débil*, así como el número de MVs *on-premise* y *off-premise*. Se han seleccionado dos aplicaciones MapReduce iterativas reales, K-Means y PageRank, muy populares en el campo del Big Data. Por tanto, mediante un despliegue de 3 MVs *on-premise* y 12 MVs *off-premise*, junto con una configuración del *enlace débil* de 1 Gbps y 100 Mbps, se han evaluado las distintas estrategias propuestas.

Los resultados obtenidos en cada escenario han permitido concluir que la estrategia de *Reequilibrio asíncrono con rack local* (nuestra contribución) es la que mejor resultados de rendimiento obtiene para la aplicación K-Means, incluso es un escenario con un bajo ancho de banda del *enlace débil* de 100 Mbps, ofreciendo una mejora del 53 % del tiempo de finalización con respecto al límite superior propuesto (3 MVs *on-premise*). Estos resultados justifican la mejora introducida en el planificador de Hadoop para las tareas *Map* en este tipo de entornos. Por tanto, puesto que K-Means es una aplicación *Map*-intensiva, puede beneficiarse de la nueva planificación propuesta a través de la estrategia *Reequilibrio asíncrono con rack local*.

PageRank es una aplicación compuesta por dos tipos de trabajos, uno de tipo *Map*-intensivo y otro *Reduce*-intensivo. Por tanto, la nueva política propuesta solo obtiene beneficios en el trabajo de tipo *Map*-intensivo, lo cual se ha podido observar en los resultados obtenidos, siendo la estrategia de *Reequilibrio asíncrono* junto con nuestra estrategia de *Reequilibrio asíncrono con rack local* las que mejor resultados ofrecen. Otra conclusión obtenida de PageRank es que este tipo de aplicación no obtiene ninguna mejora en el tiempo de finalización al utilizar un *enlace débil* de 100 Mbps, con respecto al límite superior. En cambio, utilizando un *enlace débil* de 1 Gbps, obtiene una mejora del tiempo de finalización del 72 % con respecto al límite superior, mediante la estrategia de *Reequilibrio asíncrono*.

Finalmente, se ha contribuido con una metodología de análisis del coste económico que conlleva la ejecución de una aplicación MapReduce iterativa sobre una nube híbrida. Para ello, se ha propuesto una expresión matemática que parametriza distintos factores que suponen un coste económico en la ejecución de una aplicación en este tipo de entornos, como el tiempo de utilización de las MVs *off-premise* y el ISP, así como el tráfico de datos entre nubes. Para ello, se ha hecho uso del entorno H-Stat para la extracción del tráfico producido entre nubes. Esta metodología se ha evaluado a través de los resultados de las dos aplicaciones seleccionadas, y se ha podido concluir que la estrategia de *Reequilibrio asíncrono con rack local* es la que mejor relación tiempo/coste obtiene para aplicaciones de tipo *Map*-intensivas.

CAPÍTULO 5

Metodología de predicción de rendimiento

Tal y como se ha mostrado en los capítulos anteriores de esta tesis, la utilización de un entorno híbrido de computación en la nube para aplicaciones MapReduce iterativas permite mejorar su rendimiento a través de estrategias de localidad de datos adaptadas a este nuevo escenario. Sin embargo, dicho rendimiento no solo depende de una buena estrategia, sino también de otros factores como el ancho de banda de la red entre ambas nubes, el tipo de aplicación iterativa utilizada (*Map*-intensiva/*Reduce*-intensiva), o la cantidad de máquinas virtuales adquiridas puntualmente de la nube pública. Dado que complementar una nube pública con recursos de una nube privada supone un gasto económico, que tal vez no satisfaga las expectativas de rendimiento esperadas debido a los múltiples factores presentados, se plantea la necesidad de estimar cuál será el rendimiento de una aplicación concreta sobre dicho entorno antes de ser ejecutada. De esta forma, se podrá determinar de antemano si resulta conveniente invertir dinero en recursos de cómputo adicionales durante un pico de trabajo, con el fin de mejorar el rendimiento de sus aplicaciones.

En este capítulo se presenta una solución a este problema a través de una metodología de predicción del rendimiento de aplicaciones MapReduce iterativas, ejecutadas sobre una nube híbrida a través de un despliegue de *cloud bursting*. Esta metodología se basa en la realización de una calibración inicial de la arquitectura híbrida sobre la que se desea estimar su rendimiento, y una caracterización de la aplicación deseada. Finalmente, se aplica una expresión matemática que contempla tanto el proceso de migración inicial de datos, como todas las características intrínsecas de una aplicación MapReduce para estimar el tiempo de finalización de una aplicación MapReduce sobre dicha arquitectura de nube híbrida.

Por último, se evalúa la precisión de esta metodología a través de la predicción de cuatro aplicaciones MapReduce iterativas reales sobre distintos escenarios de nube híbrida. Estas aplicaciones han sido extraídas de *benchmarks* MapReduce estandarizados, y permiten cubrir el distinto abanico de posibilidades con respecto a tipos de aplicaciones *Map*-intensivas/*Reduce*-intensivas.

5.1 Metodología

Estimar el comportamiento de cualquier aplicación en una determinada arquitectura conlleva el estudio y control de cómo se comporta dicha aplicación bajo determinadas condiciones. Además, el comportamiento de dicha aplicación debe ser reproducible bajo las mismas condiciones, con un cierto grado de tolerancia.

La ejecución de una aplicación desarrollada con el modelo de programación MapReduce se realiza en paralelo, a través de la ejecución de tareas *Map* y *Reduce* sobre los distintos slots disponibles de cada uno de los nodos participantes. De esta forma, se crean ondas de ejecución, donde cada una de ellas cubre todos los slots disponibles hasta completar el número de tareas designado, de una forma equilibrada. En la Figura 5.1 se muestra un ejemplo de ejecución con 3 ondas de tareas *Map* y 2 ondas de tareas *Reduce*. La ubicación de cada una de estas tareas para la formación de ondas depende tanto de la política de planificación de tareas, como de la ubicación y localidad de los datos. Su rendimiento dependerá de la duración y aprovechamiento de los *slots* de cada onda. Por tanto, para estimar el comportamiento de una aplicación MapReduce sobre un determinado entorno es necesario comprender cómo evolucionan cada una de estas ondas de ejecución a partir de la ubicación y duración de sus tareas.

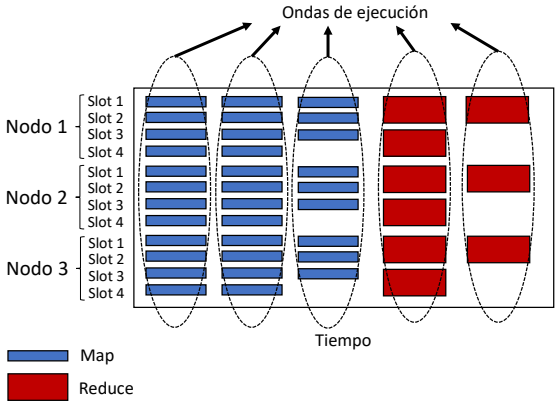


Figura 5.1: Ejemplo de ondas de ejecución de una aplicación MapReduce.

Esta evolución de tareas en la ejecución de una aplicación MapReduce depende de distintos factores que deben ser parametrizados a priori para poder realizar una predicción de dicha evolución. Estos factores se dividen en dos grupos principales: los dependientes de la aplicación y los dependientes de la arquitectura.

En el capítulo anterior se ha realizado un análisis de los factores que afectan al rendimiento en la ejecución de una aplicación iterativa MapReduce en un entorno de *cloud bursting*. Para mejorar el rendimiento en este tipo de entornos se han propuesto distintas estrategias que permiten mejorar la localidad de los datos, teniendo en cuenta tanto la migración inicial de los datos desde las MVs *on-premise* a las MVs *off-premise* como la localidad de dichos datos. De todas ellas, la que mejor se adapta a este tipo de escenarios es la de *Reequilibrio asíncrono con rack local*, ya que permite migrar los datos de entrada de una aplicación a las MVs *off-premise* de forma asíncrona y transparente al usuario. Esto sucede al mismo tiempo que el planificador de Hadoop se adapta a dicha migración, forzando a las tareas *Map* a que se planifiquen únicamente en las MVs donde el dato se encuentre en local o dentro del mismo rack. Tal y como se ha comprobado en el capítulo

anterior, esta nueva política de planificación de datos para Hadoop (propuesta en esta tesis) es consciente de la infraestructura híbrida en la que se está ejecutando la aplicación, permitiendo un rendimiento mucho mejor en aplicaciones *Map*-intensivas. Por tanto, se ha elegido esta estrategia como base para la obtención de un modelo matemático de predicción del tiempo de finalización de aplicaciones MapReduce iterativas sobre un entorno de nube híbrida.

En esta sección presentamos los principios generales y la metodología subyacente de nuestra propuesta de modelado de rendimiento. Este modelo está específicamente dirigido a la obtención del tiempo de finalización de una aplicación MapReduce iterativa sobre una nube híbrida con despliegue de *cloud bursting*, haciendo uso de la estrategia de *Reequilibrio asíncrono con rack local*. Nuestro objetivo es estimar el tiempo de finalización de una aplicación MapReduce iterativa a través de una expresión matemática basada en una serie de parámetros a nivel de sistema y a nivel de aplicación, recopilados de antemano.

Por simplicidad, asumimos que la configuración inicial consiste en un conjunto fijo y homogéneo de MVs *on-premise* y MVs *off-premise*, lo que nos permite obtener un conjunto fijo de parámetros a nivel de sistema. Es posible utilizar la metodología propuesta para estimar el tiempo de finalización de varias configuraciones (por ejemplo, averiguar el número óptimo de MVs *off-premise* para lograr el tiempo de ejecución deseado), aplicándola para cada configuración individualmente, como se mostrará más adelante en la sección de evaluación.

Nuestra metodología consta de tres pasos:

1. En primer lugar, ejecutamos un test sintético para extraer los parámetros a nivel de sistema, correspondientes a una arquitectura de nube híbrida, con un número determinado de MVs *on-premise* y *off-premise*. Estos parámetros son independientes de la aplicación y serán reutilizados en distintas aplicaciones o usuarios (por ejemplo, pueden ser almacenados en las MVs *on-premise* para futuras predicciones y despliegues). Nos referimos a este paso como **calibración del sistema**.

2. En segundo lugar, extraemos los parámetros a nivel de aplicación. Estos parámetros son independientes de la arquitectura de nube híbrida y pueden ser conocidos de antemano u obtenidos ejecutando la aplicación a menor escala en las MVs *on-premise* únicamente. De esta forma, los usuarios pueden estimar los beneficios del uso de una nube híbrida sin ejecutar su aplicación en dicho entorno, siempre y cuando el paso de calibración se haya realizado previamente para la arquitectura deseada. Nos referimos a este paso como **caracterización de aplicaciones**.

3. Finalmente, una vez completada la calibración y la caracterización, aplicamos una expresión matemática para estimar el tiempo de finalización. Obsérvese que existe una variabilidad inherente en las aproximaciones introducidas anteriormente debido a la complejidad de la ejecución del modelo de programación MapReduce, en general, y a la complejidad adicional introducida por el *enlace débil*. Por lo tanto, es importante poder presentar al usuario una predicción del tiempo de finalización optimista (límite inferior) y una pesimista (límite superior). Para derivar las expresiones matemáticas para ambos casos, hacemos uso de una adaptación del teorema de *makespan* aplicado al contexto MapReduce.

5.2 Adapatación del teorema de Makespan a nuestro modelo de predicción

El teorema de Makespan fue propuesto por Verma et al. [92] para la metodología de predicción de ARIA (*Automatic Resource Inference and Allocation*) a partir de ideas ya exploradas en 1966, y presentadas en el trabajo de Graham [46]. Este trabajo trata de obtener la secuenciación óptima para la realización de una serie de tareas por parte de un conjunto de trabajadores, con el fin de minimizar su tiempo de finalización (*makespan*). Esto se puede aplicar también a la planificación de trabajos para su ejecución en un computador, proporcionando una cota o límite inferior y cota o límite superior al tiempo de ejecución.

Los razonamientos aplicados para la solución al problema del *makespan*, pueden utilizarse también al problema de acotar los límites inferior y superior del tiempo de ejecución de una aplicación MapReduce, a partir de los parámetros extraídos en las etapas de calibración y caracterización. Cada una de las fases *Map* y *Reduce* puede considerarse como una serie de n tareas de duración t_i que van a ser ejecutadas en k slots. Asimilando estos datos a la teoría del *makespan*, n se correspondería con el número de tareas a realizar (cada una con una duración determinada t_i) y k con los trabajadores que van a llevarlas a cabo. Los límites se calculan suponiendo una política de planificación basada en un algoritmo voraz (*greedy*) [35].

El teorema del *makespan* introducido en el trabajo por Verma et al. [92] nos indica los límites inferior y superior del tiempo necesario para ejecutar n tareas de duración t_i utilizando k slots, si se aplica una planificación voraz, vienen dados por las expresiones: $n \cdot t_{avg}/k$, para el límite inferior y $(n-1) \cdot t_{avg}/k + t_{max}$, para el límite superior. Donde t_{avg} es el tiempo medio calculado a partir de los tiempos (t_i, $i = 1, \ldots, n$) de las n tareas, y t_{max} es el tiempo de ejecución de la tarea de mayor duración. Estas fórmulas indican una ejecución en ondas de forma implícita, es decir se ejecutarán k tareas en paralelo, después otras k y así sucesivamente hasta que se lleguen a ejecutar las n tareas.

Nosotros, a partir de los datos y conclusiones obtenidas para la definición de la nueva política de planificación desarrollada en el capítulo anterior, hemos refinado las expresiones anteriores con la finalidad de ajustarlas mejor a una ejecución de tareas en ondas, de la siguiente forma:

- El límite inferior lo definimos como

$$\frac{n}{k} \times t_{avg}. \tag{5.1}$$

  En este caso es evidente que la expresión es equivalente a la definida en el teorema de *makespan*. Únicamente modificamos la forma de mostrarla, haciendo hincapié en la ejecución en ondas. En este caso queda patente que lo que se modela es la ejecución de $\frac{n}{k}$ ondas, de duración t_{avg}.

- Por su parte, el límite superior lo definimos como

$$\left(\left\lceil \frac{n}{k} \right\rceil - 1\right) \times t_{avg} + t_{max}. \tag{5.2}$$

  Aunque nos encontramos ante una ejecución en ondas, ésta no se realiza en pasos perfectamente definidos. Cuando se utiliza un algoritmo de planificación voraz, en cuanto un recurso finaliza la ejecución de una tarea se le asigna inmediatamente otra, sin ningún tipo de sincronización entre ellas, más allá de las que dicte la dependencia de datos. Esto hace, como hemos comprobado en el capítulo anterior, que aparezca un cierto solapamiento en la ejecución de

estas ondas. Un límite superior del tiempo de ejecución del conjunto de tareas se dará ante una planificación, en la que la onda en la que se ejecuta la tarea con mayor tiempo de ejecución (t_{max}) se ejecute sin solapamiento con ninguna otra onda. Esto es precisamente lo que modela la expresión que acabamos de introducir como límite superior. En ella, del total de ondas necesarias para la ejecución de n, se considera que aquella que va a ejecutar la tarea de máxima duración se ejecuta sin solapamiento con ninguna otra. Además, dado que lo que se modela es un límite superior, el número de ondas se expresa a través de una función techo, indicando un caso más pesimista en el que la planificación de tareas sufra ciertos retrasos. La fórmula expuesta modela dicho comportamiento, considerando la ejecución del total de ondas menos 1, $\left(\left\lceil \frac{n}{k} \right\rceil - 1\right)$, con una duración media de t_{avg} y añadiendo a dicho tiempo el tiempo necesario para ejecutar la onda que contiene la tarea de máxima duración (t_{max}).

En las secciones siguientes vamos a utilizar las expresiones (5.1) y (5.2) para definir un nuevo modelo de predicción del tiempo de ejecución de aplicaciones MapReduce ejecutándose en un entorno de *cloud bursting*. Cabe destacar que, como se ha mostrado en las expresiones anteriores, para obtener un límite inferior y superior se debe obtener el tiempo medio y máximo de las tareas, respectivamente. Por tanto, los distintos parámetros que extraemos durante las etapas de calibración y caracterización corresponden con los valores medios y máximos.

5.3 Calibración del sistema

Tal y como se ha comentado, la primera parte de la metodología de predicción consiste en la obtención de una calibración del sistema. A partir de esta calibración se obtienen los parámetros que definen el comportamiento del sistema, que son independientes de la aplicación que se ejecute. Se debe tener en cuenta que nos encontramos ante una arquitectura de nube híbrida de MVs homogéneas, que está formada por una nube privada interconectada con una nube pública a través de un único canal de comunicación. Este *enlace débil* entre ambas nubes es de un rendimiento menor al que tienen sus redes internas (al menos un orden de magnitud), lo que produce un claro impacto en las comunicaciones que se originan (tanto para realizar operaciones de acceso al sistema de almacenamiento, como las debidas a transferencias de datos entre tareas) durante la ejecución de una aplicación MapReduce. Por tanto, se deben calibrar los parámetros que afectan a una ejecución MapReduce para poder modelar su comportamiento en futuras ejecuciones.

Las fases de MapReduce que más se ven afectadas por este tipo de comunicación heterogénea entre nubes son:

- Fase *Shuffle*, ya que se encarga de trasladar todos los datos intermedios de las tareas *Map* a las tareas *Reduce*. Se produce un esquema de comunicación desde todas las tareas *Map* hacia todas las tareas *Reduce*. Esto implica un intercambio de datos entre prácticamente todas las MVs participantes de la ejecución de una aplicación MapReduce. En concreto, tanto las tareas *Map* como *Reduce* son homogéneamente distribuidas entre todas las MVs, tanto *on-premise* como *off-premise*. Por tanto, prácticamente todo el tráfico de datos intermedio producido por la fase *Shuffle* se ve afectado por el *enlace débil*.
- Fase *Reduce*, donde cada tarea *Reduce* debe escribir su resultado en el sistema de almacenamiento distribuido HDFS. Como se ha descrito en la sección 4.2 del capítulo anterior, HDFS se ha configurado para utilizar la característica de *Rack Awareness* para garantizar la migración de, al menos, una réplica de los datos a las nuevas MVs *off-premise* recién adquiridas en

una nube híbrida. Esta característica obliga al sistema de almacenamiento HDFS a escribir siempre al menos una réplica de los datos fuera del rack donde se ha realizado la escritura; en nuestro caso, un rack corresponde al grupo de MVs *on-premise* y el otro a las MVs *off-premise*. Por tanto, todas las escrituras al sistema HDFS producidas por las tareas de *Reduce* generan un tráfico de datos a través del *enlace débil* que incrementa el tiempo de escritura (dependiendo del ancho de banda del *enlace débil*).

Cabe destacar que la fase *Map* no se ve prácticamente afectada en este caso, ya que la política utilizada en la estrategia de *Reequilibrio asíncrono con rack local* no permite la ejecución de una tarea *Map* que no disponga de su entrada de datos en local. Sin embargo como se verá más adelante, existen pequeños fragmentos de entrada de datos que deben ser leídos de otros nodos remotos, lo que produce un pequeño incremento del tiempo de lectura del sistema de almacenamiento HDFS.

5.3.1 Test Sintético de calibración

Para realizar la calibración del sistema hemos diseñado y desarrollado un test sintético (inspirado en *WordCount*[1]) enfocado a la extracción de los parámetros de rendimiento de E/S y de sobrecarga de comunicación provocada por una configuración de nube híbrida. Concretamente, el objetivo de este test es obtener los parámetros que se ven más afectados en las fases de MapReduce (*Map*, *Shuffle* y *Reduce*), en función de los siguientes datos:

- La cantidad de datos leídos desde el sistema de almacenamiento HDFS.
- La cantidad de datos escritos al sistema de almacenamiento HDFS.
- La cantidad de datos intermedios (*Shuffle*) entre las tareas *Map* y *Reduce*.

Para lograr este objetivo, hemos desarrollado un código MapReduce con una función de *Map* y otra de *Reduce*, detallados en la Función 1 y la Función 2, respectivamente. La función *Map* se estructura en dos partes: inicialmente lee un fragmento de datos de entrada, y posteriormente escribe una cantidad específica de datos intermedios (en un formato clave/valor) en la función de combinación (*Combiner*). La función de combinación forma parte de una tarea *Map* y permite realizar un pequeño procesamiento de los datos parecido al de la fase *Reduce*, con el objetivo de reducir la cantidad de datos de intercambio y, de esta forma, disminuir la sobrecarga de comunicación. La salida del combinador se agrupa por clave, donde cada grupo de claves es recogido por su correspondiente tarea *Reduce*, controlando de esta forma la cantidad de datos intermedios de *Shuffle* que debe leer cada tarea *Reduce*, al ser especificada como parámetro de entrada. Las tareas *Reduce* simplemente escriben una cantidad específica de datos en el sistema de almacenamiento HDFS. Hay que destacar que tanto la función *Map* como la función *Reduce* evitan realizar ningún tipo de cálculo (salvo una mínima carga en CPU) para aislar la sobrecarga producida por la comunicación de E/S.

Hay que hacer especial hincapié en las funciones de combinación y de *Reduce*. En este caso existen dos parámetros a controlar: por una parte los datos que provienen de la fase *Shuffle*, y por otra los datos que se generan como resultado de la función *Reduce*. Por lo tanto, fijando el tamaño de datos de salida de las tareas *Reduce* a un determinado valor, y variando la cantidad de datos de *Shuffle* que recibe cada tarea *Reduce*, podemos obtener información acerca del tiempo que requiere

[1] `https://wiki.apache.org/hadoop/WordCount`

Función 1 Funciones de *Map* y combinación para el test sintético de calibración del sistema con: M tareas *Map*, R tareas *Reduce* y tamaño de *Shuffle* sh.

```
1: function MAP(n)                                          ▷ n palabras por map
2:     count = 0
3:     while hasMoreData and count < n do
4:         Read input data
5:         emit(word,1)
6:         count = count + 1
7:     end while
8: end function
9: function COMBINER(sh, out_map, M, R)                     ▷ out_map: tamaño del map
10:    size = (out_map × M)/R
11:    sum = sum of values for the same key
12:    iter = sh / size
13:    for i = 1 to iter do
14:        emit(word,sum)
15:    end for
16: end function
```

Función 2 Función de *Reduce* para el test sintético de calibración del sistema con: tamaño de salida rs, tamaño de *Shuffle* sh, y R tareas *Reduce*.

```
1: function REDUCE(rs, sh)
2:     size = out_comb/R                                    ▷ out_comb: tamaño del combiner
3:     sum = sum of values for the same key
4:     iter = rs/size
5:     for i = 1 to iter do
6:         emit(word,sum)
7:     end for
8: end function
```

cada tarea *Reduce* en leer una cantidad variable de datos de *Shuffle*. De esta forma, variando dicha cantidad de *Shuffle*, podemos obtener varias nubes de puntos de *Shuffle* por cada tarea *Reduce*, que representen el tiempo de lectura por cantidad de datos, y realizar una regresión lineal que aproxime el tiempo que necesita cualquier cantidad de datos de *Shuffle* por *Reduce* para dicha arquitectura.

De la misma manera, fijar una cantidad determinada de *Shuffle* y variar la cantidad de datos de salida de las tareas *Reduce*, permite extraer el tiempo que necesita cada tarea *Reduce* en escribir en el sistema de almacenamiento HDFS utilizando la misma metodología.

5.3.2 Parámetros extraídos de la calibración

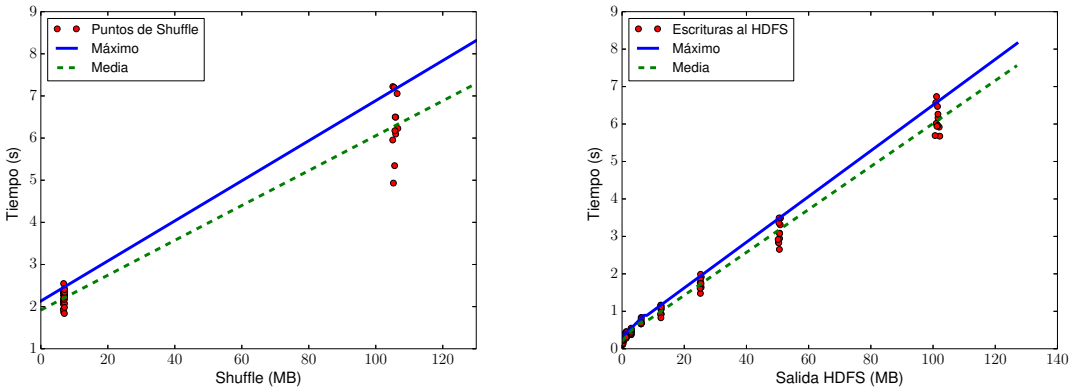
A partir de la ejecución del test sintético en una configuración de nube híbrida comprendida por un número determinado de MVs *on-premise* y *off-premise*, para una cantidad variable de datos, y junto con la utilización de nuestra herramienta H-Stat para la extracción de datos, se obtienen los siguientes parámetros de calibración del sistema:

- **Factor de sobrecarga de la fase *Map***: aunque el planificador fuerza la ejecución de las tareas *Map* sobre los nodos donde están presentes los datos de entrada de HDFS, se producen lecturas adicionales de datos cuando el final lógico de los datos no coincide exactamente con el final del bloque inicialmente leído desde HDFS [98]. Por tanto, en ciertos casos, algunas tareas *Map* deben leer bloques de datos remotos a través del *enlace débil*. Esto genera una sobrecarga en las lecturas de HDFS comparada con el caso de las lecturas producidas únicamente por las MVs *on-premise*. Esta sobrecarga la medimos a través del test sintético y la expresamos como un coeficiente de sobrecarga denotado como α.

- **Función de aproximación de la fase *Shuffle***: una vez que un número suficiente de tareas *Map* ha terminado de producir los datos intermedios, se inicia la ejecución de las tareas *Reduce*, y éstas empiezan a recoger dichos datos intermedios. Sin embargo, una tarea *Reduce* no puede iniciar la fase *Reduce* al mismo tiempo que se recopilan los datos intermedios, ya que necesita ordenarlos primero (lo que no puede ocurrir antes de que todas las tareas *Map* hayan terminado y sus datos intermedios hayan sido recopilados). Por lo tanto, cada tarea *Reduce* experimenta una sobrecarga de la fase *Shuffle* que es proporcional a la cantidad de datos intermedios que debe procesar. Puesto que la cantidad de datos intermedios por *Reduce* (denotada como d_{Sh}) es dependiente de la aplicación, expresamos la sobrecarga de la fase *Shuffle* como una función de aproximación (denotada como f_{Sh}). Para ello, se elige un conjunto de tamaños de datos representativos de *Shuffle*, y se miden los tiempos obtenidos de la fase *Shuffle* para cada tarea *Reduce* utilizando el test sintético de calibración. Finalmente, aplicando una regresión lineal a dichos datos, se obtienen dos funciones de aproximación: una para la media (usando todos los puntos) y otra para el máximo, utilizando sólo el tiempo de *Reduce* más lento para cada tamaño de *Shuffle*.

- **Función de aproximación de escritura de la fase *Reduce***: en el caso de la fase de *Reduce*, la carga computacional de la ejecución de cada una de las tareas *Reduce* representa una parte significativa del tiempo de ejecución de dicha fase, pero no depende del *enlace débil*. Sin embargo, una vez que cada tarea *Reduce* ha terminado el cálculo, necesita escribir sus datos de salida en la capa de almacenamiento HDFS, lo que implica un tráfico a través del *enlace débil* y un sobrecoste en el tiempo de escritura. De nuevo, la cantidad de datos de salida por cada *Reduce* (denotada como d_{Rd}) depende de la aplicación. Por lo tanto, al igual que en la función de aproximación de la fase *Shuffle*, se elige un conjunto de tamaños representativos de salida de *Reduce*, se miden los tiempos de escritura obtenidos para cada tarea *Reduce*, y se aplica una regresión lineal con la que se obtienen dos funciones de aproximación para la fase *Reduce* (denotada como f_{Rd}): una para la media (usando todos los puntos) y otra para el máximo, utilizando sólo el tiempo de escritura más lento para cada tamaño de salida de *Reduce*.

En la Figura 5.2 se muestra un ejemplo de calibración del sistema para la fase *Shuffle* y para la escritura en el sistema de almacenamiento HDFS (salida del *Reduce*), para una arquitectura de nube híbrida con 3 MVs *on-premise* y 3 MVs *off-premise* con un *enlace débil* de 1 Gbps. En este ejemplo existen 24 slots para las tareas *Map* y 12 para las tareas *Reduce*. Como se muestra en la Figura 5.2, y teniendo en cuenta la adaptación del teorema de *makespan* de la sección 5.2, se ha realizado una regresión lineal para los puntos que están por encima de la media más la desviación típica, representando el límite superior o peor caso, y otra regresión lineal para la media de los puntos, representando el límite inferior o mejor caso. Este procedimiento se ha realizado tanto para los puntos de tiempos de *Shuffle* como los de escritura al HDFS o salida del *Reduce*.



(a) Shuffle (b) Reduce

Figura 5.2: Ejemplo de calibración del sistema de las fases *Shuffle* y *Reduce* para una arquitectura de nube híbrida con 3 MVs *on-premise* y 3 MVs *off-premise* con un *enlace débil* de 1 Gbps.

Además de los parámetros obtenidos mediante el test sintético, es necesario extraer los siguientes parámetros de calibración del sistema:

- **Ancho de banda de migración de los datos**: además de las sobrecargas relacionadas estrictamente con el tiempo de ejecución de la propia aplicación MapReduce, también es importante estimar cuánto tiempo durará la migración inicial de los datos de entrada desde la parte inicial *on-premise* del sistema HDFS hasta la parte extendida *off-premise* de dicho sistema de almacenamiento, la cual se realiza a través del *enlace débil*. Esto es importante porque dicha migración se ejecuta de forma asíncrona mientras que la aplicación MapReduce iterativa va progresando, creando una interferencia y una degradación del rendimiento de la aplicación. Para ello, creamos un fichero de entrada de gran tamaño (por ejemplo, 10 GB) en la MVs *on-premise* y desplegamos el sistema HDFS en las MVs *off-premise*, de forma que obligue al sistema a migrar una réplica de los datos en las MVs *off-premise* y podamos medir el tiempo de migración. A partir de los datos recogidos de esta migración calculamos el ancho de banda promedio denotado por β, que se utilizará en nuestro modelo de predicción para tener en cuenta el efecto de la migración de los datos.

- **Slots disponibles *on-premise* y *off-premise***: dada una configuración de nube híbrida, extraemos la cantidad de slots de *Map* configurados, tanto para las MVs *on-premise* como *off-premise* (denotados como S^M_{on} y S^M_{off}, respectivamente), así como el número total de slots de *Reduce* configurados (denotados como S^R).

Resumimos todos los parámetros a nivel de sistema extraídos usando el paso de calibración en la Tabla 5.1.

Parámetros del sistema	
Parámetro	**Descripción**
S_{on}^{M}	Slots de *Map* configurados *on-premise*.
S_{off}^{M}	Slots de *Map* configurados *off-premise*.
S^{R}	Slots totales de *Reduce*.
α	Factor de sobrecarga de la fase *Map*.
β	Ancho de banda de migración de los datos.
$f_{Sh}(d_{Sh}), f_{Sh}^{max}(d_{Sh})$	Función de aproximación de *Shuffle* (media y máximo).
$f_{Rd}(d_{Rd}), f_{Rd}^{max}(d_{Rd})$	Función de aproximación de escritura de *Reduce* (media y máximo).

Tabla 5.1: Parámetros a nivel de sistema extraídos del proceso de calibración.

5.4 Caracterización de aplicaciones

En esta sección se presenta el proceso que permite obtener la información necesaria para caracterizar una aplicación MapReduce iterativa. Esta caracterización se realiza a partir de la extracción de la información relevante que permanece invariante ante un cambio de escala de la arquitectura, dada un arquitectura homogénea. De esta forma, los datos obtenidos de la caracterización servirán como parámetros para realizar una predicción de una escala distinta de dicha arquitectura. Cabe destacar que este proceso sólo es necesario cuando los parámetros que se muestran a continuación no se conocen con antelación, o no pueden ser directamente extraídos del conocimiento existente de la aplicación.

El proceso de caracterización de una aplicación MapReduce iterativa consiste en la ejecución de ésta en pequeña escala, tanto en número de MVs como en iteraciones, sobre una arquitectura exclusivamente *on-premise*. De esta forma, a partir de los resultados obtenidos, se extrae la información relevante a través de la utilización de nuestra herramienta H-Profile de H-Stat.

Concretamente, la información invariante que se extrae para caracterizar una aplicación es la siguiente:

- El número total de tareas *Map* que se ejecutan. Este dato depende del volumen de datos de entrada.
- El número total de tareas *Reduce*. Este parámetro viene configurado por el usuario.
- El tiempo medio y máximo de finalización de una tarea *Map*. La entrada de datos de una aplicación MapReduce se divide en trozos de tamaño equivalente al tamaño de bloque configurado en HDFS (normalmente 128 MB). De esta forma, cada tarea *Map* lee y procesa la misma cantidad de datos y permite un equilibrio y paralelización de las tareas de forma más óptima. Para la ejecución de cada tarea *Map* se asigna un *slot* disponible de entre los definidos en todos los nodos participantes. Este comportamiento es independiente del número de nodos configurados: podemos ampliar o disminuir la cantidad de slots, añadiendo o quitando nodos. Esta forma de ejecución genera una serie de ondas de ejecución, tal y como se ha mostrado en la Figura 5.1. Para una misma entrada de datos y aplicación podemos estimar el número de ondas que se realizarán dependiendo del número de *slots* disponibles, y por lo tanto, el tiempo de ejecución de toda la fase *Map*. En este caso, el tiempo de ejecución de cada tarea *Map* se comporta como una invariante en una arquitectura de recursos de cómputo homogéneos.

- El tiempo medio y máximo que necesita una tarea *Reduce* en procesar su función. Este proceso incluye el tiempo de ordenación o fase *Sort* y el de cómputo de la función *Reduce* implementada, pero no incluye el tiempo de escritura de los resultados en el sistema de almacenamiento HDFS.
- El tamaño de entrada de datos que debe leer cada tarea *Reduce*, es decir, la cantidad de datos gestionados en la fase *Shuffle* por cada tarea *Reduce*.
- El tamaño de salida que debe escribir cada tarea *Reduce* en el sistema de almacenamiento HDFS.

Todos los parámetros extraídos del proceso de caracterización se muestran en la Tabla 5.2.

Parámetros de caracterización de aplicación	
Parámetro	**Descripción**
M	Número total de tareas *Map*.
R	Número total de tareas *Reduce*.
t_{Mp}, t_{Mp}^{max}	Tiempo de finalización medio y máximo de tareas *Map*.
t_{Rc}, t_{Rc}^{max}	Tiempo de finalización medio y máximo de tareas *Reduce* (sort y cómputo).
d_{Sh}	Tamaño de datos de *Shuffle* por *Reduce*.
d_{Rd}	Tamaño de datos de escritura por *Reduce*.

Tabla 5.2: Parámetros de caracterización de una aplicación MapReduce.

Cabe destacar que la planificación de cada una de las tareas *Map* o *Reduce* requiere un tiempo de planificación que no es despreciable con respecto al tiempo de ejecución total de la aplicación. Este tiempo de planificación puede ser extraído también a través de la herramienta H-Profile. En este caso, por tanto, en el tiempo medio y máximo de finalización de una tarea *Map* o *Reduce* (t_{Mp}, t_{Mp}^{max}, t_{Rc} y t_{Rc}^{max}, respectivamente) que se muestra en el resto del capítulo, se incorpora su tiempo de planificación medio o máximo según el caso.

5.5 Modelo de predicción de rendimiento

En esta sección introducimos el modelo de rendimiento que permite a los usuarios estimar el tiempo de finalización de aplicaciones iterativas en escenarios de *cloud bursting*. El modelo de rendimiento es una expresión matemática que relaciona los parámetros de calibración del sistema con los parámetros a nivel de aplicación presentados en los apartados anteriores.

La media del tiempo de finalización T_{total} de una aplicación MapReduce iterativa de n iteraciones, se expresa como:

$$T_{total} = \sum_{i=1}^{n} T(i), \tag{5.3}$$

donde $T(i)$ es la media del tiempo de finalización de la iteración i-ésima, y se puede descomponer como:

$$T(i) = T_{Mp}(i) + T_{Sh}(i) + T_{Rd}(i), \tag{5.4}$$

donde T_{Mp} es la media del tiempo de finalización de la fase *Map*, T_{Sh} el de la fase *Shuffle* y T_{Rd} el de la fase *Reduce*. Esta expresión se corresponde con el mejor de los casos (límite inferior).

Del mismo modo, si en lugar de la media se considera el caso más desfavorable, se tiene para el peor de los casos (límite superior):

$$T_{total}^{max} = \sum_{i=1}^{n} T^{max}(i), \tag{5.5}$$

en donde, de forma similar a la ecuación 5.3, $T^{max}(i)$ es el tiempo máximo de ejecución de la iteración i-ésima, y se puede obtener como:

$$T^{max}(i) = T_{Mp}^{max}(i) + T_{Sh}^{max}(i) + T_{Rd}^{max}(i), \tag{5.6}$$

donde T_{Mp}^{max} es el tiempo máximo de finalización de la fase *Map*, T_{Sh}^{max} es el tiempo máximo de finalización de la fase *Shuffle* y T_{Rd}^{max} es el correspondiente tiempo máximo de finalización de la fase *Reduce*.

Para el resto de esta sección, nos centraremos en los tiempos medios suponiendo que las diferentes iteraciones de la aplicación se comportan de forma similar, es decir $T_{Mp}(i) = T_{Mp}$, $T_{Sh}(i) = T_{Sh}$ y $T_{Rd}(i) = T_{Rd}$, $\forall i$. El desarrollo que se describe es completamente aplicable a los tiempos máximos. Posteriormente indicaremos (y se mostrará mediante la experimentación) cómo se puede aplicar esta misma metodología para el caso de aplicaciones con iteraciones complejas que presenten tiempos de finalización diferentes.

5.5.1 Tiempo de finalización de la fase Map

Con el fin de obtener una expresión matemática para estimar el tiempo de finalización de la fase *Map*, es importante comprender cómo evoluciona esta fase durante las sucesivas iteraciones de un trabajo MapReduce. Utilizando la política de planificación asociada a la localidad de datos, *Reequilibrio asíncrono con rack local*, forzamos a que cualquier tarea *Map* se planifique en una MV donde haya una copia de sus datos de entrada. Por tanto, en la primera iteración sabemos que todas las tareas *Map* se planificarán *on-premise*, dado que las MVs *off-premise* incorporadas al inicio de la ejecución no contienen todavía ningún dato. De forma paralela al inicio de la ejecución, empieza el proceso de migración de los datos de entrada a la infraestructura *off-premise* en segundo plano. Esto significa que, al principio de la segunda iteración, algunos bloques de datos de entrada ya se habrán migrado a las MVs *off-premise*, y el planificador podrá planificar algunas tareas *Map* en la infraestructura *off-premise*.

A medida que progresa la migración de una de las réplicas de datos de entrada, en cada iteración aumentará el número de tareas *Map* que se ejecutarán *off-premise* hasta estabilizarse. Esto significa que a partir de ese momento, en las iteraciones posteriores el número de tareas *Map* ejecutadas *on-premise* y *off-premise* no sufrirá variaciones, manteniéndose constante. Esta situación no tiene por qué coincidir con la finalización de la migración, ya que puede suceder que los slots de *Map*

(contando tanto los definidos en las MVs *on-premise* como en las MVs *off-premise*) se hayan saturado incluso antes de que todos los bloques que componen una réplica hayan sido migrados a MVs *off-premise*.

Este comportamiento debemos expresarlo de forma matemática para que forme parte de la predicción del tiempo de finalización. Cabe destacar, que el tiempo de finalización medio y máximo de una tarea *Map* (T_{Mp} y T_{Mp}^{max}) incluye el factor de sobrecarga α de la fase *Map*, extraído del proceso de calibración de la arquitectura sobre la que se ejecute.

Para la primera iteración, todas las tareas *Map* (M) se ejecutarán *on-premise*. Por tanto, considerando que hay S_{on}^{M} slots de *Map* en los nodos *on-premise*, el tiempo de finalización de la fase *Map* para la primera iteración es:

$$T_{Mp}(1) = \frac{M}{S_{on}^{M}} \times t_{Mp}. \tag{5.7}$$

Para la segunda iteración, habrá una serie de bloques ya transferidos a las MVs *off-premise*, que podrán ser utilizados para ejecutar un determinado número de tareas *Map* (M_1^{off}) en dichas MVs. Este número se puede aproximar utilizando el ancho de banda de migración de datos (β), el tamaño de un bloque en HDFS (s) y el tiempo de ejecución transcurrido hasta ese instante, que en este caso corresponde con el tiempo de finalización de la primera iteración $T(1)$, como se indica en la siguiente ecuación:

$$M_1^{off} = \frac{\beta \times T(1)}{s}. \tag{5.8}$$

Si M_1^{off} tareas *Map* se planifican *off-premise*, el resto de tareas *Map*, $M - M_1^{off}$ (que suponemos que es mayor que M_1^{off}), se planificarán para su ejecución en MVs *on-premise*. Por consiguiente, el tiempo de finalización de la fase *Map* para la segunda iteración es:

$$T_{Mp}(2) = \frac{M - M_1^{off}}{S_{on}^{M}} \times t_{Mp}. \tag{5.9}$$

Utilizando el razonamiento previo para la tercera iteración, el tiempo de finalización para la fase *Map* es:

$$T_{Mp}(3) = \frac{M - ((\beta \times (T(1) + T(2)))/s)}{S_{on}^{M}} \times t_{Mp}. \tag{5.10}$$

En este caso el tiempo transcurrido corresponde con la suma de primera y segunda iteración ($T(1)+T(2)$). Por generalización, para la i-ésima iteración podemos obtener el siguiente tiempo de finalización para la fase *Map*:

$$T_{Mp}(i) = \frac{M - ((\beta \times \sum_{j=1}^{i-1} T(j))/s)}{S_{on}^{M}} \times t_{Mp}. \tag{5.11}$$

Esta fórmula es cierta siempre y cuando todos los slots de *Map* de las MVs *on-premise* estén llenos y existan slots de *Map* en *off-premise* que estén ociosos y no puedan ser usados debido a que el proceso de migración aún no haya enviado suficientes bloques de datos a las MVs *off-premise*. El momento en que ocurre la estabilización puede expresarse matemáticamente de la siguiente manera:

$$M^{off} \geq \frac{S^M_{off}}{S^M_{on} + S^M_{off}} \times M. \tag{5.12}$$

Desde ese momento en adelante, el tiempo para procesar tareas *Map* planificadas *on-premise* será prácticamente el mismo que el empleado en procesar las tareas *Map* planificadas *off-premise*. Esto se debe a que el planificador intenta equilibrar el sistema asignando tareas *Map* en paralelo a todos los nodos disponibles, de forma que finalmente contribuyan de la misma forma al cálculo y por tanto se ejecute el mismo número de ondas. En esta situación, el número de tareas *Map* planificadas *off-premise* será M^{off}, y el número de tareas *Map* planificadas *on-premise* será $M - M^{off}$. Estos valores se mantendrán constantes durante el resto de iteraciones, llegando a la siguiente expresión matemática para el tiempo de finalización restante:

$$T_{Mp}(i) = \frac{M - M^{off}}{S^M_{on}} \times t_{Mp}, \tag{5.13}$$

donde

$$M^{off} = \frac{S^M_{off}}{S^M_{on} + S^M_{off}} \times M. \tag{5.14}$$

Es posible unir las expresiones de las ecuaciones (5.11) y (5.13) en una única expresión que prediga el tiempo de finalización de la fase *Map* para cualquier iteración:

$$T_{Mp}(i) = \frac{M - M^{off}_{i-1}}{S^M_{on}} \times t_{Mp}, \tag{5.15}$$

donde

$$M^{off}_{i-1} = min\left(\frac{\beta \times \sum_{j=1}^{i-1} T(j)}{s}, \frac{S^M_{off}}{S^M_{on} + S^M_{off}} \times M\right). \tag{5.16}$$

La expresión (5.15) ofrece una estimación del tiempo de finalización de la fase *Map* para el caso optimista (límite inferior). Para establecer un límite superior del tiempo de finalización (versión pesimista), aplicamos la adaptación del teorema de *makespan* presentado en la sección 5.2, a partir del cual obtenemos la siguiente expresión:

$$T^{max}_{Mp}(i) = \left(\left\lceil \frac{M - M^{off}_{i-1}}{S^M_{on}} \right\rceil - 1\right) \times t_{Mp} + t^{max}_{Mp}. \tag{5.17}$$

5.5.2 Tiempo de finalización de la fase Shuffle

Como se ha indicado con anterioridad, antes de poder ejecutar una tarea *Reduce*, los datos intermedios procesados por las tareas *Map* deben ser enviados a las diferentes tareas *Reduce*. Esto

provoca un patrón de comunicaciones desde todas las tareas *Map* hacia todas las tareas *Reduce*. Así pues, cada una de estas tareas *Reduce* posee una fase inicial de *Shuffle* en la que recibe una parte del total de datos intermedios a procesar. Para estimar el tiempo medio de la fase *Shuffle* simplemente aplicamos la función f_{Sh}, obtenida de la calibración del sistema, teniendo como valor de la incógnita el tamaño en Megabytes de datos, d_{Sh}, gestionados durante la fase inicial de *Shuffle* por cada tarea *Reduce*. Esta última información se obtiene de la caracterización de la aplicación. Por tanto, la expresión que predice el tiempo de finalización de la fase *Shuffle* para el mejor caso es (límite inferior):

$$T_{Sh}(i) = \frac{R}{S^R} \times f_{Sh}(d_{Sh}), \quad \forall i = 1..n, \tag{5.18}$$

donde R es el número total de tareas *Reduce* que compiten por S^R slots destinados a tareas *Reduce*.

De la misma manera, aplicando la adaptación del teorema de *makespan*, obtenemos la aproximación para el peor caso (límite superior):

$$T_{Sh}^{max}(i) = \left(\left\lceil \frac{R}{S^R} \right\rceil - 1 \right) \times f_{Sh}(d_{Sh}) + f_{Sh}^{max}(d_{Sh}), \quad \forall i = 1..n \tag{5.19}$$

5.5.3 Tiempo de finalización de la fase Reduce

La fase *Reduce* consiste en la ejecución en paralelo de un número de instancias de la función *Reduce* (R) que compiten por un número de slots de *Reduce* (S^R). En este caso, el tiempo medio de finalización de una tarea *Reduce*, t_{Rd}, depende tanto de parámetros a nivel de aplicación como a nivel de sistema. Concretamente éste depende del tiempo de cómputo de la función *Reduce*, t_{Rc}, y del tiempo necesario para escribir los resultados de dicha función en el sistema de ficheros HDFS. Este último se obtiene aplicando la función obtenida durante la calibración del sistema, $f_{Rd}(x)$, en la que al sustituir la variable x por el valor concreto de la cantidad de datos escritos por cada función *Reduce*, se tiene como resultado el tiempo de escritura en HDFS. Así pues:

$$t_{Rd} = t_{Rc} + f_{Rd}(d_{Rd}). \tag{5.20}$$

Por tanto, la estimación del tiempo de finalización de la fase *Reduce* en el mejor caso (límite inferior) se aproxima de la siguiente forma:

$$T_{Rd}(i) = \frac{R}{S^R} \times t_{Rd}, \quad \forall i = 1..n. \tag{5.21}$$

De igual forma, el tiempo máximo de finalización de una tarea *Reduce* se obtiene como:

$$t_{Rd}^{max} = t_{Rc}^{max} + f_{Rd}^{max}(d_{Rd}), \tag{5.22}$$

donde aplicando la adaptación del teorema de *makespan*, se obtiene la siguiente expresión para el peor caso (límite superior):

$$T_{Rd}^{max}(i) = \left(\left\lceil \frac{R}{S^R} \right\rceil - 1 \right) \times t_{Rd} + t_{Rd}^{max}, \quad \forall i = 1..n. \tag{5.23}$$

5.5.4 Iteraciones complejas

Hasta ahora hemos hecho una suposición importante acerca de las aplicaciones: cada iteración implica un solo trabajo MapReduce que es computacionalmente similar al resto de iteraciones. Sin embargo, en la práctica puede suceder que las iteraciones sean complejas e impliquen una serie de pasos expresados como trabajos separados de MapReduce (por ejemplo, PageRank o Connected Components). El tratamiento de este tipo de aplicaciones MapReduce dentro de nuestro esquema de predicción requiere de un pequeño ajuste.

Con el propósito de facilitar la comprensión de este ajuste vamos ayudarnos de un pequeño ejemplo. Supongamos que partimos de una aplicación MapReduce iterativa, que requiere de tres tipos de trabajos MapReduce distintos en cada iteración:

- El primer trabajo lo denominaremos A.
- Al segundo tipo de trabajo lo denominaremos B.
- Por último, el tercer tipo de trabajo lo denominaremos C.

La ejecución de la aplicación completa sería una secuencia de trabajos de la siguiente forma:

$$\overbrace{\underset{1}{A}\ \underset{2}{B}\ \underset{3}{C}}^{1}\ \overbrace{\underset{4}{A}\ \underset{5}{B}\ \underset{6}{C}}^{2}\ \overbrace{\underset{7}{A}\ \underset{8}{B}\ \underset{9}{C}}^{3}\ \overbrace{\underset{10}{A}\ \underset{11}{B}\ \underset{12}{C}}^{4}\cdots$$

Como se puede observar, en este caso habría 12 trabajos MapReduce agrupados en 4 iteraciones compuestas por 3 trabajos, uno de cada tipo.

Con el objetivo de aplicar nuestra estrategia de predicción, y desde el punto de vista exclusivamente del tiempo de finalización, la secuencia anterior es completamente equivalente a:

$$\overbrace{\underset{1}{A}\ \underset{2}{A}\ \underset{3}{A}\ \underset{4}{A}}^{1}\ \overbrace{\underset{5}{B}\ \underset{6}{B}\ \underset{7}{B}\ \underset{8}{B}}^{2}\ \overbrace{\underset{9}{C}\ \underset{10}{C}\ \underset{11}{C}\ \underset{12}{C}}^{3}\cdots$$

en donde lo que hemos hecho es reordenar la secuencia anterior. De esta forma, tenemos igualmente 12 trabajos MapReduce, agrupados según su tipo y número de iteraciones ejecutadas. Sin embargo esta vez, en lugar de aparecer tal cual lo harían en la ejecución de la aplicación, se han reordenado agrupando todos los trabajos del mismo tipo de forma consecutiva, por lo que aparecen 3 bloques de 4 iteraciones homogéneas en cada bloque.

De este modo se consigue pasar de una situación compleja en la que habían iteraciones MapReduce compuestas por un conjunto de trabajos diferentes, a una situación en la que tenemos una secuencia de 3 trabajos MapReduce con iteraciones idénticas, y a las que se puede aplicar nuestra estrategia de predicción.

Generalizando el ejemplo anterior, supongamos que una aplicación MapReduce iterativa está compuesta por k iteraciones de m trabajos MapReduce distintos. Esto significa que el trabajo i-ésimo se comporta, a nivel de tiempo de finalización, igual que el trabajo $(i + m)$-ésimo. Para estimar el tiempo de finalización de k iteraciones aplicamos nuestra estrategia de predicción a m aplicaciones

MapReduce, en donde cada una de ellas está compuesta por k iteraciones iguales. Con esto la predicción del tiempo de finalización se obtendrá como:

$$T = \sum_{j=1}^{m} T_j, \tag{5.24}$$

donde T_j es la predicción del tiempo de finalización de una aplicación MapReduce compuesta por k iteraciones iguales.

5.6 Evaluación

En esta sección se evalúa la metodología de predicción descrita anteriormente. Para ello, se utilizan cuatro aplicaciones MapReduce iterativas reales, extraídas de *benchmarks* estandarizados como *HiBench* [53] y *BigDataBench* [96], las cuales se ejecutan sobre distintos entornos de nube híbrida. De esta forma, se abarca todo el abanico de posibilidades en cuanto a tipos de aplicación MapReduce iterativas (*Map*-intensivas o *Reduce*-intensivas). Además, algunas de ellas son ejemplos de aplicaciones con iteraciones complejas.

El entorno experimental sobre el que se han llevado a cabo los experimentos es el mismo que se describe en la sección 2.4 del capítulo 2, compuesto por 4 nodos físicos de baja capacidad (*thin*), encargados de las tareas de gestión de OpenStack, junto con otros 4 nodos físicos de gran capacidad (*fat*), capaces de albergar 4 MVs (4 vCPUs, 16 GB RAM y 100 GB HDD) cada uno, teniendo un total de 16 MVs homogéneas. Estos nodos se han separado en dos infraestructuras de nube independientes a través de dos instancias de OpenStack (2 nodos *thin* y 1 nodo *fat* para la nube que realiza la función de nube privada, y 2 nodos *thin* y 3 nodos *fat* para la nube con función de nube pública). Cada nube consta de un nodo *thin* que controla el tráfico de salida de red, permitiendo gestionar el ancho de banda entre nubes. A lo largo de la evaluación se comparan cuatro enfoques:

- **On-premise - Real:** corresponde al caso en el que todas las MVs están *on-premise* y no existe ningún *enlace débil* que pueda convertirse en un cuello de botella para la E/S del HDFS. En este caso, se utiliza un despliegue de Hadoop estándar. Este enfoque se utiliza como un límite inferior de tiempo o línea base de comparación, mostrando lo que sucedería en un escenario ideal donde el usuario no tuviera limitaciones de coste y pudiera permitirse adquirir recursos adicionales *on-premise* para lograr un mejor rendimiento, en lugar de adoptar una solución híbrida.
- **ARIA** (*Automatic Resource Inference and Allocation*) [92]: es uno de los modelos más utilizados hasta el momento para calcular el tiempo de ejecución de un trabajo MapReduce en un clúster homogéneo. Se basa en su propia caracterización (parámetros a nivel de aplicación) para realizar una planificación de trabajos que cumplan unos determinados límites de tiempo de ejecución. Utilizamos ARIA como base comparativa de los modelos de predicción que se hayan actualmente en la literatura, para mostrar que las técnicas para estimar el tiempo de ejecución de una aplicación MapReduce iterativa para un entorno único *on-premise* no son lo suficientemente precisas para su uso en un escenario de *cloud bursting*. Esto demuestra la necesidad de desarrollar un modelo como el presentado en esta tesis, especializado en entornos híbridos.

- **Híbrido - Real:** corresponde al escenario en el que se obtiene el tiempo de ejecución real para una aplicación MapReduce iterativa, usando un despliegue de *cloud bursting* compuesto por una serie de MVs *on-premise* y *off-premise*. La versión de Hadoop utilizada en este caso está optimizada para un despliegue de *cloud bursting*, usando la estrategia de *Reequilibrio asíncrono con rack local* descrita en la sección 4.3.4. Utilizamos este enfoque para comparar la precisión de nuestras predicciones obtenidas sobre los mismos escenarios.
- **Híbrido - Estimado:** corresponde a un escenario de *cloud bursting* en el que se obtiene una estimación del tiempo de finalización de una aplicación MapReduce iterativa a través de nuestra metodología de predicción (sección 5.5) que, al igual que el enfoque anterior, utiliza la estrategia de *Reequilibrio asíncrono con rack local*.

5.6.1 Escenarios

Dado el entorno experimental, se presentan varios escenarios donde se realiza una escalabilidad horizontal a través de la incorporación de nuevas MVs *off-premise* a las ya existentes MVs *on-premise*.

Se parte de un primer escenario con 4 MVs ubicadas en uno de los nodos con OpenStack, que realiza la función de nube privada u *on-premise*, en las cuales está desplegado Hadoop MapReduce. En esta configuración, una de las MVs realiza la función de maestro y las otras 3 MVs restantes de esclavos. Por tanto, a nivel de rendimiento de aplicaciones, partimos de un escenario *on-premise* donde hay 3 MVs destinadas a cálculo.

Los otros 3 nodos físicos restantes, con otra instancia independiente de OpenStack instalada, forman la nube pública, con capacidad para almacenar hasta 12 MVs *off-premise*. Estas MVs tienen instalado y configurado Hadoop MapReduce para tomar parte como esclavos cuando se añadan al despliegue de Hadoop de las MVs *on-premise*.

Por tanto, se evalúan cuatro escenarios de nube híbrida partiendo de 3 MVs *off-premise* que se añaden a las 3 MVs *on-premise* ya existentes, y aumentando de tres en tres el número de MVs *off-premise*. De esta forma, los escenarios que se utilizan para la experimentación son:

- Únicamente una nube privada (*on-premise*) con 3 MVs.
- Una nube privada compuesta por 3 MVs *on-premise* extendida con una nube pública compuesta por 3 MVs *off-premise*.
- Una nube privada compuesta por 3 MVs *on-premise* extendida con una nube pública compuesta por 6 MVs *off-premise*.
- Una nube privada compuesta por 3 MVs *on-premise* extendida con una nube pública compuesta por 9 MVs *off-premise*.
- Una nube privada compuesta por 3 MVs *on-premise* extendida con una nube pública compuesta por 12 MVs *off-premise*.

Por otro lado, el rendimiento del *enlace débil* que separa las dos nubes puede ser controlado por software, lo cual nos permite limitar la red a distintas velocidades. En esta evaluación se comparan

los resultados con dos configuraciones distintas de *enlace débil*, una a 100 Mbps y otra a 1 Gbps, para los cuatro escenarios de nube híbrida propuestos.

5.6.2 Calibración del sistema

En primer lugar se ha realizado la calibración del sistema, tal y como se ha descrito en la sección 5.3, para los todos los escenarios propuestos en la sección 5.6.1.

El test sintético de calibración parte de una pequeña entrada de datos de 64 MB, que contiene información de texto de páginas de Wikipedia, la cual se replica automáticamente tantas veces como sea necesario para cubrir todos los slots de *Map* disponibles de un escenario concreto, de manera que se consigue una onda completa de tareas *Map* en cada caso.

Se trata de un test iterativo, el cual consta de dos iteraciones complejas principales para extraer una función de aproximación de la fase *Shuffle*, y otra para la fase de *Reduce*:

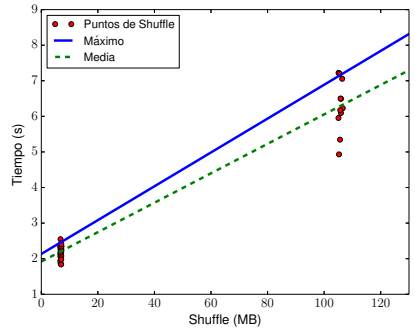
- La primera iteración para la extracción de la función de *Shuffle* lanza distintos trabajos MapReduce fijando la salida de la fase *Reduce* a un tamaño muy reducido, para que influya lo menos posible en la posterior extracción de información de dicha fase, y variando en cada trabajo MapReduce la entrada de la fase *Shuffle* (esto se consigue variando la salida de las tareas *Map*).
- La segunda iteración, encargada de la extracción de la función de *Reduce*, lanza distintos trabajos MapReduce, fijando en este caso la entrada del *Shuffle* en un tamaño insignificante para, de igual forma, no influir en el resultado, variando en cada trabajo la salida de la fase *Reduce*.

Cabe destacar que la función de aproximación de la fase *Reduce* se forma a través de dos regresiones lineales. Inicialmente, y hasta 7 MB de salida de la fase *Reduce*, para un tamaño de *Shuffle* fijado, la regresión sigue una pendiente distinta a la obtenida con salidas mayores de la fase *Reduce*. Esto es debido a que la sobrecarga producida por la inicialización de la escritura de las tareas *Reduce* para tamaños reducidos es mayor que el tiempo que necesita para escribir su salida. Por tanto, inicialmente se producen tiempos de escritura mayores que no son despreciables a la hora de obtener una buena predicción.

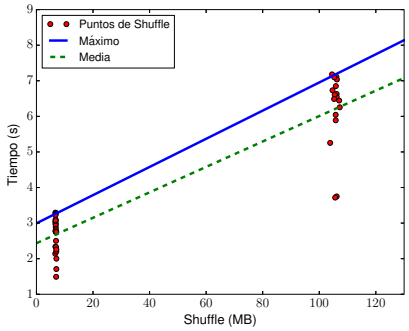
Todos los resultados son procesados y analizados a través de las herramientas H-Profile y H-HDFS de H-Stat, las cuales se encargan de obtener todos los parámetros de calibración descritos en la Tabla 5.1 para cada uno de los escenarios propuestos.

A continuación se muestran las gráficas de las funciones de aproximación de la fase *Shuffle* y de la escritura de la fase *Reduce* para todos los escenarios de nube híbrida propuestos, para un *enlace débil* de 1 Gbps (Figura 5.3), y de 100 Mbps (Figura 5.4).

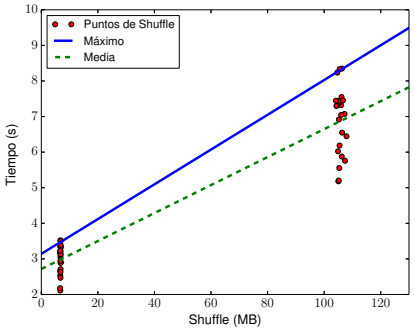
Además, en las Tablas 5.3 y 5.4 se muestran con más detalle todos los parámetros de calibración extraídos para cada uno de los escenarios de nube híbrida.



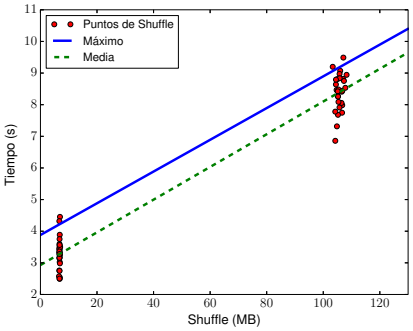
(a) *Shuffle* 3-On-3-Off



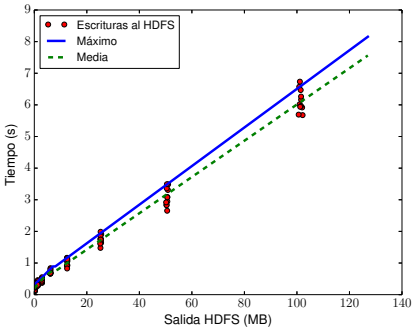
(b) *Shuffle* 3-On-6-Off



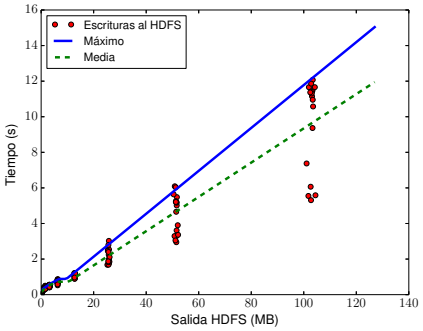
(c) *Shuffle* 3-On-9-Off



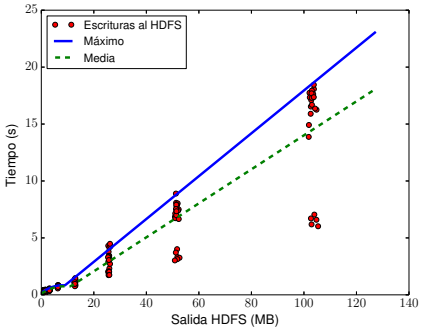
(d) *Shuffle* 3-On-12-Off



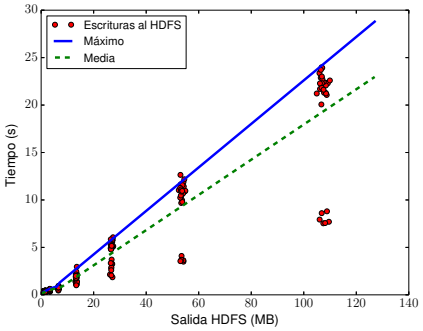
(e) *Reduce* 3-On-3-Off



(f) *Reduce* 3-On-6-Off

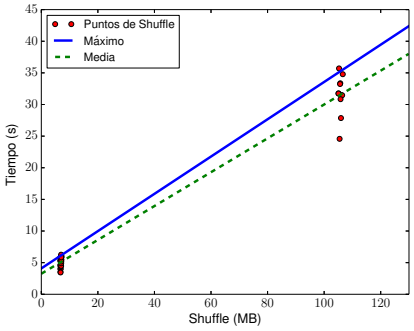


(g) *Reduce* 3-On-9-Off

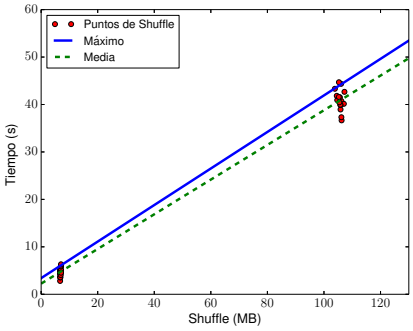


(h) *Reduce* 3-On-12-Off

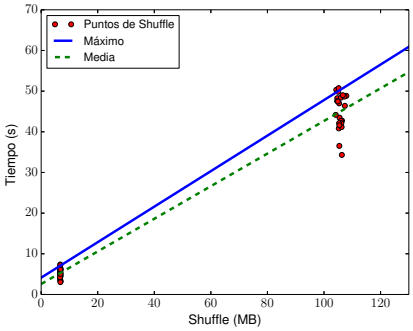
Figura 5.3: Calibración del sistema de la fase *Shuffle* (a), (b), (c) y (d) y *Reduce* (e), (f), (g), (h) para los cuatro escenarios propuestos, utilizando un *enlace débil* de 1 Gbps.



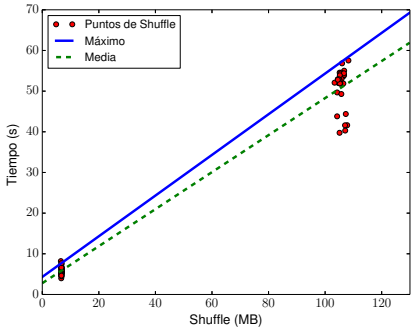
(a) *Shuffle* 3-On-3-Off



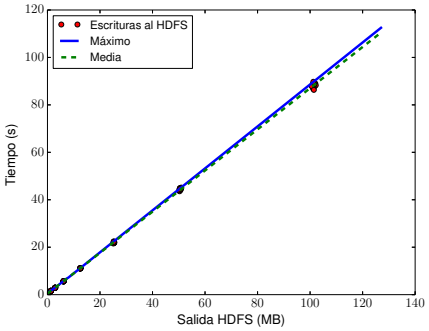
(b) *Shuffle* 3-On-6-Off



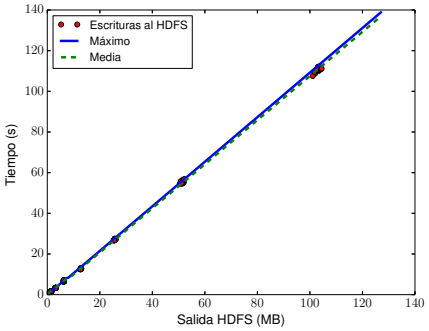
(c) *Shuffle* 3-On-9-Off



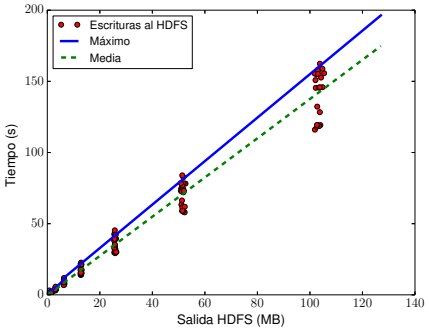
(d) *Shuffle* 3-On-12-Off



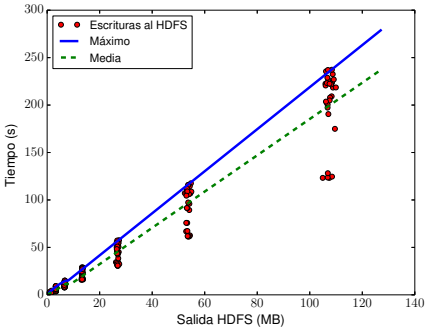
(e) *Reduce* 3-On-3-Off



(f) *Reduce* 3-On-6-Off



(g) *Reduce* 3-On-9-Off



(h) *Reduce* 3-On-12-Off

Figura 5.4: Calibración del sistema de la fase *Shuffle* (a), (b), (c) y (d) y *Reduce* (e), (f), (g), (h) para los cuatro escenarios propuestos, utilizando un *enlace débil* de 100 Mbps.

Parámetro	**3-On-3-Off**	**3-On-6-Off**	**3-On-9-Off**	**3-On-12-Off**
S_{on}^{M}	10	10	10	10
S_{off}^{M}	12	24	36	48
S^{R}	11	17	23	29
α	0	0	0,57	1,47
β	98 Mbps	96 Mbps	95 Mbps	95 Mbps
$f_{Sh}(d_{Sh})$	$f(x) = 0{,}041x + 1{,}92$	$f(x) = 0{,}035x + 2{,}43$	$f(x) = 0{,}039x + 2{,}71$	$f(x) = 0{,}05x + 2{,}92$
$f_{Sh}^{max}(d_{Sh})$	$f(x) = 0{,}047x + 2{,}13$	$f(x) = 0{,}039x + 2{,}99$	$f(x) = 0{,}048x + 3{,}14$	$f(x) = 0{,}05x + 3{,}87$
$f_{Rd}(d_{Rd})$	$f(x) = \begin{cases} f_1 : -0{,}0045x^2 + 0{,}112x + 0{,}19 & \text{si } x < 7 \\ f_2 : 0{,}057x + 0{,}28 & \text{si } x > 7 \\ max(f_1, f_2) & \text{si } x = 7 \end{cases}$	$f(x) = \begin{cases} f1 : -0{,}0092x^2 + 0{,}146x + 0{,}15 & \text{si } x < 7 \\ f2 : 0{,}096x + 0{,}28 & \text{si } x > 7 \\ max(f1, f2) & \text{si } x = 7 \end{cases}$	$f(x) = \begin{cases} f1 : -0{,}0079x^2 + 0{,}145x + 0{,}12 & \text{si } x < 7 \\ f2 : 0{,}149x - 0{,}92 & \text{si } x > 7 \\ max(f1, f2) & \text{si } x = 7 \end{cases}$	$f(x) = \begin{cases} f1 : -0{,}0096x^2 + 0{,}156x + 0{,}1 & \text{si } x < 7 \\ f2 : 0{,}185x - 0{,}598 & \text{si } x > 7 \\ max(f1, f2) & \text{si } x = 7 \end{cases}$
$f_{Rd}^{max}(d_{Rd})$	$f(x) = \begin{cases} f1 : 0{,}0009x^2 + 0{,}075x + 0{,}32 & \text{si } x < 7 \\ f2 : 0{,}061x + 0{,}4 & \text{si } x > 7 \\ max(f1, f2) & \text{si } x = 7 \end{cases}$	$f(x) = \begin{cases} f1 : -0{,}0039x^2 + 0{,}119x + 0{,}24 & \text{si } x < 7 \\ f2 : 0{,}12x - 0{,}27 & \text{si } x > 7 \\ max(f1, f2) & \text{si } x = 7 \end{cases}$	$f(x) = \begin{cases} f1 : -0{,}0099x^2 + 0{,}162x + 0{,}18 & \text{si } x < 7 \\ f2 : 0{,}188x - 0{,}86 & \text{si } x > 7 \\ max(f1, f2) & \text{si } x = 7 \end{cases}$	$f(x) = \begin{cases} f1 : -0{,}012x^2 + 0{,}199x + 0{,}11 & \text{si } x < 7 \\ f2 : 0{,}229x - 0{,}323 & \text{si } x > 7 \\ max(f1, f2) & \text{si } x = 7 \end{cases}$

Tabla 5.3: Parámetros de calibración del sistema para cada escenario, utilizando un *enlace débil* de 1 Gbps.

Parámetro	**3-On-3-Off**	**3-On-6-Off**	**3-On-9-Off**	**3-On-12-Off**
S_{on}^{M}	10	10	10	10
S_{off}^{M}	12	24	36	48
S^{R}	11	17	23	29
α	0,57	0,88	1,57	2,22
β	7,7 Mbps	7,65 Mbps	7,53 Mbps	7,48 Mbps
$f_{Sh}(d_{Sh})$	$f(x) = 0{,}26x + 3{,}24$	$f(x) = 0{,}36x + 2{,}24$	$f(x) = 0{,}4x + 2{,}56$	$f(x) = 0{,}45x + 2{,}77$
$f_{Sh}^{max}(d_{Sh})$	$f(x) = 0{,}29x + 4{,}08$	$f(x) = 0{,}38x + 3{,}4$	$f(x) = 0{,}43x + 4{,}12$	$f(x) = 0{,}5x + 4{,}31$
$f_{Rd}(d_{Rd})$	$f(x) = \begin{cases} f1 : 0{,}0012x^2 + 0{,}839x + 0{,}45 & \text{si } x < 7 \\ f2 : 0{,}86x + 0{,}268 & \text{si } x > 7 \\ max(f1, f2) & \text{si } x = 7 \end{cases}$	$f(x) = \begin{cases} f1 : 0{,}023x^2 + 0{,}844x + 0{,}43 & \text{si } x < 7 \\ f2 : 1{,}08x - 0{,}919 & \text{si } x > 7 \\ max(f1, f2) & \text{si } x = 7 \end{cases}$	$f(x) = \begin{cases} f1 : 0{,}048x^2 + 0{,}954x + 0{,}41 & \text{si } x < 7 \\ f2 : 1{,}37x - 0{,}2186 & \text{si } x > 7 \\ max(f1, f2) & \text{si } x = 7 \end{cases}$	$f(x) = \begin{cases} f1 : 0{,}0419x^2 + 1{,}31x + 0{,}24 & \text{si } x < 7 \\ f2 : 1{,}911x - 5{,}82 & \text{si } x > 7 \\ max(f1, f2) & \text{si } x = 7 \end{cases}$
$f_{Rd}^{max}(d_{Rd})$	$f(x) = \begin{cases} f1 : 0{,}012x^2 + 0{,}757x + 0{,}65 & \text{si } x < 7 \\ f2 : 0{,}884x + 0{,}137 & \text{si } x > 7 \\ max(f1, f2) & \text{si } x = 7 \end{cases}$	$f(x) = \begin{cases} f1 : 0{,}039x^2 + 0{,}79x + 0{,}52 & \text{si } x < 7 \\ f2 : 1{,}09x - 0{,}42 & \text{si } x > 7 \\ max(f1, f2) & \text{si } x = 7 \end{cases}$	$f(x) = \begin{cases} f1 : 0{,}0916x^2 + 0{,}946x + 1{,}22 & \text{si } x < 7 \\ f2 : 1{,}52x + 2{,}31 & \text{si } x > 7 \\ max(f1, f2) & \text{si } x = 7 \end{cases}$	$f(x) = \begin{cases} f1 : -0{,}11x^2 + 2{,}88x + 0{,}24 & \text{si } x < 7 \\ f2 : 2{,}21x - 2{,}68 & \text{si } x > 7 \\ max(f1, f2) & \text{si } x = 7 \end{cases}$

Tabla 5.4: Parámetros de calibración del sistema para cada escenario, utilizando un *enlace débil* de 100 Mbps.

5.6.3 Predicción de la aplicación IGrep

Iterative Grep (IGrep) es una aplicación utilizada en el análisis de grandes volúmenes de textos no estructurados. Consiste en un conjunto de trabajos ***Grep*** independientes que identifica todas las coincidencias de cadenas de una expresión regular dada, que posteriormente son ordenadas según el número de coincidencias. La naturaleza iterativa aparece porque los datos de entrada sobre los que se realiza la búsqueda son siempre los mismos, pero la expresión regular cambia como un refinamiento de la iteración anterior. Por ejemplo, uno puede querer contar cuántas veces está presente un cierto concepto en los artículos de Wikipedia, y, dependiendo del resultado, preparar la siguiente expresión regular para encontrar correlaciones con otro concepto. Dado que la expresión regular es típicamente un patrón exacto, la salida de los correlacionadores es muy simple y consiste en un pequeño número de pares clave-valor que se reducen a un solo par clave-valor. Hay que destacar que esta aplicación está compuesta por iteraciones complejas, presentando en este caso concreto dos trabajos diferentes por iteración: uno en el que se realiza la fase de búsqueda y otro en el que se realiza una ordenación de los datos obtenidos en la fase anterior.

En este experimento utilizamos un conjunto de datos de 20 GB de artículos extraídos de Wikipedia [99] como datos de entrada, y 50 palabras clave para ejecutar 50 iteraciones sobre estos datos de entrada. El tamaño de los datos a tratar en la fase *Shuffle* para cada iteración es de menos de 1 MB. Esto implica que es la fase *Map* la que más aporta al tiempo de ejecución. De ahí que consideremos a esta aplicación como un ejemplo de uso de *Map*-intensivo.

Parámetros de caracterización

Para extraer los parámetros de caracterización de esta aplicación se ha ejecutado una instancia de ésta sobre las MVs *on-premise*, utilizando un fichero con la misma entrada de datos y número de iteraciones indicadas anteriormente para obtener una mayor estadística y precisión a la hora de caracterizar cada iteración.

Tal y como se ha descrito, esta aplicación tiene dos tipos de trabajo distintos por iteración, lo que hace que se deba parametrizar cada uno de ellos de forma independiente. En la Tabla 5.5 se muestran los parámetros de caracterización obtenidos para IGrep.

Parámetro	Fase de búsqueda	Fase de ordenación
M	150	1
R	1	1
t_{Mp}	7,54 s	3,3 s
t_{Mp}^{max}	15,73 s	5,26 s
t_{Rc}	1,72 s	1,255 s
t_{Rc}^{max}	2,79 s	4,17 s
d_{Sh}	3,19 KB	21 Bytes
d_{Rd}	108 Bytes	12,8 Bytes

Tabla 5.5: Parámetros de caracterización de la aplicación IGrep.

Resultados

Los resultados obtenidos para la aplicación IGrep se muestran en la Tabla 5.6. Realizando un primer análisis podemos observar que IGrep es una aplicación escalable: al aumentar los recursos se obtienen incrementos de velocidad, aunque al mantener la dimensión de los datos de entrada solo se consigue un incremento de velocidad 3,19×, mientras que los recursos utilizados se han incrementado en 5× (6483 s utilizando 3 MVs frente a 2035 s utilizando 15 MVs).

Tiempos reales para una nube *on-premise*					
Arquitectura	**3 MVs**	**6 MVs**	**9 MVs**	**12 MVs**	**15 MVs**
	6483	**3310**	**2569**	**2256**	**2035**

Predicción utilizando el modelo ARIA				
Límite superior de la predicción	4226	3324	2892	2639
Valor medio de la predicción	**3421**	**2508**	**2071**	**1816**
Límite inferior de la predicción	2615	1692	1251	992

Predicción utilizando nuestra metodología				
Nube híbrida con un *enlace débil* de **1 Gbps**				
Arquitectura	**3-On 3-Off**	**3-On 6-Off**	**3-On 9-Off**	**3-On 12-Off**
Tiempos reales	**3474**	**2734**	**2422**	**2227**
Límite superior de la predicción	3953	3299	3087	2947
Valor medio de la predicción	**3548**	**2803**	**2563**	**2453**
Límite inferior de la predicción	3144	2307	2038	1959
Nube híbrida con un *enlace débil* de **100 Mbps**				
Arquitectura	**3-On 3-Off**	**3-On 6-Off**	**3-On 9-Off**	**3-On 12-Off**
Tiempos reales	**4528**	**3876**	**3705**	**3407**
Límite superior de la predicción	4795	4282	4345	4141
Valor medio de la predicción	**4336**	**3745**	**3719**	**3599**
Límite inferior de la predicción	3877	3208	3093	3057

Tabla 5.6: Tiempos de finalización reales y de predicción para la aplicación IGrep (expresados en segundos). Para los tiempos de predicción se muestran tanto los límites inferiores y superiores, como el valor medio de predicción.

Al mismo tiempo, esta aplicación también presenta una alta variabilidad entre las tareas *Map*, lo que afecta a los límites inferiores y superiores de la predicción. Así, en la predicción obtenida con ARIA para 15 MVs (3-On 12-Off) el límite superior se sobreestima en un 30 % con respecto a los tiempos obtenidos solo *on-premise*, mientras que el límite inferior se subestima en un 50 %. Si analizamos estos mismos datos con 15 MVs con respecto a los resultados reales obtenidos en una nube híbrida con *enlaces débiles* de 1 Gbps y 100 Mbps, se cometen errores del 18 % y del 55 % para los límites superiores e inferiores en 1 Gbps de *enlace débil*, y de un 23 % y un 70 % para los límites superiores e inferiores para 100 Mbps.

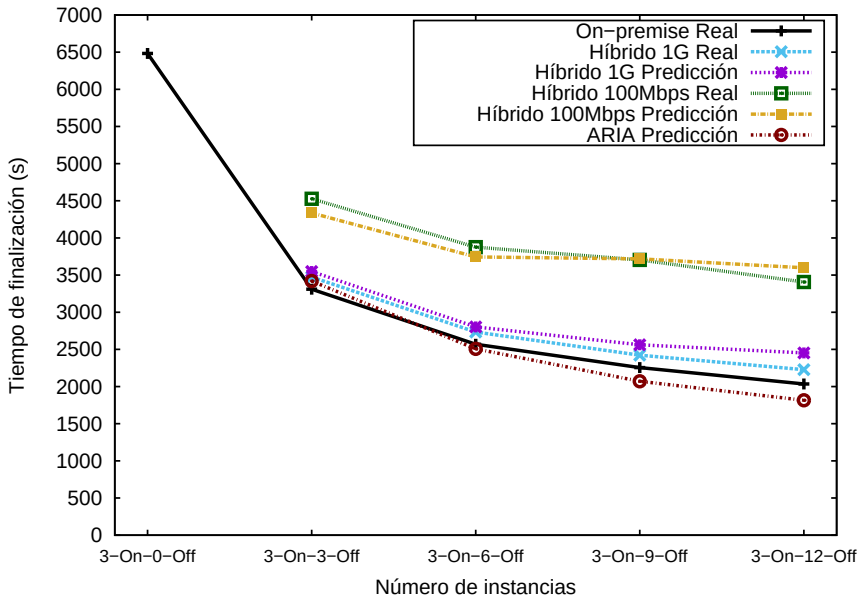


Figura 5.5: Tiempos de ejecución reales y estimados para la aplicación IGrep

Arquitectura	**3-On 3-Off**	**3-On 6-Off**	**3-On 9-Off**	**3-On 12-Off**
Nube híbrida con un *enlace débil* de 1 Gbps				
Porcentajes de error (%) del valor medio estimado con ARIA				
En la fase Map	4.1	-1.8	-8.4	-12.8
En la fase Reduce	-34.9	-36.5	-37.2	-37.5
Error Total	**-1.5**	**-8.3**	**-14.5**	**-18.5**
Porcentajes de error (%) del valor medio estimado con nuestra metodología				
En la fase Map	3.9	2.3	5.6	8.7
En la fase Reduce	-8.3	3.6	6.7	15.3
Error Total	**2.1**	**2.5**	**5.8**	**10.1**
Nube híbrida con un *enlace débil* de 100Mbps				
Porcentajes de error (%) del valor medio estimado con ARIA				
En la fase Map	-18.2	-30.0	-40.0	-43.1
En la fase Reduce	-56.3	-57.4	-59.4	-59.0
Error Total	**-24.5**	**-35.3**	**-44.1**	**-46.7**
Porcentajes de error (%) del valor medio estimado con nuestra metodología				
En la fase Map	-2.6	1.9	5.1	13.4
En la fase Reduce	-12.4	-25.3	-17.2	-20.8
Error Total	**-4.2**	**-3.4**	**0.4**	**5.6**

Tabla 5.7: IGrep: Precisión de la predicción media vs. el tiempo de finalización de un despliegue híbrido real detallado por fase.

Nuestra metodología de predicción muestra un comportamiento mejor en lo que respecta a estos límites. Así, para 15 MVs y 1 Gbps de *enlace débil* obtiene unos errores del 32 % y del 12 % para los límites superiores e inferiores, con respecto a los resultados reales de una arquitectura de nube híbrida. Además, en el caso de 100 Mbps se obtienen unos errores del 21 % y del 10 % para los mismos límites. Como se puede observar, en prácticamente todos los casos obtenemos predicciones más precisas que las que se obtienen utilizando ARIA, considerando además que ARIA se ha comparado con una arquitectura únicamente *on-premise*.

Sin embargo, la métrica más importante es la de la predicción media, que debe ser lo más ajustada posible a la realidad. En la Figura 5.5, se muestra una gráfica de los tiempos reales junto con las predicciones medias. Como se puede observar, a pesar de las grandes diferencias que se han indicado entre el límite superior e inferior de ARIA, su estimación media está bastante cercana a los valores reales obtenidos *on-premise*. Sin embargo, su estimación para una nube híbrida produce un error de hasta un 18,5 % para 1 Gbps y un 46 % para 100 Mbps. Esto demuestra claramente la influencia del cuello de botella que representa el *enlace débil* durante la ejecución de la aplicación. Estos resultados contrastan con los errores obtenidos con nuestra metodología de predicción, donde el error máximo que se produce es del 10,1 % para 1 Gbps y del 5,6 % para 100 Mbps.

En la Tabla 5.7 se muestran los porcentajes de error de predicción para cada enfoque con más detalle, desglosado en la fase *Map* y *Reduce*. Además se indica el porcentaje de error medio cometido en cada caso con respecto a su medición real en una nube híbrida. Conviene destacar que, al ser una aplicación *Map*-intensiva, el peso del porcentaje de error de la fase *Map* es mucho mayor que el del error producido en la fase *Reduce*.

Analizando los resultados obtenidos por ARIA con 15 MVs (3-On 12-Off), podemos observar que para 1 Gbps se produce un error en la fase *Map* menor del 13 %, y de un 37 % en la fase *Reduce*. A pesar de ello, el error total está por debajo del 19 %. Para un *enlace débil* de 100 Mbps, como hemos visto anteriormente, el error que se produce tanto para la fase *Map* como *Reduce* aumenta considerablemente, ya que ARIA no se adapta a una situación donde el cuello de botella producido por el *enlace débil* es mucho mayor, produciendo un error total de hasta un 46,7 % con 15 MVs.

Analizando nuestra predicción podemos observar que para 1 Gbps los errores producidos para la fase *Map* son menores del 9 %, con lo que los resultados finales producen un error total menor del 10,1 % en el peor de los casos, incluso teniendo un error del 15 % en la fase *Reduce*. Algo similar ocurre para 100 Mbps donde, incluso aumentando el error cometido para la fase *Reduce* hasta un 25 %, se obtiene un error total menor del 5,6 % en el peor de los casos.

A nivel general se puede observar que si un usuario decide mejorar el rendimiento de su aplicación utilizando más MVs *off-premise* a través de un modelo de *cloud bursting*, podría conseguir un incremento de velocidad de hasta 2,9× añadiendo 12 MVs *off-premise* a su plataforma de 3 MVs *on-premise*, utilizando una conexión entre la parte *on-premise* y *off-premise* con un ancho de banda de 1 Gbps, o hasta 1,9× utilizando 100 Mbps. Estos datos los podría obtener con antelación a la ejecución de la aplicación, mediante la utilización de nuestra metodología de predicción, cometiendo un error aceptable. Con ello podría determinar de antemano el número de MVs a contratar en una nube pública, y también el ancho de banda necesario para conseguir un determinado incremento de velocidad.

5.6.4 Predicción de la aplicación K-Means

K-Means [17] es una aplicación ampliamente utilizada para la cuantificación vectorial en el procesamiento de señales, análisis de clústeres en minería de datos, clasificación de patrones y extracción de rasgos para el aprendizaje automático, etc. Esta aplicación se describe con más detalle en la sección 4.4. Su versión MapReduce se implementa en el proyecto *Mahout* [41] de la fundación Apache. Además, esta aplicación forma parte del *benchmark* para BigData *HiBench* de Intel.

En este experimento utilizamos 20 GB de entrada de datos que son procesados en 30 iteraciones simples, es decir, cada iteración posee un único trabajo MapReduce. En este caso nos encontramos con una aplicación iterativa sencilla, en el que todas las iteraciones tienen el mismo comportamiento computacional. El peso de la ejecución para este tipo de aplicación iterativa recae sobre la fase *Map*. Por tanto, igual que IGrep, esta aplicación es de tipo *Map*-intensiva.

Parámetros de caracterización

Para obtener los parámetros de caracterización de K-Means se ha ejecutado esta aplicación sobre las 3 MVs *on-premise*, utilizando el mismo tamaño de datos de entrada y número de iteraciones descrito anteriormente para obtener una mejor precisión en la caracterización de esta aplicación.

En la Tabla 5.8 se muestran los parámetros obtenidos de la caracterización de K-Means, mostrando la media y máximo según el parámetro.

Parámetro	**Iteración**
M	150
R	1
t_{Mp}	13,96 s
t_{Mp}^{max}	16,08 s
t_{Rc}	2,24 s
t_{Rc}^{max}	3,46 s
d_{Sh}	433 KB
d_{Rd}	2,65 KB

Tabla 5.8: Parámetros de caracterización de K-Means.

Resultados

K-Means es una aplicación que también muestra un alto grado de escalabilidad consiguiendo un 77 % de mejora al incrementar 5× el número de MVs en una plataforma solo *on-premise*, como se muestra en la Tabla 5.9 de resultados. En esa misma tabla se muestran todas las predicciones realizadas para K-Means, utilizando tanto ARIA como nuestro enfoque.

Analizando los resultados de dicha tabla, podemos observar que el comportamiento de K-Means es muy similar al de IGrep, por lo que se obtienen unos resultados similares. A diferencia de IGrep,

cada iteración de K-Means está compuesta por un solo trabajo. Por tanto, permite mayor estabilidad en cuanto a la predicción, lo cual se traduce en unos resultados de predicción más precisos.

La predicción media, tanto para el escenario híbrido como para ARIA, se muestra gráficamente en la Figura 5.6, donde se aprecia un error de predicción de casi el 50 % en el caso de ARIA para una nube híbrida con un *enlace débil* de 100 Mbps. Para el caso en el que se utiliza un *enlace débil* de 1 Gbps el error es aún significativo, llegando al 16 %. Sin embargo, utilizando nuestra metodología de predicción se obtienen unos resultados de predicción mucho mejores, con errores máximos menores del 4 % en 1 Gbps y del 6 % en 100 Mbps.

Tiempos reales para una nube *on-premise*					
Arquitectura	**3 MVs**	**6 MVs**	**9 MVs**	**12 MVs**	**15 MVs**
	6471	**3024**	**2159**	**1786**	**1511**

Predicción utilizando el modelo ARIA				
Límite superior de la predicción	3501	2499	2020	1739
Valor medio de la predicción	**3185**	**2178**	**1696**	**1414**
Límite inferior de la predicción	2870	1857	1373	1089

Predicción utilizando nuestra metodología				
Nube híbrida con un *enlace débil* de **1 Gbps**				
Arquitectura	**3-On 3-Off**	**3-On 6-Off**	**3-On 9-Off**	**3-On 12-Off**
Tiempos reales	**3175**	**2391**	**1992**	**1685**
Límite superior de la predicción	3282	2495	2168	1858
Valor medio de la predicción	**3190**	**2316**	**1954**	**1700**
Límite inferior de la predicción	3098	2137	1741	1543
Nube híbrida con un *enlace débil* de **100 Mbps**				
Arquitectura	**3-On 3-Off**	**3-On 6-Off**	**3-On 9-Off**	**3-On 12-Off**
Tiempos reales	**3872**	**3207**	**2893**	**2743**
Límite superior de la predicción	3900	3374	3266	3091
Valor medio de la predicción	**3775**	**3185**	**3033**	**2902**
Límite inferior de la predicción	3649	2997	2799	2713

Tabla 5.9: Tiempos de finalización reales y de predicción para la aplicación K-Means (expresados en segundos). Para los tiempos de predicción se muestran tanto los límites inferiores y superiores, como el valor medio de predicción.

Analizando con más detalle los errores medios cometidos en cada una de las etapas en la Tabla 5.10, se puede observar que ARIA comete unos errores de predicción que subestiman la fase *Reduce* con respecto al resultado real de nube híbrida con 1 Gbps, con errores de hasta el 45 %. Para el caso de 100 Mbps sus errores están muy por debajo de la medición real para ambas fases *Map* y *Reduce*, hasta un 46,9 % y un 65,1 % respectivamente. Con lo cual, la predicción media total subestima mucho el resultado real, obteniendo errores totales de hasta un 48,5 % con 15 MVs (3-On 12-Off) y 100 Mbps de *enlace débil*.

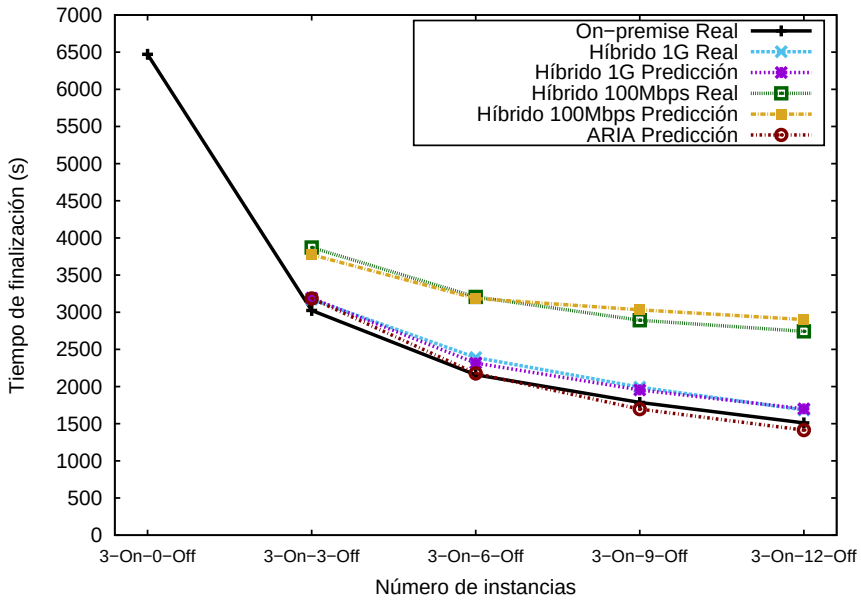


Figura 5.6: Tiempos de ejecución reales y estimados para la aplicación K-Means.

Arquitectura	**3-On 3-Off**	**3-On 6-Off**	**3-On 9-Off**	**3-On 12-Off**

Nube híbrida con un *enlace débil* de 1 Gbps

Porcentajes de error (%) del valor medio estimado con ARIA				
En la fase Map	2,6	-6,3	-12	-12,9
En la fase Reduce	-41,5	-43,5	-45,1	-45,2
Error Total	**0,3**	**-8,9**	**-14,9**	**-16,1**
Porcentajes de error (%) del valor medio estimado con nuestra metodología				
En la fase Map	1	-3,5	-2,3	-0,2
En la fase Reduce	-8,8	1,4	2,6	11
Error Total	**0,5**	**-3,1**	**-1,9**	**0,9**

Nube híbrida con un *enlace débil* de 100Mbps

Porcentajes de error (%) del valor medio estimado con ARIA				
En la fase Map	-14,7	-29,3	-38,9	-46,9
En la fase Reduce	-61,6	-63,9	-65,1	-63,9
Error Total	**-17,7**	**-32,1**	**-41,4**	**-48,5**
Porcentajes de error (%) del valor medio estimado con nuestra metodología				
En la fase Map	-1,6	1,8	7,5	8,6
En la fase Reduce	-15,7	-29,1	-20,8	-21,4
Error Total	**-2,5**	**-0,7**	**4,8**	**5,8**

Tabla 5.10: K-Means: Precisión de la predicción media vs. el tiempo de finalización de un despliegue híbrido real detallado por fase.

En cambio, nuestra metodología de predicción obtiene unos resultados mucho más precisos para ambas fases. En el caso de 1 Gbps, en la fase *Map* se producen errores menores del 3,5 % y del 11 % para la fase *Reduce*, lo que se traduce en unos errores totales para la media que son menores del 3,1 % en el peor de los casos. Para un *enlace débil* de 100 Mbps, la predicción de la fase *Reduce* tiene unos errores más significativos que subestiman la medición real en hasta un 29,1 %. Esto se debe a que los tiempos de esta fase (3,46 s en el peor de los casos según la Tabla 5.8), hacen que cualquier pequeña desviación en los cálculos se traduzca en errores muy grandes. Sin embargo, esto se ve compensado por el poco peso de la fase *Reduce* con respecto a la fase *Map*. El tiempo medio de la ejecución de cada instancia de la función *Map* es de 13,96 s frente a los 2,24 s de tiempo de ejecución para cada instancia de la función *Reduce*, además de ejecutar 150 tareas *Map* frente a 1 tarea *Reduce*. Por lo tanto, el error de la fase *Map* es el que más influye en el error de predicción medio, siendo en este caso de un 8,6 % en el peor de lo casos. Todo ello hace que el error total máximo sea menor del 5,8 % en el peor de los casos.

A nivel general, y al igual que ocurría con la aplicación IGrep, se puede conseguir un incremento de velocidad de hasta 3,84× añadiendo 12 MVs *off-premise* a la plataforma de 3 MVs *on-premise* con un *enlace débil* de 1 Gbps, o hasta 2,36× empleando 100 Mbps.

5.6.5 Predicción de la aplicación PageRank

PageRank [18] es un algoritmo de análisis de enlaces entre páginas web que asigna un peso numérico a cada elemento de un conjunto de documentos con hipervínculos, con el propósito de cuantificar su importancia relativa dentro del conjunto. La descripción de esta aplicación está más detalla en la sección 4.4. PageRank se encuentra implementado para Hadoop MapReduce como parte del *benchmark HiBench*, el cual se ha utilizado en este experimento para crear una serie de hipervínculos web siguiendo una distribución Zipf (o distribución zeta).

En este experimento se han generado 2,8 GB de datos de entrada que son procesados por 5 iteraciones complejas formadas por 2 trabajos cada una de ellas. En esta aplicación la carga computacional está repartida por los dos trabajos por iteración, en los que la primera fase es de tipo *Reduce*-intensiva y la segunda fase *Map*-intensiva.

Parámetro	**Fase 1**	**Fase 2**
M	98	48
R	48	48
t_{Mp}	10,36 s	18,62 s
t_{Mp}^{max}	13,45 s	23,44 s
t_{Rc}	5,29 s	6,33s
t_{Rc}^{max}	6,56 s	7,51 s
d_{Sh}	56,81 MB	116,4 MB
d_{Rd}	122 MB	4,48 MB

Tabla 5.11: Parámetros de caracterización de PageRank para la primera iteración.

Parámetro	**Fase 1**	**Fase 2**
M	144	48
R	48	48
t_{Mp}	9,93 s	19,12 s
t_{Mp}^{max}	12,12 s	23,44 s
t_{Rc}	5,51 s	6,4 s
t_{Rc}^{max}	6,48 s	10,17 s
d_{Sh}	56,7 MB	116 MB
d_{Rd}	121,6 MB	4,48 MB

Tabla 5.12: Parámetros de caracterización de PageRank para el resto de iteraciones.

Parámetros de caracterización

Para extraer los parámetros de caracterización de esta aplicación se ha ejecutado una instancia de ésta sobre las MVs *on-premise*, utilizando la misma entrada de datos y número de iteraciones indicadas anteriormente para obtener mayor precisión a la hora de caracterizar cada iteración.

Cabe destacar que la iteración inicial de PageRank debe leer los primeros datos de entrada, lo que produce una primera fase que es distinta del resto de iteraciones. Este comportamiento se ha caracterizado realizando una diferenciación entre la primera iteración y el resto de iteraciones, que están caracterizadas en las Tablas 5.11 y 5.12.

Resultados

PageRank es una aplicación escalable en la que, como podemos observar en la Tabla 5.13, se consigue un tiempo de finalización un 51 % más rápido cuando incrementamos en un factor 2× el número de MVs con respecto al caso inicial de 3 MVs *on-premise*.

Tiempos reales para una nube *on-premise*					
Arquitectura	**3 MVs**	**6 MVs**	**9 MVs**	**12 MVs**	**15 MVs**
	3145	**1511**	**1053**	**981**	**796**

Predicción utilizando el modelo ARIA				
Límite superior de la predicción	1763	1287	1059	924
Valor medio de la predicción	**1572**	**1090**	**859**	**724**
Límite inferior de la predicción	1379	892	660	523

Predicción utilizando nuestra metodología				
Nube híbrida con un *enlace débil* de **1 Gbps**				
Arquitectura	**3-On 3-Off**	**3-On 6-Off**	**3-On 9-Off**	**3-On 12-Off**
Tiempos reales	**1585**	**1130**	**1059**	**892**
Límite superior de la predicción	1831	1256	1351	1006
Valor medio de la predicción	**1670**	**1149**	**1101**	**886**
Límite inferior de la predicción	1509	1037	850	767
Nube híbrida con un *enlace débil* de **100 Mbps**				
Arquitectura	**3-On 3-Off**	**3-On 6-Off**	**3-On 9-Off**	**3-On 12-Off**
Tiempos reales	**4886**	**3764**	**3794**	**3330**
Límite superior de la predicción	5490	4006	4764	4064
Valor medio de la predicción	**5083**	**3806**	**3952**	**3583**
Límite inferior de la predicción	4675	3607	3140	3102

Tabla 5.13: Tiempos de finalización reales y de predicción para la aplicación PageRank (expresados en segundos). Para los tiempos de predicción se muestran tanto los límites inferiores y superiores, como el valor medio de predicción.

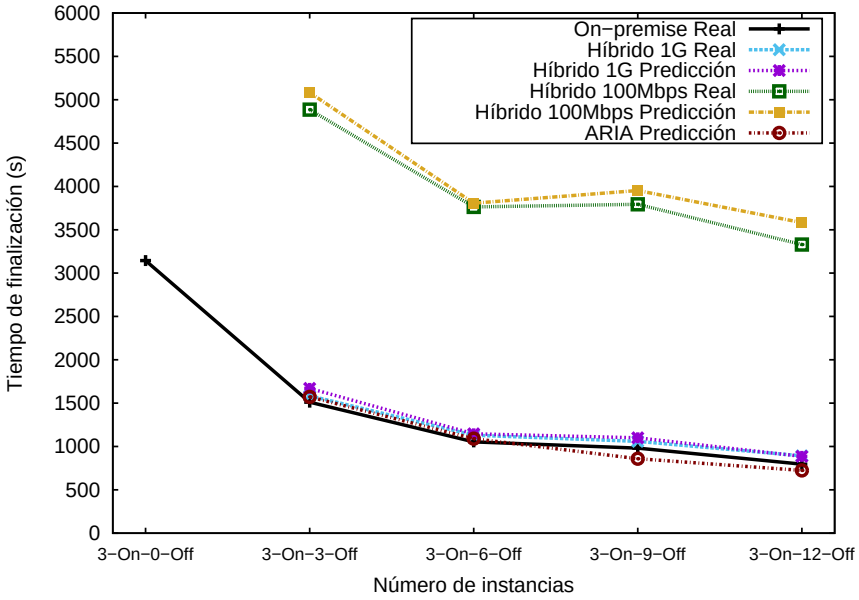


Figura 5.7: Tiempos de ejecución reales y estimados para la aplicación PageRank.

Arquitectura	**3-On 3-Off**	**3-On 6-Off**	**3-On 9-Off**	**3-On 12-Off**
Arquitectura híbrida con un *enlace débil* de 1 Gbps				
Porcentajes de error (%) del valor medio estimado con ARIA				
En la fase Map	8,6	2,5	-11,6	-8,6
En la fase Reduce	-5,8	-6,7	-22,8	-24,2
Error Total	**-0,8**	**-3,6**	**-18,9**	**-18,9**
Porcentajes de error (%) del valor medio estimado con nuestra metodología				
En la fase Map	5,2	-4	-9	-14
En la fase Reduce	5,4	4,7	11,2	6,7
Error Total	**5,4**	**1,5**	**3,9**	**-0,7**
Arquitectura híbrida con un *enlace débil* de 100Mbps				
Porcentajes de error (%) del valor medio estimado con ARIA				
En la fase Map	-1,2	-6,6	-16,1	-13,1
En la fase Reduce	-77,3	-79,8	-84,5	-85,3
Error Total	**-67,8**	**-71**	**-77,4**	**-78,3**
Porcentajes de error (%) del valor medio estimado con nuestra metodología				
En la fase Map	-0,1	-6,7	-7,4	-14,2
En la fase Reduce	4,6	2,2	5,6	10
Error Total	**4**	**1,1**	**4,2**	**7,6**

Tabla 5.14: PageRank: Precisión de la predicción media vs. el tiempo de finalización de un despliegue híbrido real detallado por fase.

En lo que se refiere a las predicciones del tiempo de finalización, analizando el límite superior e inferior obtenido para ARIA, podemos observar que, para el caso de 15 MVs (3-On 12-Off), se sobrestima el resultado en un 16 % para el límite superior y subestima en un 34 % el inferior con respecto al resultado real obtenido *on-premise*. Por otra parte, utilizando nuestra metodología de predicción, se obtienen unos resultados más ajustados con respecto a un escenario de nube híbrida de 15 MVs y un *enlace débil* de 1 Gbps, obteniendo una sobrestimación del 12 % en el límite superior y una subestimación del 14 % para el límite inferior. De la misma manera, para un *enlace débil* de 100 Mbps, nuestra predicción sobrestima el resultado en un 22 % en el límite superior y subestima en un 6 % el límite inferior con respecto al resultado real obtenido en un entorno híbrido con 100 Mbps de *enlace débil*.

En la Figura 5.7 se muestra la predicción media obtenida para cada enfoque. Como se observa, nuestra predicción se adapta perfectamente tanto al resultado real obtenido para 1 Gbps como para 100 Mbps, hecho que no ocurre con ARIA, que únicamente se adapta al escenario *on-premise* real e híbrido con 1 Gbps.

Analizando con más detalle las medias de error de cada una de las fases *Map* y *Reduce* que se presentan en la Tabla 5.14, podemos ver que la predicción con ARIA para un entorno híbrido con un *enlace débil* de 1 Gbps obtiene unos valores aceptables, con errores totales menores del 4 % cuando se utilizan 3 o 6 MVs en la parte *off-premise*. Este buen comportamiento desaparece a medida que se añaden más MVs *off-premise*, llegando a presentar errores del 18,9 %. Si nos centramos en ARIA con respecto a un entorno híbrido de 100 Mbps, podemos observar que este error se incrementa drásticamente. Esto es debido a su incapacidad de estimar la fase de *Reduce*, como se deduce de los errores cometidos para dicha fase, de hasta un 85 % por debajo de lo esperado. Esto perjudica la predicción del tiempo total, generando predicciones con errores totales de hasta un 78 % de desviación con respecto a los valores reales.

En cambio, podemos ver que los errores medios cometidos en nuestra predicción, tanto para la fase *Map* como *Reduce*, son todos menores del 15 %, tanto para 1 Gbps como 100 Mbps, lo que se traduce en un error total del 5,4 % en 1 Gbps para el peor de los casos experimentados, y en un 7,6 % en el peor de los casos para 100 Mbps. Para valorar la precisión de nuestra propuesta, hay que tener en cuenta que en los trabajos previos en predicción del tiempo de finalización de arquitecturas exclusivamente *on-premise*, se considera que los errores por debajo del 10 % son completamente aceptables.

En este caso solo se puede conseguir un incremento de velocidad añadiendo 12 MVs *off-premise* a la plataforma de 3 MVs *on-premise* con un *enlace débil* de 1 Gbps, en concreto de hasta un 3,53×. Para el caso de 100 Mbps no se obtendría incremento de velocidad alguno. Por tanto, la predicción presentada puede dar una idea al usuario de entre qué rangos de ancho de banda podría obtener un incremento de velocidad que le permita amortizar el gasto de utilizar una plataforma de nube híbrida.

5.6.6 Predicción de la aplicación Connected Components

La última aplicación utilizada para comprobar el funcionamiento de nuestra propuesta de metodología de predicción es Connected Components [82]. La detección de componentes conectados en gráficos es un problema bien conocido que surge en un gran número de aplicaciones, incluyendo minería de datos, análisis de redes sociales, análisis de imágenes y muchos otros problemas relacionados. Encontrar vértices conectados es un proceso inherentemente iterativo, de modo que una implementación basada en MapReduce da como resultado la ejecución repetida de un programa MapReduce, donde la salida de una iteración sirve como entrada para la siguiente iteración. La entrada del algoritmo de Connected Components es un grafo (V, E) con un conjunto de vértices V y un conjunto de aristas $E \subseteq V \times V$. El objetivo de este algoritmo es transformar el grafo de entrada en un conjunto de subgrafos de tipo estrella, asignando iterativamente cada vértice a su vecino más pequeño, usando un orden total de los vértices como el orden lexicográfico de las etiquetas de vértice.

Connected Components es una de las aplicaciones que componen el *benchmark BigDataBench*. A través de este *benchmark* se han generado 300 MB de información, que simula datos de extraídos de Facebook, la cual contiene un grafo en el que los vértices representan los usuarios y las aristas la relación de amistad entre ellos. Esta entrada de datos es procesada en este ejemplo por 9 iteraciones complejas, que están compuestas por 3 trabajos MapReduce.

Parámetros de caracterización

Como en el resto de aplicaciones, para obtener los parámetros de caracterización de esta aplicación se ha lanzado una instancia de Connected Components sobre las 3 MVs *on-premise* con la misma entrada de datos e iteraciones propuestas anteriormente (300 MB y 9 iteraciones complejas). La ejecución de esta aplicación genera iteraciones complejas compuestas por 3 trabajos MapReduce o fases de la iteración, donde finalmente ejecuta una última iteración con una última fase de salida. Por otra parte, de forma similar a lo que ocurre con PageRank, la primera iteración tiene un comportamiento ligeramente distinto al del resto de iteraciones. Así pues, nos encontramos con uno de los casos de aplicaciones MapReduce más complejos desde el punto de vista de la caracterización y de la predicción del tiempo de finalización. Dada la complejidad de esta aplicación, la caracterización de la misma se debe realizar en tres grupos.

- Una parte de inicialización compuesta por tres trabajos MapReduce.
- Una parte iterativa compuesta también por tres trabajos MapReduce.
- Una parte de finalización compuesta de un único trabajo MapReduce.

En las Tablas 5.15 y 5.16 se muestra la caracterización de las tres fases para la primera iteración y para el resto de iteraciones, respectivamente. Finalmente, en la Tabla 5.17 se muestra la caracterización de la última parte de la aplicación.

Cabe destacar que Connected Components combina tanto fases de tipo *Map*-intensivo como *Reduce*-intensivo. Por tanto, ambas fases influyen de forma similar en la predicción de los tiempos de finalización.

Parámetro	Fase 1	Fase 2	Fase 3
M	11	27	24
R	24	24	1
t_{Mp}	18,58 s	11,37 s	5,95 s
t_{Mp}^{max}	21,45 s	14,63 s	6,35 s
t_{Rc}	3,79 s	3,05 s	1,76 s
t_{Rc}^{max}	5,78 s	4,11 s	1,98 s
d_{Sh}	24,3 MB	120,25 MB	648 Bytes
d_{Rd}	31,8 MB	2,88 MB	18 Bytes

Tabla 5.15: Parámetros de caracterización de Connected Components para la primera iteración.

Parámetro	Fase 1	Fase 2	Fase 3
M	33	27	24
R	24	24	1
t_{Mp}	6,08 s	11,37 s	5,91 s
t_{Mp}^{max}	16,49 s	15,36 s	6,91 s
t_{Rc}	3,5 s	3,16 s	1,85 s
t_{Rc}^{max}	6,14 s	5,07 s	2,5 s
d_{Sh}	23,6 MB	14,45 MB	568 Bytes
d_{Rd}	23,4 MB	2,25 MB	16 Bytes

Tabla 5.16: Parámetros de caracterización de Connected Components para el resto de iteraciones.

Parámetro	Fase 4
M	24
R	24
t_{Mp}	6,06 s
t_{Mp}^{max}	7,72 s
t_{Rc}	2,03 s
t_{Rc}^{max}	3,56 s
d_{Sh}	126,7 KB
d_{Rd}	124,7 KB

Tabla 5.17: Parámetros de caracterización de Connected Components para la última iteración.

Resultados

En la Tabla 5.18 se muestran los tiempos de finalización tanto reales como las predicciones realizadas utilizando ARIA y empleando nuestra metodología de predicción. Connected Components es también una aplicación con un cierto grado de escalabilidad, consiguiendo una mejora del tiempo de ejecución de un 42 % si se añaden 3 MVs *on-premise* más al entorno inicial de 3 MVs *on-premise*.

Analizando los límites inferiores y superiores de las predicciones, podemos ver que ARIA subestima su predicción en hasta un 67 % con respecto al resultado *on-premise* real en su límite inferior para 15 MVs. Si lo comparamos con el resultado real obtenido para un entorno híbrido de 100 Mbps, la predicción se realiza con una subestimación del 85 %. Con nuestra metodología de predicción, para nubes híbridas que utilizan un *enlace débil* de 1 Gbps subestimamos el resultado real en un 30 % en el peor de los casos y en un 33 % en el peor de los casos si el *enlace débil* es de 100 Mbps. Estos valores demuestran que nuestra aproximación está mucho más cerca de los resultados reales.

Tiempos reales para una nube *on-premise*					
Arquitectura	**3 MVs**	**6 MVs**	**9 MVs**	**12 MVs**	**15 MVs**
	1621	**928**	**757**	**686**	**647**

Predicción utilizando el modelo ARIA				
Límite superior de la predicción	1178	972	883	810
Valor medio de la predicción	**885**	**674**	**584**	**510**
Límite inferior de la predicción	592	377	284	209

Predicción utilizando nuestra metodología				
Nube híbrida con un *enlace débil* de **1 Gbps**				
Arquitectura	**3-On 3-Off**	**3-On 6-Off**	**3-On 9-Off**	**3-On 12-Off**
Tiempos reales	**947**	**758**	**714**	**650**
Límite superior de la predicción	1157	848	896	787
Valor medio de la predicción	**911**	**684**	**695**	**661**
Límite inferior de la predicción	665	520	494	536
Nube híbrida con un *enlace débil* de **100 Mbps**				
Arquitectura	**3-On 3-Off**	**3-On 6-Off**	**3-On 9-Off**	**3-On 12-Off**
Tiempos reales	**1833**	**1380**	**1451**	**1419**
Límite superior de la predicción	2131	1585	1878	1450
Valor medio de la predicción	**1743**	**1308**	**1422**	**1253**
Límite inferior de la predicción	1353	1031	967	1056

Tabla 5.18: Tiempos de finalización reales y de predicción para la aplicación Connected Components (expresados en segundos). Para los tiempos de predicción se muestran tanto los límites inferiores y superiores, como el valor medio de predicción.

Viendo la gráfica de la Figura 5.8, donde se muestran las medias de las predicciones con respecto a su resultado real, podemos observar que nuestra predicción sigue la misma tendencia que los resultados reales, tanto para 1 Gbps como para 100 Mbps. Por el contrario, la aproximación de

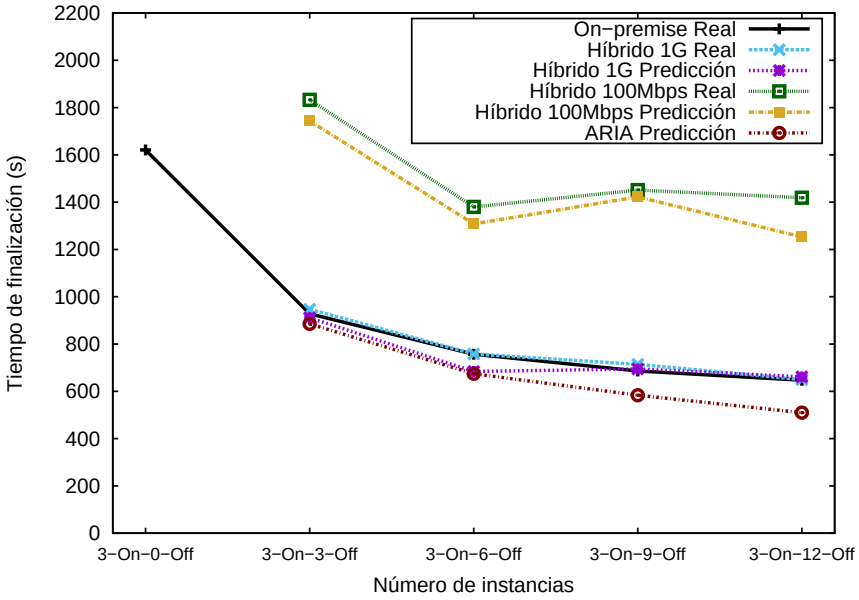


Figura 5.8: Tiempos de ejecución reales y estimados para la aplicación Connected Components.

Arquitectura	**3-On 3-Off**	**3-On 6-Off**	**3-On 9-Off**	**3-On 12-Off**
Arquitectura híbrida con un *enlace débil* de 1 Gbps				
Porcentajes de error (%) del valor medio estimado con ARIA				
En la fase Map	-4,5	-8	-15,4	-21,6
En la fase Reduce	-8,3	-13,6	-14,4	-21,9
Error Total	**-6,5**	**-11**	**-18,3**	**-21,6**
Porcentajes de error (%) del valor medio estimado con nuestra metodología				
En la fase Map	-2,8	-13,7	-3,8	5,7
En la fase Reduce	-4,5	-5,2	7	-3,5
Error Total	**-3,8**	**-9,8**	**-2,7**	**1,7**
Arquitectura híbrida con un *enlace débil* de 100Mbps				
Porcentajes de error (%) del valor medio estimado con ARIA				
En la fase Map	-12,9	-15,9	-29	-35,1
En la fase Reduce	-66,8	-66,5	-72,9	-76,8
Error Total	**-51,7**	**-51,1**	**-59,8**	**-64,1**
Porcentajes de error (%) del valor medio estimado con nuestra metodología				
En la fase Map	-6	-14,2	-11,8	-6,8
En la fase Reduce	-4,5	-1,3	2,2	-13,8
Error Total	**-4,9**	**-5,2**	**-2**	**-11,7**

Tabla 5.19: Connected Components: Precisión de la predicción media vs. el tiempo de finalización de un despliegue híbrido real detallado por fase.

ARIA se queda muy por debajo del resultado real, produciendo un error de hasta un 64 % para el caso de 100 Mbps.

Si analizamos los porcentajes de error de las predicciones desglosadas por fases *Map* y *Reduce* que se muestran en la Tabla 5.19, podemos observar que utilizando ARIA se comenten errores de hasta el 21 %, tanto en la fase *Map* como *Reduce*, en todos los escenarios de nube híbrida con un *enlace débil* de 1 Gbps, obteniendo un error total de un 21,6 % en el peor de los casos. Esta situación todavía es peor cuando el *enlace débil* es de 100 Mbps, incrementándose entonces el error hasta un 35 % para la fase *Map* y hasta un 76 % para la fase *Reduce*, produciendo un error total subestimado del 64 % para 15 MVs (3-On 12-Off). Esto hace descartar completamente la predicción de ARIA cuando nos enfrentamos a la ejecución de aplicaciones MapReduce sobre nubes híbridas.

Si nos centramos en los resultados de nuestra metodología de predicción, podemos observar que para una arquitectura de nube híbrida con un *enlace débil* de 1 Gbps, el error máximo que se produce en la fase *Map* es del 13 %, y del 7 % para la fase *Reduce*, dando lugar a un error medio total menor del 10 % en el peor de los casos. Si el *enlace débil* de la arquitectura híbrida es de 100 Mbps, obtenemos un error máximo (por subestimación) del 14 % para la fase *Map* y del 13 % para la fase *Reduce*, produciendo un error total menor del 12 % en el peor de los casos. Así pues, ante una de las estructuras más complejas de aplicación MapReduce, se observa un comportamiento adecuado de nuestra metodología de predicción, con errores de hasta el 12 % en el peor de los casos; sobre todo si la comparamos con el estado del arte actual, con errores de hasta el 76 % utilizando ARIA.

A nivel general, en este caso también se observa que se puede obtener un incremento de velocidad de hasta 2,49× al añadir 12 MVs *off-premise* a la plataforma original de 3 MVs *on-premise* para un *enlace débil* de 1 Gbps. Para el caso de 100 Mbps se puede obtener un incremento de velocidad próximo a 1,16×, pero con la peculiaridad de que no tendría sentido aumentar el número de MVs *off-premise* más allá de 6, puesto que no se conseguiría mejora alguna.

5.7 Conclusiones

En este capítulo se ha abordado el problema de cómo estimar el tiempo de finalización de aplicaciones iterativas de MapReduce en escenarios de nubes híbridas con despliegues de *cloud bursting*, donde las MVs *on-premise* y *off-premise* se comunican a través de un *enlace débil*. Dichas estimaciones de tiempo de finalización son una herramienta fundamental para ayudar a decidir si la aceleración justifica el coste de pagar por MVs *off-premise* adicionales.

Para abordar este problema, se ha propuesto una metodología de predicción de rendimiento que combina el modelado analítico junto con un test sintético para estimar el tiempo de finalización específicamente para una configuración de nube híbrida, donde el *enlace débil* tiene un impacto decisivo prácticamente en todas las fases de MapReduce.

Hemos demostrado los beneficios de nuestra propuesta desde múltiples ángulos, utilizando una mezcla de aplicaciones reales iterativas *Map*-intensivas, *Reduce*-intensivas y balanceadas, que cubren un amplio espectro de casos de uso. Concretamente, hemos demostrado las siguientes ventajas:

1. La predicción del límite superior e inferior de nuestro enfoque frente a los resultados reales obtenidos para una configuración de nube híbrida es mucho más precisa que las predicciones que realizan modelos de predicción como ARIA.

2. La predicción media de nuestro enfoque está siempre dentro de un intervalo de error de entre el 1 % y el 10 %, independientemente del escenario y el ancho de banda del *enlace débil*, siendo un orden de magnitud más preciso que modelos de predicción del estado del arte.

3. Nuestro enfoque presenta un comportamiento consistente que predice con precisión tanto la fase *Map* como *Reduce* de una aplicación MapReduce, lo que se traduce en una estimación global de mucha precisión.

La metodología de predicción propuesta requiere una calibración inicial del sistema híbrido sobre el que se realizará la predicción, pero ésta servirá para el resto de predicciones de distintas aplicaciones que se realicen en dicha configuración según convenga, ya que las variaciones de rendimiento que se puedan producir en un proveedor de la nube son insignificantes comparadas con la precisión que obtenemos en nuestra predicción. De igual forma, inicialmente se deben obtener los parámetros característicos de cada aplicación a través de su ejecución a pequeña escala en una infraestructura *on-premise*. Tantos los parámetros de calibración del sistema como de caracterización de cada aplicación pueden ser almacenados para posteriores predicciones en entornos y configuraciones distintas.

CAPÍTULO 6

Conclusions

This chapter summarizes the main contributions of this thesis and lists the corresponding publications where they were originally introduced. It concludes with a series of open research directions derived from this work that address new challenges or aspects that were insufficiently developed.

6.1 Achievements

The main objective of this thesis has been the design and validation of a methodology to predict the completion time and cost of iterative MapReduce applications in hybrid cloud bursting deployments.

The work presented in this thesis has led to the following achievements, which fulfill the main objective:

- The development of an environment for extracting statistical data and traces from the execution of a MapReduce application on a hybrid cloud. This environment is composed of a collection of tools that facilitate the extraction of the relevant data that allow the visualization of the task scheduling, the application characterization, and the system calibration, as well as other statistical parameters related to the traffic between the on-premise and off-premise clouds.

- A study of the main factors that influence the performance of iterative MapReduce applications in a hybrid cloud bursting deployment, which facilitates a better understanding of the problem.

- The design, implementation and evaluation of several off-premise data migration strategies for the storage layer, that facilitate the MapReduce runtime to leverage data locality efficiently in a hybrid cloud bursting deployment, which enables performance improvement.

- The design, implementation and evaluation of a MapReduce task scheduling strategy specifically designed to complement the off-premise data migration strategies in a hybrid cloud bursting deployment, which enables performance improvement.
- A methodology to predict the completion time of running iterative MapReduce applications in a hybrid cloud bursting deployment, which enables the understanding of how much performance improvement can be expected for a certain amount of off-premise resources that complement the on-premise resources.
- A methodology to analyze the cost of running iterative MapReduce applications in a hybrid cloud deployment using the hybrid-specific data migration and task scheduling strategies, which enables reasoning about the necessary cost to obtain better performance.

6.1.1 Statistical extraction environment for Hadoop MapReduce applications

To achieve a level of detail that facilitates a fine-grain analysis of each of the stages of a MapReduce execution and the extraction of key metrics (such as how much network traffic is exchanged between the on-premise cloud and the off-premise cloud), it is necessary to create a set of tools that are simple and easily manageable by the user.

This contribution is described in Chapter 3, which presents a statistical extraction environment for Hadoop MapReduce, called H-Stat. It is composed of a collection of tools that enable the extraction of essential information to perform a complete analysis of the behavior of a MapReduce job on a given environment, with specific focus on the information related to a hybrid cloud bursting scenario. Specifically, this environment allows the extraction of traces containing detailed information about the MapReduce task scheduling performed at each node, in addition to information regarding the resource utilization of each node: CPU, disk and network I/O, memory. These traces can be graphically visualized through the Paraver application and help to identify the challenges of running iterative MapReduce applications in a hybrid cloud bursting scenario. Likewise, the statistical information extracted has made it easier to obtain essential parameters that describe both the virtualized environment and the application behavior. Both aspects are essential in obtaining a formula to predict the application completion time for MapReduce, as described in Chapter 5. In addition, the extraction of traffic information produced between two clouds enables a detailed analysis of the economic cost of such traffic in a public cloud, as presented in Chapter 4.

6.1.2 Data locality strategies for Hadoop MapReduce

Chapter 4 provides an extensive analysis of the implications of running a MapReduce application on a hybrid cloud bursting scenario. To do this, it delves into one of the main factors that affects the performance of a MapReduce application, such as the ability to exploit data locality in order to schedule the computation close to the data. In this context, the temporary off-premise resources provisioned from a public cloud provider are linked with the on-premise resources through a "weak link", that is an order of magnitude slower than the network links between the nodes within the same cloud. Thus, unnecessary or redundant traffic between the two clouds must be avoided, since otherwise it damages the performance of the application. However, in order to maximize a cloud bursting deployment, where on-premise resources are complemented by off-premise resources with the aim of improving the performance of a particular application, it is necessary to study

and analyze different strategies to exploit data locality, as well as scheduling policies that allow to obtain the maximum performance in these type of deployments. This chapter addresses a specific MapReduce application type, such as iterative applications, whose main feature is a refinement between iterations, usually through the re-use of input data. This property makes these type of applications very suitable for a cloud bursting deployment, in which once the data is sent, it can be reused again and again, thus masking the cost of sending the data through a determined number of iterations.

Among the contributions made in this chapter are the proposal and study of two data migration strategies, through a new proposed approach of the Hadoop, *Rack Awareness* feature. This feature has been exploited and configured to replicate data from on-premise nodes to off-premise nodes in a transparent way to the user. These two strategies allow either sending a replica of the input data and then launching the application (*Blocking Rebalance*), or sending the replica asynchronously (*Plain Asynchronous Rebalance*). Both strategies use the default Hadoop scheduler, which schedules each of the *Map* or *Reduce* tasks in any available slots, prioritizing the slots where their input data is closest. However, after an extensive analysis of the scheduler behavior in a cloud bursting deployment, it has been concluded that it unconsciously schedules MapReduce tasks to slots of off-premise nodes where the data has not yet been migrated, leading to unnecessary memory transfers among remote nodes, thus depleting its performance.

The main contribution in this chapter has been the introduction of a new scheduling policy aware of this new heterogeneous network environment, which avoids the task scheduling in remote nodes that do not contain a copy of their input data, at least within their own network. This new policy leads us to the creation of a new strategy, *Rack-Local Asynchronous Rebalance*, based on the *Plain Asynchronous Rebalance* strategy, but with the new scheduling policy. All these strategies have been evaluated and compared with a base strategy (*No HDFS Off-Premise*), in which no input data is sent, through the experimentation and evaluation of two iterative MapReduce applications (K-Means and PageRank) in the *Kinton* platform. The results obtained show that our strategy obtains the best results in all the cases, being the MapReduce applications with more weight in the *Map* phase, or *Map*-intensive applications the ones that better adapt to a cloud bursting deployment, improving its performance in each iteration until it reaches its maximum performance once all the input data has been migrated.

In addition, at the end of the chapter we also contribute with a methodology for analyzing the economic cost of a cloud bursting deployment for each strategy presented, taking into account both the time consumed by each VM and the amount of traffic produced between the two clouds. We can also conclude that our strategy provides the best results for both cost and performance of the applications, being the *Map*-intensive applications the best cost/performance ratio can be obtained.

6.1.3 Methodology for predicting the completion time of iterative MapReduce applications

In Chapter 5, the main contribution has been a methodology to predict the completion time of an iterative MapReduce application in a hybrid cloud bursting scenario. Since extending on-premise resources with off-premise resources of a public cloud for a certain time is costly for the user, it is advisable to know in advance if this new hybrid cloud configuration will obtain the expected performance, thus justifying the cost involved in such deployment. However, such performance

depends not only on the number of extended resources, but also on the bandwidth of the contracted weak link. Therefore, scenarios can be given in which an application fails to improve its performance while expanding its resources with a large number of off-premise resources. In this chapter, these problems are analyzed through a methodology that is able to predict the completion time of a given iterative MapReduce application in a particular cloud bursting deployment. In this way, a user can decide whether he needs to increase or decrease the number of off-premise resources to obtain the expected performance, as well as determine the economic cost of such a change and if it really justifies the price he will pay for it.

This new prediction methodology consists of three main steps: the first one calibrates the entire set of on-premise and off-premise resources that form the hybrid cloud to extract the main system parameters, that influence the performance of a MapReduce application. Secondly, the desired application is characterized through its small scale execution (both number of VMs and iterations) in the on-premise VMs to determine its invariant parameters against a scale change. Third, a mathematical expression is applied that predicts the time for each of the MapReduce phases, and obtains an upper and lower prediction bound. In addition, this mathematical expression contemplates both the initial migration of data and the prediction of complex iterations composed of different MapReduce jobs. This allows to accurately predict any type of iterative application. It should be noted that only one execution of the initial two steps of this methodology is required, as the information obtained can be reused in future predictions.

This methodology has been evaluated and validated by four real-life iterative MapReduce applications, extracted from standardized benchmarks like *HiBench* or *BigDataBench*, which cover the full range of possibilities for MapReduce phase balancing. To do this, two OpenStack clouds have been created through the *Kinton* cluster to form a hybrid cloud platform, with a software-controlled weak link bandwidth. Starting from on-premise 3 VMs, and extending them with off-premise resources from 3 to 12 VMs, our methodology has been applied for both a 1 Gbps weak link and a 100 Mbps weak link scenario. All prediction results obtained have been compared in detail (disaggregating each MapReduce phase) with their respective actual results for each hybrid cloud scenario. In addition, to give more validity to our prediction, the accuracy obtained has been compared to other similar state-of-the-art prediction methodologies.

We can conclude that our methodology obtains an average prediction very accurately, with an error range between 1% and 10%, regardless of the scenario, application and weak link bandwidth. This does not happen with other methodologies which yield an error of almost 80% with respect to the actual result obtained for a scenario with a 100 Mbps weak link. In addition, analyzing in detail the accuracy obtained in each of the MapReduce phases, we can conclude that our approach is capable of accurately predicting these phases, which corroborates our good results. Therefore, we have developed a methodology that is able to predict the completion time of any iterative MapReduce application in a hybrid cloud bursting scenario, so that we can anticipate both the economic cost that will lead to its execution as well as its performance. In this way, a user can decide whether to increase/decrease the off-premise resources or hire a better weak link bandwidth to achieve the expected performance improvement with the best performance-price ratio, without having to run their applications for it, thus saving both economic and time costs.

6.2 Main publications

The scientific contributions presented in this thesis have been validated through a series of peer reviewed national and international publications in venues appropriate for the scope of this thesis, as listed below:

Journal [30]

Clemente-Castelló, F.J., Nicolae, B., Mayo, R., Fernández, J.C. Performance Model of MapReduce Iterative Applications for Hybrid Cloud Bursting. In *IEEE Transactions on Parallel and Distributed Systems* (2017), (Under Review).

> Hybrid cloud bursting (i.e., leasing temporary off-premise cloud resources to boost the overall capacity during peak utilization) can be a cost-effective way to deal with the increasing complexity of big data analytics, especially for iterative applications. However, the low throughput, high latency network link between the on-premise and off-premise resources ("weak link") makes it difficult to maintain scalability. While there are several data locality techniques dedicated for big data bursting on hybrid clouds, their effectiveness is difficult to estimate in advance. On the other hand, such estimations are critical for users, because they aid in the decision of whether the extra pay-as-you-go cost incurred by using the off-premise resources justifies the runtime speed-up. To this end, the current paper contributes with a performance model and methodology to estimate the runtime of iterative MapReduce applications in a hybrid cloud bursting scenario. A key idea of the proposal is to focus on the overhead incurred by the weak link at fine granularity, both for the map and reduce phase. This enables high estimation accuracy, as demonstrated by extensive experiments at scale using a mix of real-life iterative MapReduce applications from standard big data benchmarking suites that cover a broad spectrum of data patterns. Not only are the produced estimations accurate in absolute terms compared with the actual experimental results, but they are also up to an order of magnitude more accurate than applying state-of-art estimation approaches originally designed for single-site MapReduce deployments.

Conference Proceedings [32]

Clemente-Castelló, F.J., Nicolae, B., Mustafa, M.R., Mayo, R., Fernández, J.C. Evaluation of Data Locality Strategies for Hybrid Cloud Bursting of Iterative MapReduce. In *CCGrid'17: 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing* (2017)

> Hybrid cloud bursting (i.e., leasing temporary off-premise cloud resources to boost the overall capacity during peak utilization) is a popular and cost-effective way to deal with the increasing complexity of big data analytics. It is particularly promising for iterative MapReduce applications that reuse massive amounts of input data at each iteration, which compensates for the high overhead and cost of concurrent data transfers from the on-premise to the off-premise VMs over a weak inter-site link that is of limited capacity. In this paper we study how to combine various MapReduce data locality techniques designed for hybrid cloud bursting in order to achieve scalability for iterative MapReduce applications in a cost-effective fashion. This is a non-trivial problem due to the complex interaction between the data movements over the weak link and the scheduling of computational tasks that have to adapt to the shifting data distribution. We show that using the right combination of techniques, iterative MapReduce applications can scale

well in a hybrid cloud bursting scenario and come even close to the scalability observed in single sites.

Conference Proceedings [25]

Clemente-Castelló, F.J., Mayo, R., Fernández, J.C. Cost Model and Analysis of Iterative MapReduce Applications for Hybrid Cloud Bursting. In *SCRAMBL'17: 3rd International Workshop on Scalable Computing For Real-Time Big Data Applications* (2017)

A popular and cost-effective way to deal with the increasing complexity of big data analytics is hybrid cloud bursting that leases temporary off-premise cloud resources to boost the overall capacity during peak utilization. The main challenge of hybrid cloud bursting is that the network link between the on-premise and the off-premise computational resources often exhibit high latency and low throughput ("weak link") compared to the links within the same data-center. This paper introduces a cost model that is specifically designed for iterative MapReduce applications running in a hybrid cloud bursting scenario, which are a popular class of large-scale data-intensive applications that provides near real-time responsiveness. Using this cost model, users can discover trends that can be leveraged to reason about how to balance performance, accuracy and cost such that it optimizes their requirements. We illustrated this approach through a cost analysis that focuses on two real-life iterative MapReduce applications using extensive horizontal scalability experiments that involve multiple hybrid cloud bursting strategies.

Conference Proceedings [31]

Clemente-Castelló, F.J., Nicolae, B., Mustafa, M.R., Mayo, R., Fernández, J.C. On exploiting data locality for iterative mapreduce applications in hybrid clouds. In *BDCAT '16: 3rd IEEE/ACM International Conference on Big Data Computing, Applications and Technologies* (2016), pp. 118–122.

Hybrid cloud bursting (i.e., leasing temporary off-premise cloud resources to boost the capacity during peak utilization), has made significant impact especially for big data analytics, where the explosion of data sizes and increasingly complex computations frequently leads to insufficient local data center capacity. Cloud bursting however introduces a major challenge to runtime systems due to the limited throughput and high latency of data transfers between on-premise and off-premise resources (weak link). This issue and how to address it is not well understood. We contribute with a comprehensive study on what challenges arise in this context, what potential strategies can be applied to address them and what best practices can be leveraged in real-life. Specifically, we focus our study on iterative MapReduce applications, which are a class of large-scale data intensive applications particularly popular on hybrid clouds. In this context, we study how data locality can be leveraged over the weak link both from the storage layer perspective (when and how to move it off-premise) and from the scheduling perspective (when to compute off-premise). We conclude with a brief discussion on how to set up an experimental framework suitable to study the effectiveness of our proposal in future work.

Conference Proceedings [29]

Clemente-Castelló, F.J., Nicolae, B., Katrinis, K., Mustafa, M.R., Mayo, R., Fernández, J.C. Enabling Big Data Analytics in the Hybrid Cloud Using Iterative MapReduce. In *UCC '15: 8th IEEE/ACM International Conference on Utility and Cloud Computing* (2015), pp. 290–299.

> The cloud computing model has seen tremendous commercial success through its materialization via two prominent models to date, namely public and private cloud. Recently, a third model combining the former two service models as on-/off-premise resources has been receiving significant market traction: hybrid cloud. While state of art techniques that address workload performance prediction and efficient workload execution over hybrid cloud setups exist, how to address data-intensive workloads - including Big Data Analytics - in similar environments is nascent. This paper addresses this gap by taking on the challenge of bursting over hybrid clouds for the benefit of accelerating iterative MapReduce applications. We first specify the challenges associated with data locality and data movement in such setups. Subsequently, we propose a novel technique to address the locality issue, without requiring changes to the MapReduce framework or the underlying storage layer. In addition, we contribute with a performance prediction methodology that combines modeling with micro-benchmarks to estimate completion time for iterative MapReduce applications, which enables users to estimate cost-to-solution before committing extra resources from public clouds. We show through experimentation in a dual-Openstack hybrid cloud setup that our solutions manage to bring substantial improvement at predictable cost-control for two real-life iterative MapReduce applications: large-scale machine learning and text analysis.

Actas Congreso [26]

Clemente-Castelló, F.J., Mayo, R, Fernández, J.C., Quintana-Ortí, E.S., Nicolae, B., Katrinis, K. Estudio Preliminar de una Nube Híbrida: Análisis de Rendimiento con MapReduce. In *JP'15: XXVI Jornadas de Paralelismo* (2012), pp. 274–281.

> En este artículo se estudia la utilización de una nube híbrida y los problemas que conlleva. Para ello, se ha realizado un análisis desde la perspectiva del procesamiento de grandes volúmenes de datos utilizando el modelo de programación MapReduce mediante su implementación en Hadoop. A diferencia de otros trabajos, en éste se analiza el rendimiento en la ejecución de un solo trabajo Hadoop utilizando todos los recursos de cómputo híbrido conjuntamente. Los primeros resultados de este trabajo en progreso nos indican las dificultades que se pueden producir en este tipo de entorno y el tipo de aplicaciones y estrategias a emplear en cada caso.

6.3 Other publications

Before focusing on the line of research presented in this thesis, an initial thematic was developed based on the research of a layer underlying the cloud computing, as these are virtual machines. Specifically, this research focused on the evaluation of live migration of virtual machines between two physical nodes, in terms of performance and energy. In addition, we contributed a new adaptive

method for the QEMU/KVM hypervisor that allows to migrate specific scenarios of VMs that can not be migrated. This is the case of applications with a high degree of memory concurrency running in VMs. It was achieved by adapting and minimizing the VM downtime. This corresponds to the last step of the migration, in which all memory and CPU status are transferred.

These investigations were validated through two international conferences, and two previous works were contributed at national conferences.

Conference Proceedings [23]

Clemente-Castelló, F.J., Cervera, S., Mayo, R., Quintana-Ortí, E.S. Evaluating the Impact of Virtualization on Performance and Power Dissipation. In *CLOSER '14: 4th International Conference on Cloud Computing and Services Science* (2014), pp. 513–518.

> In this paper we assess the impact of virtualization in both performance-oriented environments, like high performance computing facilities, and throughput-oriented systems, like data processing centers, e.g., for web search and data serving. In particular, our work-in-progress analyzes the power consumption required to dynamically migrate virtual machines at runtime, a technique that is crucial to consolidate underutilized servers, reducing energy costs while maintaining service level agreements. Preliminary experimental results are reported for two different applications, using the KVM virtualization solution for Linux, on an Intel Xeonbased cluster.

Conference Proceedings [24]

Clemente-Castelló, F.J., Fernández, J.C., Mayo, R., Quintana-Ortí, E.S. Adaptive Downtime for Live Migration of Virtual Machines. In *UCC '14: 7th IEEE/ACM International Conference on Utility and Cloud Computing* (2014), pp. 457–464.

> Live migration of virtual machines is a pivotal mechanism to offer consolidation services in virtualized and cloud computing environments. In general, the standard live migration process favors short downtime periods, to avoid affecting on-line services provided by virtual machines. In particular, live migration can only be applied in scenarios where the dirty page rate is low compared to the communication network bandwidth. Furthermore, while setting a short downtime is convenient for on-line services, this may not be necessary, e.g., for long time-consuming high performance applications as, in this sort of computations, a relative large of downtime period, (say, in the order of minutes,) due to a live migration, only introduces a small overhead on the total execution time. In this paper we enhance the live migration process in a variety of scenarios where the standard live migration configuration does not permit live migration. For these situations, we propose, analyse, and validate a modification of the KVM hypervisor that accommodates live migration with short downtime, independently of the type of application and services in execution on the virtual machine.

Actcas Congreso [27]

Clemente-Castelló, F.J., Mayo, R., Quintana-Ortí, E.S. Estudio y Requisitos de la Simulación de Sistemas de Cloud Computing. In *JP'13: XXIV Jornadas de Paralelismo* (2013), pp. 199–204.

> El cloud computing o computación en la nube es una tecnología reciente que permite que los recursos TI (Tecnologías de la Información) y las aplicaciones, sean ofertadas a los usuarios finales como servicios, bajo un modelo de pago por uso. Cada vez más

empresas están utilizando esta tecnología para virtualizar sus aplicaciones y utilizar únicamente los recursos TI necesarios para su ejecución en cada momento, ahorrando costes de adquisición, mantenimiento y energía. Normalmente estas aplicaciones tienen un complejo despliegue y distintas configuraciones y requisitos. Evaluar y predecir el rendimiento, la asignación de recursos, las cargas de trabajo y los costes que éstas conllevan en un entorno de cloud computing, utilizando distintas configuraciones y variaciones del sistema, son tareas complejas y costosas de lograr. Para llevar a cabo estas tareas, sería conveniente utilizar un simulador de cloud computing que permitiera evaluar y predecir el comportamiento de estas aplicaciones en este entorno. En este artículo se presentan los requisitos básicos que debería tener un simulador de cloud computing, y se hace una comparativa entre varios simuladores de cloud computing existentes en la actualidad.

Actcas Congreso [28]

Clemente-Castelló, F.J., Mayo, R., Cervera S., Quintana-Ortí, E.S. Efecto de la Virtualización sobre el Rendimiento, la Productividad y el Consumo. In *JP'12: XXIII Jornadas de Paralelismo* (2012), pp. 181–126.

En este artículo se presenta un análisis sobre la utilización de la virtualización en entornos donde se busca rendimiento, como sería el caso de los centros de datos destinados a computación de altas prestaciones; o como cuando el rendimiento se buscan en el ámbito de la productividad, como es el caso de los centros de datos destinados a aplicaciones de gestión o de servicios de Internet. En este trabajo se realiza un análisis desde el punto de vista del consumo energético y de la influencia de las migraciones en vivo de las máquinas virtuales. Esta última técnica es básica para adaptar la consolidación de las máquinas virtuales sobre servidores físicos y reducir al máximo el consumo energético, atendiendo siempre a la calidad de servicio requerida por los usuarios.

6.3.1 Collaborative publications

While conducting research for this thesis I collaborated with other researchers, in which I contributed my knowledge on the deployment and use of VMs in public clouds (e.g. Amazon) and private clouds (e.g. OpenStack).

Conference Proceedings [23]

Iserte, S., Clemente-Castelló, F.J., Castelló, A., Mayo, R., Quintana-Ortí, E.S. Enabling GPU Virtualization in Cloud Environments In *CLOSER '16: 6th International Conference on Cloud Computing and Services Science* (2016), pp. 249–256.

The use of accelerators, such as graphics processing units (GPUs), to reduce the execution time of computeintensive applications has become popular during the past few years. These devices increment the computational power of a node thanks to their parallel architecture. This trend has led cloud service providers as Amazon or middlewares such as OpenStack to add virtual machines (VMs) including GPUs to their facilities instances. To fulfill these needs, the guest hosts must be equipped with GPUs which, unfortunately, will be barely utilized if a non GPU-enabled VM is running in the host. The solution presented in this work is based on GPU virtualization and shareability in order to reach an equilibrium between service supply and the applications demand of

accelerators. Concretely, we propose to decouple real GPUs from the nodes by using the virtualization technology rCUDA. With this software configuration, GPUs can be accessed from any VM avoiding the need of placing a physical GPUs in each guest host. Moreover, we study the viability of this approach using a public cloud service configuration, and we develop a module for OpenStack in order to add support for the virtualized devices and the logic to manage them. The results demonstrate this is a viable configuration which adds flexibility to current and well-known cloud solutions.

6.4 Open research lines

Cloud computing and specifically hybrid clouds have opened up a new paradigm for distributed computing that is allowing industry to improve its computing solutions efficiently. This thesis has contributed in this aspect, allowing applications such as the ones used for the analysis of Big Data to be efficiently executed in such an environment, improving their task scheduling and predicting their performance to avoid unnecessary costs. However, being a very recent technology, there are many open lines in this research that can improve and expand the possibilities offered in this thesis in terms of performance and estimation:

- Currently, the process of data migration and task scheduling is decoupled, that is, it occurs independently. It would be interesting to explore how to prioritize the order in which the data is migrated based on the decisions made by the task scheduler, so that performance is improved by sending the data that the scheduler first needs and thus better exploiting the off-premise resources.

- In this research the task scheduling of *Map* phase in hybrid cloud environments has been improved in order to avoid unnecessary traffic through the weak link. However, the *Reduce* phase requires the research of more complex scheduling techniques to reduce the traffic it produces in the weak link, and thus further optimize the execution of MapReduce applications over a hybrid cloud environment.

- The prediction methodology proposed in this thesis predicts the behavior of an application in a hybrid cloud environment, but with homogeneous computing resources. Therefore, this methodology could be expanded by incorporating the possibility of predicting behavior with heterogeneous computational resources.

- Another open research line may be focused on elasticity. Specifically, in this thesis we assume that we have a fixed set of off-premise VMs during the execution of an application. However, it may be very useful to be able to have an elastic behavior where the off-premise VMs configuration could change during the execution of the application (adding/decreasing VMs in each iteration) to adapt to changing objectives (e.g. results are needed faster than initially anticipated).

- Apache Hadoop is one of the most widely used environments for Big Data analysis; however, environments like Apache Spark [86] are being used widely in different sectors. Therefore, a research line would be the adaptation of our prediction methodology to the new Spark environment.

6.5 Conclusiones generales

El principal objetivo de esta tesis ha sido el diseño y validación de una metodología de predicción del tiempo de finalización de aplicaciones MapReduce iterativas sobre una nube híbrida en un despliegue de *cloud bursting*.

A continuación se muestran las conclusiones y contribuciones generales de esta tesis:

- El desarrollo de un entorno de extracción de datos estadísticos y trazas, llamado H-Stat, a partir de la ejecución de una aplicación MapReduce sobre una nube híbrida. Este entorno está compuesto por una colección de herramientas que facilitan la extracción de datos relevantes, los cuales permiten la visualización de la planificación de tareas, la caracterización de aplicaciones, y la calibración del sistema, así como la obtención de otros parámetros estadísticos relativos al tráfico de datos entre nubes.
- Un estudio de los principales factores que influyen en el rendimiento de una aplicación MapReduce sobre una nube híbrida en un despliegue de *cloud bursting*.
- El diseño, implementación y evaluación de varias estrategias de localidad de datos que permiten mejorar el rendimiento de una aplicación MapReduce iterativa en un despliegue de *cloud bursting*, haciendo uso de las características propias de Hadoop.
- El diseño, implementación y evaluación de una nueva estrategia de localidad de datos para Hadoop MapReduce que permite adaptarse al proceso inicial de migración de datos de un despliegue de *cloud bursting* a través de la implementación de una nueva política de planificación, mejorando así el rendimiento de las aplicaciones MapReduce iterativas en este tipo de entornos.
- El desarrollo de una metodología de análisis del coste económico que conlleva la ejecución de aplicaciones MapReduce iterativas sobre un entorno de *cloud bursting*. Mediante esta metodología se han analizado los costes de las distintas estrategias de localidad de datos propuestas, siendo nuestra estrategia la que mejores resultados obtiene, tanto en coste económico como en rendimiento de las aplicaciones.
- El desarrollo de una metodología de predicción del tiempo de finalización de aplicaciones MapReduce iterativas que se ejecuten sobre una plataforma de nube híbrida, a través de un despliegue de *cloud bursting*. Esta metodología está compuesta por tres pasos: una calibración del sistema, una calibración de la aplicación, y la aplicación de una expresión matemática. Dicha metodología se ha evaluado y validado a través de la predicción de cuatro aplicaciones MapReduce iterativas reales, junto con la comparación de ésta con otras metodologías de predicción del estado del arte. Podemos concluir que nuestra metodología obtiene una predicción media con mucha precisión, cometiendo un error en un intervalo entre el 1 % y el 10 %, independientemente del escenario, aplicación y ancho de banda del *enlace débil*. Esto no ocurre con otras metodologías, las cuales cometen un error de hasta casi un 80 % con respecto al resultado real obtenido para un escenario con un *enlace débil* de 100 Mbps.

Bibliografía

[1] Hybrid Cloud Market: Forecasts and Analysis (2013 – 2018). Market Research Insight, 2013.

[2] Abouzeid, A., Bajda-Pawlikowski, K., Abadi, D., Silberschatz, A., and Rasin, A. HadoopDB: an architectural hybrid of MapReduce and DBMS technologies for analytical workloads. *Proceedings of the VLDB Endowment 2*, 1 (2009), 922–933.

[3] Ahmad, F., Chakradhar, S. T., Raghunathan, A., and Vijaykumar, T. N. Tarazu: Optimizing MapReduce on Heterogeneous Clusters. *SIGPLAN Not. 47*, 4 (2012), 61–74.

[4] Alford, T., and Morton, G. The Economics of Cloud Computing, Addressing the Benefits of Infrastructure in the Cloud. *Booz Allen Hamilton* (2009).

[5] Amazon.com, I. Amazon Web Services. `https://aws.amazon.com`, 2006.

[6] Analysis, M. Hadoop Market Forecast 2017-2022. `https://www.marketanalysis.com/?p=279`, 2016.

[7] Apache. Apache Rumen. `https://hadoop.apache.org/docs/r1.2.1/rumen.html`.

[8] Apache, F. Apache Hadoop. `http://hadoop.apache.org`.

[9] Apache, F. HDFS Architecture. `http://hadoop.apache.org/docs/r2.6.0/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html`.

[10] Apache, F. Powered by Apache Hadoop. `https://wiki.apache.org/hadoop/PoweredBy`.

[11] Apache, F. YARN Architecture. `http://hadoop.apache.org/docs/r2.6.0/hadoop-yarn/hadoop-yarn-site/YARN.html`.

[12] Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., and Zaharia, M. A View of Cloud Computing. *Commun. ACM 53*, 4 (2010), 50–58.

[13] Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R. H., Konwinski, A., Lee, G., Patterson, D. A., Rabkin, A., Stoica, I., and Zaharia, M. Above the Clouds: A Berkeley View of Cloud Computing. Tech. Rep. UCB/EECS-2009-28, EECS Department, University of California, Berkeley, 2009.

[14] Bancilhon, F., and Ramakrishnan, R. *An amateur's introduction to recursive query processing strategies*, vol. 15. ACM, 1986.

[15] Barranco, I. R. ¿Qué es el Big Data? `http://www.ibm.com/developerworks/ssa/local/im/que-es-big-data`.

[16] Bicer, T., Chiu, D., and Agrawal, G. A framework for data-intensive computing with cloud bursting. In *CLUSTER '11: IEEE International Conference on Cluster Computing* (2011), pp. 169–177.

[17] Bock, H.-H. Clustering Methods: A History of K-Means Algorithms. In *Selected Contributions in Data Analysis and Classification*, Studies in Classification, Data Analysis, and Knowledge Organization. Springer Berlin Heidelberg, 2007, pp. 161–172.

[18] Brin, S., and Page, L. The Anatomy of a Large-scale Hypertextual Web Search Engine. *Comput. Netw. ISDN Syst. 30*, 1-7 (1998), 107–117.

[19] Bu, Y., Howe, B., Balazinska, M., and Ernst, M. D. HaLoop: efficient iterative data processing on large clusters. *Proceedings of the VLDB Endowment 3*, 1-2 (2010), 285–296.

[20] Cardosa, M., Wang, C., Nangia, A., Chandra, A., and Weissman, J. Exploring MapReduce Efficiency with Highly-distributed Data. In *MapReduce '11: 2on International Workshop on MapReduce and Its Applications* (New York, NY, USA, 2011), pp. 27–34.

[21] Chen, K., Powers, J., Guo, S., and Tian, F. CRESP: Towards Optimal Resource Provisioning for MapReduce Computing in Public Clouds. *IEEE Transactions on Parallel and Distributed Systems 25*, 6 (2014), 1403–1412.

[22] Claudel, B., Huard, G., and Richard, O. TakTuk, adaptive deployment of remote executions. In *HPDC '09: 18th ACM International Symposium on High Performance Distributed Domputing* (Munich, Germany, 2009), pp. 91–100.

[23] Clemente-Castelló, F. J., Cervera, S., Mayo, R., and Quintana-Ortí, E. S. Evaluating the Impact of Virtualization on Performance and Power Dissipation. In *CLOSER '14: 4th International Conference on Cloud Computing and Services Science* (Barcelona, Spain, 2014), pp. 513–518.

[24] Clemente-Castelló, F. J., Fernández, J. C., Mayo, R., and Quintana-Ortí, E. S. Adaptive Downtime for Live Migration of Virtual Machines. In *UCC '14: 7th IEEE/ACM International Conference on Utility and Cloud Computing* (London, UK, 2014), pp. 457–464.

[25] Clemente-Castello, F. J., Mayo, R., and Fernandez, J. C. Cost Model and Analysis of Iterative MapReduce Applications for Hybrid Cloud Bursting. In *SCRAMBL'17: 3rd International Workshop on Scalable Computing For Real-Time Big Data Applications* (Madrid, Spain, 2017).

[26] Clemente-Castelló, F. J., Mayo, R., Fernández, J. C., Quintana-Ortí, E. S., Nicolae, B., and Katrinis, K. Estudio Preliminar de una Nube Híbrida: Análisis de Rendimiento con MapReduce. In *JP '15: XXVI Edición de las Jornadas de Paralelismo* (Córdoba, Spain, 2015), pp. 274–281.

[27] Clemente-Castelló, F. J., Mayo, R., and Quintana-Ortí, E. S. Estudio y Requisitos de la Simulación de Sistemas de Cloud Computing. In *JP '13: XXIV Edición de las Jornadas de Paralelismo* (Madrid, Spain, 2013), pp. 199–204.

[28] Clemente-Castelló, F. J., Mayo, R., Valls, S. C., and Quintana-Ortí, E. S. Efecto de la Virtualización sobre el Rendimiento, la Productividad y el Consumo. In *JP '12: XXIII Edición de las Jornadas de Paralelismo* (Elche, Spain, 2012), pp. 181–186.

[29] Clemente-Castelló, F. J., Nicolae, B., Katrinis, K., Rafique, M. M., Mayo, R., Fernández, J. C., and Loreti, D. Enabling Big Data Analytics in the Hybrid Cloud Using Iterative MapReduce. In *UCC '15: 8th IEEE/ACM International Conference on Utility and Cloud Computing* (Limassol, Cyprus, 2015), pp. 290–299.

[30] Clemente-Castelló, F. J., Nicolae, B., Mayo, R., and Fernández, J. C. Performance Model of MapReduce Iterative Applications for Hybrid Cloud Bursting. *IEEE Transactions on Parallel and Distributed Systems* (2017). (Under Review).

[31] Clemente-Castelló, F. J., Nicolae, B., Mayo, R., Fernández, J. C., and Rafique, M. M. On exploiting data locality for iterative MapReduce applications in hybrid clouds. In *BDCAT '16: 3rd IEEE/ACM International Conference on Big Data Computing, Applications and Technologies* (Shanghai, China, 2016), pp. 118–122.

[32] Clemente-Castello, F. J., Nicolae, B., Rafique, M. M., Mayo, R., and Fernandez, J. C. Evaluation of Data Locality Strategies for Hybrid Cloud Bursting of Iterative MapReduce. In *CCGrid'17: 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing* (Madrid, Spain, 2017).

[33] Computing, R. C. Openstack. `https://www.openstack.org`.

[34] Dean, J., and Ghemawat, S. MapReduce: Simplified Data Processing on Large Clusters. *Commun. ACM 51*, 1 (2008), 107–113.

[35] Edmonds, J. Matroids and the greedy algorithm. *Mathematical programming 1*, 1 (1971), 127–136.

[36] Ek, D., and Lorentzon, M. Spotify. `https://www.spotify.com`, 2008.

[37] EMC. IDC Digital Universe Study: Big Data, Bigger Digital Shadows and Biggest Growth in the Far East. `http://www.whizpr.be/upload/medialab/21/company/Media_Presentation_2012_DigiUniverseFINAL1.pdf`, 2012.

[38] EMC. Data Growth, Business Opportunities, and the IT Imperativest. `http://www.emc.com/leadership/digital-universe/2014iview/executive-summary.html`, 2014.

[39] Erl, T., Puttini, R., and Mahmood, Z. *Cloud computing: concepts, technology & architecture*. Pearson Education, 2013.

[40] Fernández, A., del Río, S., López, V., Bawakid, A., del Jesus, M. J., Benítez, J. M., and Herrera, F. Big Data with Cloud Computing: an insight on the computing environment, MapReduce, and programming frameworks. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 4*, 5 (2014), 380–409.

[41] Foundation, T. A. S. Apache Mahout. `http://mahout.apache.org`.

[42] Gartner. Gartner Survey. `http://www.gartner.com/newsroom/id/2848718`.

[43] Godard, S. Sysstat utilities home page. `http://sebastien.godard.pagesperso-orange.fr`.

[44] Google, I. G Suite. `https://gsuite.google.com`.

[45] Google, I. Google App Engine. `https://appengine.google.com`, 2008.

[46] Graham, R. L. Bounds for Certain Multiprocessing Anomalies. *Bell System Technical Journal 45*, 9 (1966), 1563–1581.

[47] Gunarathne, T., Wu, T.-L., Qiu, J., and Fox, G. MapReduce in the Clouds for Science. In *CloudCom '10: 2on IEEE Conference on Cloud Computing Technology and Science* (2010), pp. 565–572.

[48] Guo, T., Sharma, U., Wood, T., Sahu, S., and Shenoy, P. Seagull: Intelligent Cloud Bursting for Enterprise Applications. In *USENIX ATC '12: Conference on Annual Technical Conference* (Berkeley, CA, USA, 2012), pp. 33–33.

[49] Hadoop. Rack Awareness. `http://hadoop.apache.org/docs/r2.7.2/hadoop-project-dist/hadoop-common/RackAwareness.html`.

[50] Heintz, B., Chandra, A., and Weissman, J. Cross-Phase Optimization in MapReduce. *Cloud Computing for Data-Intensive Applications* (2014), 277–302.

[51] Herodotou, H., Lim, H., Luo, G., Borisov, N., Dong, L., Cetin, F. B., and Babu, S. Starfish: A Self-tuning System for Big Data Analytics. In *In CIDR* (2011), pp. 261–272.

[52] Hofmann, P., and Woods, D. Cloud Computing: The Limits of Public Clouds for Business Applications. *IEEE Internet Computing 14* (2010), 90–93.

[53] Huang, S., Huang, J., Dai, J., Xie, T., and Huang, B. The HiBench benchmark suite: Characterization of the MapReduce-based data analysis. In *ICDEW '10: 26th IEEE International Conference on Data Engineering Workshops* (2010), pp. 41–51.

[54] Imai, S., Chestna, T., and Varela, C. A. Accurate Resource Prediction for Hybrid IaaS Clouds Using Workload-Tailored Elastic Compute Units. In *UCC '13: 6th IEEE/ACM International Conference on Utility and Cloud Computing.* (Dresden, Germany, 2013), pp. 171–178.

[55] Isard, M., Prabhakaran, V., Currey, J., Wieder, U., Talwar, K., and Goldberg, A. Quincy: Fair Scheduling for Distributed Computing Clusters. In *SOSP '09: 22nd ACM Symposium on Operating Systems Principles* (2009), pp. 261–276.

[56] Iserte, S., Clemente-Castelló, F. J., Mayo, R., and Quintana-Ortí, E. S. Enabling GPU Virtualization in Cloud Environments. In *CLOSER '16: 6th International Conference on Cloud Computing and Services Science* (Rome, Italy, 2016), pp. 249–256.

[57] Javadi, B., Abawajy, J., and Buyya, R. Failure-aware Resource Provisioning for Hybrid Cloud Infrastructure. *J. Parallel Distrib. Comput. 72*, 10 (2012), 1318–1331.

[58] Kashef, M. M., and Altmann, J. A cost model for hybrid clouds. In *International Workshop on Grid Economics and Business Models* (2011), Springer, pp. 46–60.

[59] Kleinberg, J. M. Authoritative sources in a hyperlinked environment. *Journal of the ACM 46*, 5 (1999), 604–632.

[60] Kllapi, H., Sitaridi, E., Tsangaris, M. M., and Ioannidis, Y. Schedule Optimization for Data Processing Flows on the Cloud. In *SIGMOD '11: ACM International Conference on Management of Data* (2011), pp. 289–300.

[61] Kondo, D., Javadi, B., Malecot, P., Cappello, F., and Anderson, D. P. Cost-benefit Analysis of Cloud Computing Versus Desktop Grids. In *IPDPS '09: IEEE International Symposium on Parallel & Distributed Processing* (Washington, DC, USA, 2009), pp. 1–12.

[62] Kosar, T., Arslan, E., Ross, B., and Zhang, B. StorkCloud: Data Transfer Scheduling and Optimization As a Service. In *ScienceCloud'13: 4th ACM Workshop on Scientific Cloud Computing* (New York, New York, USA, 2013), pp. 29–36.

[63] Lama, P., and Zhou, X. AROMA: Automated Resource Allocation and Configuration of MapReduce Environment in the Cloud. In *ICAC '12: 9th International Conference on Autonomic Computing* (New York, NY, USA, 2012), pp. 63–72.

[64] Laney, D. 3D Data Management: Controlling Data Volume, Velocity, and Variety., 2012.

[65] Laoutaris, N., Sirivianos, M., Yang, X., and Rodriguez, P. Inter-datacenter Bulk Transfers with Netstitcher. *SIGCOMM Comput. Commun. Rev. 41* (2011), 74–85.

[66] Magazine, F. Mind Boggling. `http://www.forbes.com/sites/bernardmarr/2015/\09/30/big-data-20-mind-boggling-facts-everyone-must-read`, 2015.

[67] Mattess, M., Calheiros, R. N., and Buyya, R. Scaling MapReduce applications across hybrid clouds to meet soft deadlines. In *AINA'13: IEEE 27th International Conference on Advanced Information Networking and Applications* (Barcelona, Spain, 2013), pp. 629–636.

[68] McCafferty, D. Surprising Statistics About Big Data. `http://www.baselinemag.com/analytics-big-data/slideshows/surprising-statistics-about-big-data.html`.

[69] Mell, P., and Grance, T. The NIST definition of cloud computing.

[70] Microsoft. Microsoft Azure. `https://azure.microsoft.com`, 2010.

[71] Morton, K., Balazinska, M., and Grossman, D. ParaTimer: A Progress Indicator for MapReduce DAGs. In *SIGMOD '10: ACM International Conference on Management of Data* (2010), pp. 507–518.

[72] Morton, K., Friesen, A., Balazinska, M., and Grossman, D. Estimating the progress of MapReduce pipelines. In *ICDE '10: 26th IEEE International Conference on Data Engineering* (2010), pp. 681–684.

[73] Netflix, I. Netflix. `https://www.netflix.com`.

[74] Ohnaga, H., Aida, K., and Abdul-Rahman, O. Performance of Hadoop Application on Hybrid Cloud. In *ICCCRI '15: International Conference on Cloud Computing Research and Innovation* (2015), pp. 130–138.

[75] Ousterhout, K., Rasti, R., Ratnasamy, S., Shenker, S., and Chun, B.-G. Making Sense of Performance in Data Analytics Frameworks. In *NSDI'15: The 12th USENIX Conference on Networked Systems Design and Implementation* (Oakland, USA, 2015), pp. 293–307.

[76] Page, L., Brin, S., Motwani, R., and Winograd, T. The PageRank citation ranking: bringing order to the web.

[77] Pillet, V., Labarta, J., Cortes, T., and Girona, S. Paraver: A tool to visualize and analyze parallel code. In *WoTUG-18: Transputer and occam Developments* (1995), vol. 44, pp. 17–31.

[78] Polo, J., Carrera, D., Becerra, Y., Beltran, V., Torres, J., and Ayguade, E. Performance Management of Accelerated MapReduce Workloads in Heterogeneous Clusters. In *ICPP '10: 39th IEEE International Conference on Parallel Processing* (Washington, DC, USA, 2010), pp. 653–662.

[79] Rackspace, I. Rackspace Hybrid Cloud. `https://www.rackspace.com/es/cloud/hybrid`, 2012.

[80] Roman, R.-I., Nicolae, B., Costan, A., and Antoniu, G. Understanding Spark Performance in Hybrid and Multi-Site Clouds. In *BDAC '15: 6th International Workshop on Big Data Analytics: Challenges and Opportunities (in conjunction with SC15)* (Austin, United States, 2015).

[81] Ruiz-Alvarez, A., and Humphrey, M. Toward Optimal Resource Provisioning for Cloud MapReduce and Hybrid Cloud Applications. In *BDC '14: IEEE/ACM International Symposium on Big Data Computing* (2014), pp. 74–82.

[82] Seidl, T., Boden, B., and Fries, S. CC-MR – Finding Connected Components in Huge Graphs with MapReduce. In *ECML PKDD '12: Machine Learning and Knowledge Discovery in Databases* (Berlin, Heidelberg, 2012), pp. 458–473.

[83] Sharma, B., Wood, T., and Das, C. R. Hybridmr: A hierarchical MapReduce scheduler for hybrid data centers. In *ICDCS'13: 33rd IEEE International Conference on Distributed Computing Systems* (2013), IEEE, pp. 102–111.

[84] Silva, J. a. N., Veiga, L., and Ferreira, P. Heuristic for Resources Allocation on Utility Computing Infrastructures. In *MGC '08: 6th ACM International Workshop on Middleware for Grid Computing* (2008), pp. 9:1–9:6.

[85] SoftLayer Technologies, I. Softlayer. `https://www.softlayer.com`, 2005.

[86] Spark, A. Apache Spark: Lightning-fast cluster computing. `http://spark.apache.org`, 2014.

[87] Suen, C.-H., Kirchberg, M., and Lee, B. S. Efficient Migration of Virtual Machines between Public and Private Cloud. In *CloudCom '11: 3er IEEE International Conference on Cloud Computing Technology and Science.* (Athens, Greece, 2011), pp. 549–553.

[88] Tian, F., and Chen, K. Towards optimal resource provisioning for running MapReduce programs in public clouds. In *CLOUD '11: IEEE International Conference on Cloud Computing* (Washington DC, USA, 2011), pp. 155–162.

[89] Truong, H.-L., and Dustdar, S. Composable cost estimation and monitoring for computational applications in cloud computing environments. *Procedia Computer Science 1*, 1 (2010), 2175–2184.

[90] Tsai, W.-T., Zhong, P., Elston, J., Bai, X., and Chen, Y. Service replication strategies with MapReduce in clouds. In *ISADS '11: 10th International Symposium on Autonomous Decentralized Systems* (Kobe, Japan, 2011), pp. 381–388.

[91] Van den Bossche, R., Vanmechelen, K., and Broeckhove, J. Cost-Optimal Scheduling in Hybrid IaaS Clouds for Deadline Constrained Workloads. In *CLOUD '10: 3rd IEEE International Conference on Cloud Computing.* (Miami, 2010), pp. 228–235.

[92] Verma, A., Cherkasova, L., and Campbell, R. H. ARIA: Automatic Resource Inference and Allocation for MapReduce Environments. In *ICAC '11: 8th ACM International Conference on Autonomic Computing* (Karlsruhe, Germany, 2011).

[93] Verma, A., Cherkasova, L., and Campbell, R. H. Resource provisioning framework for MapReduce jobs with performance goals. In *Middleware '11: 12th ACM/IFIP/USENIX International Middleware Conference* (Lisbon, Portugal, 2011), pp. 165–186.

[94] VMWare, I. vCloud Air. `http://www.vmware.com/cloud-services/infrastructure.html`, 2013.

[95] Wang, G., Butt, A. R., Pandey, P., and Gupta, K. A simulation approach to evaluating design decisions in MapReduce setups. In *MASCOTS '09: IEEE International Symposium on Modeling, Analysis Simulation of Computer and Telecommunication Systems* (2009), pp. 1–11.

[96] Wang, L., Zhan, J., Luo, C., Zhu, Y., Yang, Q., He, Y., Gao, W., Jia, Z., Shi, Y., Zhang, S., et al. Bigdatabench: A big data benchmark suite from internet services. In *HPCA '14: 20th IEEE International Symposium on High Performance Computer Architecture* (2014), pp. 488–499.

[97] Wang, Y., and Shi, W. Budget-Driven Scheduling Algorithms for Batches of MapReduce Jobs in Heterogeneous Clouds. *IEEE Transactions on Cloud Computing 2*, 3 (2014), 306–319.

[98] White, T. *Hadoop: The definitive guide*. O'Reilly Media, Inc., 2012.

[99] Wikipedia. Wikipedia Enwiki Dumps. `https://dumps.wikimedia.org/enwiki/`, 2016.

[100] Wolf, J., Rajan, D., Hildrum, K., Khandekar, R., Kumar, V., Parekh, S., Wu, K.-L., and Balmin, A. FLEX: A Slot Allocation Scheduling Optimizer for MapReduce Workloads. In *Middleware '10: 11th ACM/IFIP/USENIX International Middleware Conference* (Bangalore, India, 2010), pp. 1–20.

[101] Xie, J., Yin, S., Ruan, X., Ding, Z., Tian, Y., Majors, J., Manzanares, A., and Qin, X. Improving MapReduce performance through data placement in heterogeneous Hadoop clusters. In *IPDPSW '10: IEEE International Symposium on Parallel Distributed Processing, Workshops and Phd Forum* (2010), pp. 1–9.

[102] Zaharia, M., Konwinski, A., Joseph, A. D., Katz, R., and Stoica, I. Improving MapReduce Performance in Heterogeneous Environments. In *OSDI '08: 8th USENIX Conference on Operating Systems Design and Implementation* (2008), pp. 29–42.

[103] Zhang, H., Jiang, G., Yoshihira, K., and Chen, H. Proactive Workload Management in Hybrid Cloud Computing. *IEEE Transactions on Network and Service Management 11*, 1 (2014), 90–100.

[104] Zhang, H., Jiang, G., Yoshihira, K., Chen, H., and Saxena, A. Intelligent Workload Factoring for a Hybrid Cloud Computing Model. In *SERVICES - I '09: IEEE Conference on Services* (July 2009), pp. 701–708.

[105] Zhang, X., Yang, L. T., Liu, C., and Chen, J. A scalable two-phase top-down specialization approach for data anonymization using MapReduce on cloud. *IEEE Transactions on Parallel and Distributed Systems 25*, 2 (2014), 363–373.

[106] Zhang, Z., Cherkasova, L., and Loo, B. T. Benchmarking Approach for Designing a MapReduce Performance Model. In *ICPE '13: 4th ACM/SPEC International Conference on Performance Engineering* (2013), pp. 253–258.

[107] Zhang, Z., Cherkasova, L., and Loo, B. T. Performance Modeling of MapReduce Jobs in Heterogeneous Cloud Environments. In *CLOUD '13: 6th IEEE International Conference on Cloud Computing* (Washington, DC, USA, 2013), pp. 839–846.

[108] Zhang, Z., Cherkasova, L., and Loo, B. T. Parameterizable benchmarking framework for designing a MapReduce performance model. *Concurrency and Computation: Practice and Experience 26*, 12 (2014), 2005–2026.

[109] Zhao, W., Ma, H., and He, Q. Parallel K-Means Clustering Based on MapReduce. In *CloudCom '09: 1st International Conference on Cloud Computing* (Beijing, China, 2009), pp. 674–679.