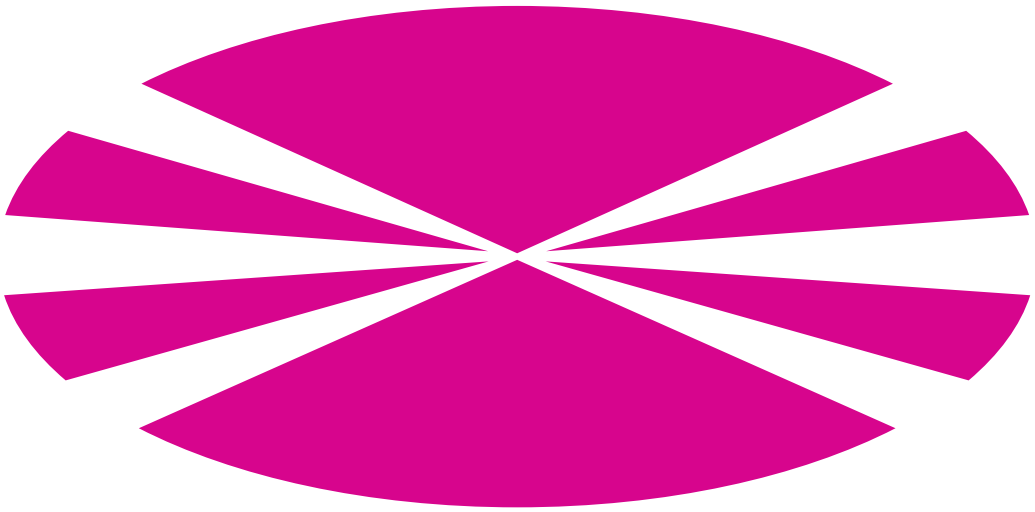

UNIVERSIDADE DA CORUÑA

SCHOOL OF INFORMATICS

Department of Electronics and Systems

Assessment, Design and Implementation of a Private Cloud for MapReduce Applications

Author: Marcos Salgueiro Balsa

Patricia González Gómez

Tutoring: Tomás Fernández Pena

José Carlos Cabaleiro Domínguez

Date: A Coruña, June 2013

Give a man a fish, and you'll feed him for a day.
Teach a man to fish, and you'll feed him for a lifetime.
Anne Isabella Thackeray Ritchie

Great spirits have always encountered violent opposition from mediocre minds.
Albert Einsten

The supreme art of war is to subdue the enemy without fighting.
Sun Tzu, *The Art of War*

[...] It takes these very simple-minded instructions – "Go fetch a number, add it to this number, put the result there, perceive if it's greater than this other number" – but executes them at a rate of, let's say, 1,000,000 per second. At 1,000,000 per second, the results appear to be magic.
Steven Paul Jobs

Summary

The history of computation has seen how the technology's unending evolution has promoted changes in its ways and means. Today, *tablets* and *smartphones*, quantitatively inferiors managing and memorizing numbers, camp freely in a global market saturated with options. The tendency is clear: users will get to use more than one device to access the Internet and will like to have all of their data synchronized and at hand, all the time.

But that is only a part in the equation. At the other side of every service request there lays a server that must deal with an ever increasingly troubling traffic volume, while it maintains response delivery at outstanding delay times — low latency "may" have helped the infant Google rise above the competition. If we also added that the idea of surrounding every implementation effort with energetic efficiency is a transcendental requisite and not simply a good practice, we would have a perfect environment for the proliferation of new distributed paradigms as the *Cloud*. The Cloud is not an intrinsically new idea but an old concept abstraction: *virtualization*. The clouds' cornerstone is flexibility.

Another technology that is constantly making it to the headlines is *MapReduce*. If the Cloud centers around easing infrastructure exploitation, MapReduce's core strength lies in its speeding up driving large masses of unstructured data; with makes them an extraordinary computational tandem. This project puts forth a solution that allows for drawing on computational resources available exploiting both technologies together. Special emphasis has been placed in flexibility of access, being a web browser the only application

required to use the service; in simplifying the virtual cluster configuration, by including a self-managed minimum deployment; and in transparency and extensibility, by freeing source code and documentation as *OSS*, favoring its usage as starting point for larger installations.

Keywords

Distributed Computing, Virtualization, Cloud Computing, MapReduce, OpenStack, Hadoop.

Contents

List of Figures

Chapter 1

Abstract

Over the last years there has been a continuous increase in the quantity of information generated with the Internet as the main driver. Furthermore, this information has reshaped from structured — and thus, susceptible to being expressed following a relational schema — to heterogeneous, which has kick-started the necessity to alter the way it is stored and transformed. As the figure 1.1 shows, those that were the undisputed back-end queens — relational database systems mostly — are seeing how their role is fading away due to their incapability to efficiently save unrelated heterogeneity.

As another related dimension, in the year 2000 many .com companies started upgrading their data centers to accommodate the inexorable demand peak that was going to follow. But it never came; and the bubble burst. What happened then was general underutilization — only `10%` of Amazon's global computational resources were in use — that pushed the search for alternative means to export the surplus as a product. Amazon's own initiative unfolded in 2006 with the *AWS* (*Amazon Web Services*) appearance. AWS, among others, implements a public API for flexible on-demand infrastructure provisioning.

Since then, similar projects have proliferated generalizing how private clusters' unused computational capacity is to be serviced, trying to stay API-compatible with the AWS to facilitate interoperability and thus avoid client's

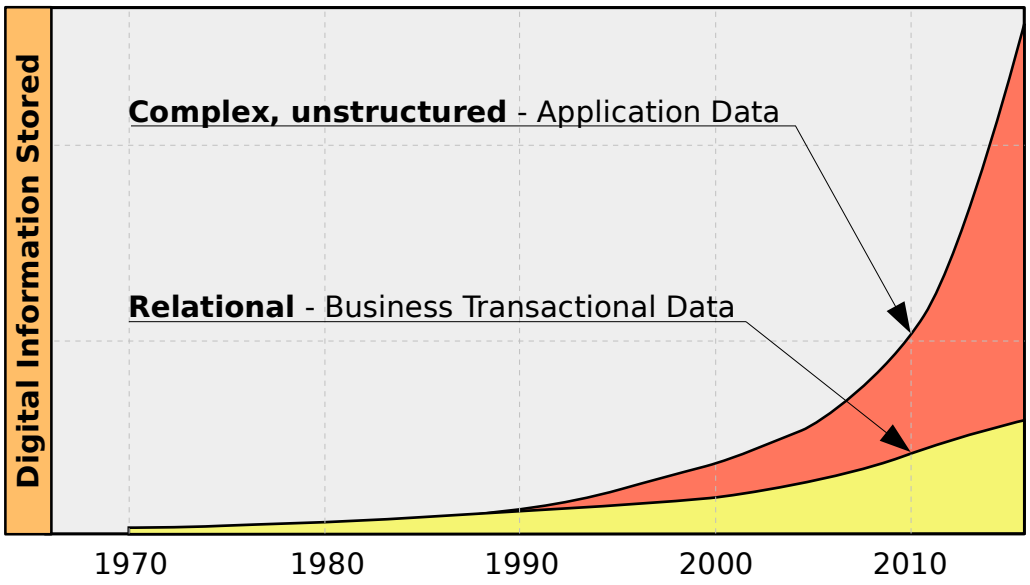

Figure 1.1: Demand in exponential growth. Source: *Cloudera Inc.*

swapping to more flexible providers.

Meanwhile, Google was also in the search for new mechanisms to exploit, with high performance and securely, their own private infrastructure to evolve the capability of their services. MapReduce, as a way to massively execute thousands transformations on input data, became a reality to thrust the generation of Google's humongous inverted index of the Internet [2]. Forthcoming contributions from Nutch's developers — by that time an Internet search engine prototype — to the MapReduce paradigm at *Yahoo!*, would traduce into the appearance of today's *de facto* standard in the field: Hadoop. Nowadays Hadoop is used in a myriad of backgrounds, ranging from travel booking sites to storing and servicing mobile data, ecommerce, image processing applications or searching for new forms of energy.

So, by stacking a MapReduce implementation atop elastic infrastructure an optimal exploitation of computational resources would be attainable, rapidly expanding or shrinking them on-demand, while simultaneously reducing the overall energy consumption required to accomplish processings

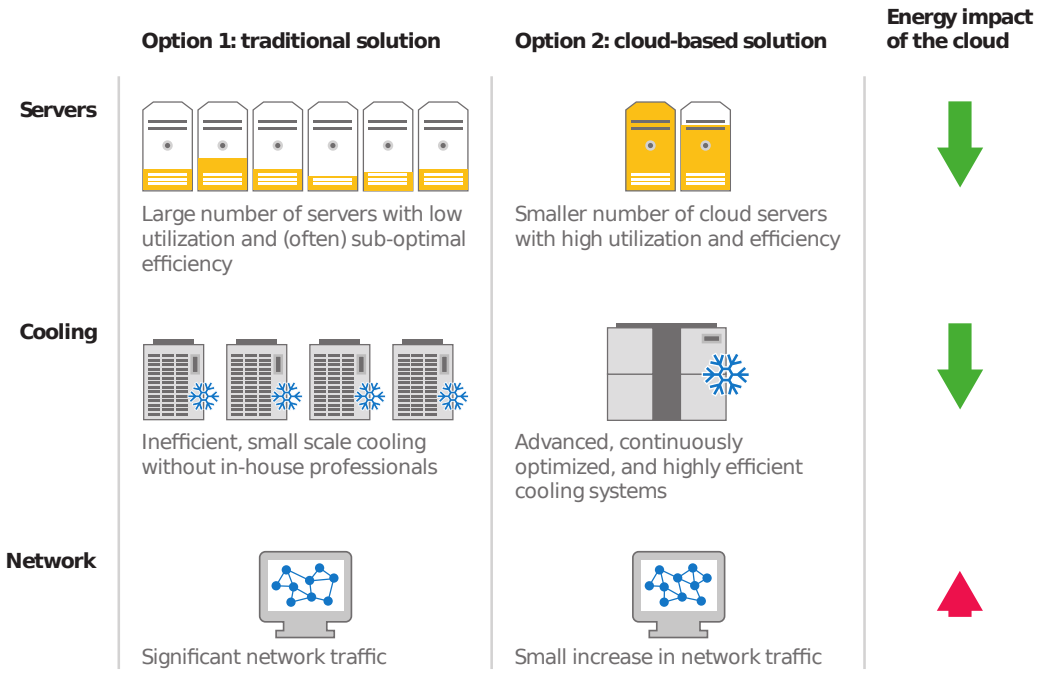

Figure 1.2: Energy savings. Source: [1]

(Figure 1.2).

1.1 Goals

The main goal with this project is to study the feasibility to develop a solution for a Cloud to drive MapReduce applications, with no need to know the particular Cloud structure and/or Hadoop configuration parameters.

In order to achieve such a simple execution model without compromising performance or applicability, a thorough analysis on different *IaaS Frameworks* will be carried out. Their features will be evaluated inside a virtual testing environment to finally narrow the selection to only one. Once an IaaS Framework had been chosen, the attention will be put towards choosing a MapReduce implementation to install over our virtual infrastructure.

Nonetheless, a mechanism to forward MapReduce execution requests will

be devised and implemented trying to focus on simplicity and universal access to this human-cloud-mapreduce interface. Yet, this transparency mustn't become an obstacle in exploiting the application or in fetching processed results. Privacy and security in communications and storage will be conveniently defined; we shan't forget it will be developed as a scaled-down model which could be infinitely scaled out.

1.2 Arrangement of the Document

The contents within this document are distributed as stated next. This first chapter introduces development guidelines in the abstract. Chapter 2 puts the reader closer to the fundamental Cloud Computing concepts — like its general architecture or virtualization —, along with the ones from the MapReduce paradigm. Chapter **??** describes an empirical evaluation of four private IaaS Cloud frameworks. Chapter **??** explores OpenStack Folsom's modular structure and particular inner workings. Analogically, chapter **??** unveils Hadoop's peculiarities as a MapReduce implementation.

The subsequent chapters center on detailing the project from diverse vantage points. Chapter **??** contains a series of design decisions and their accompanying UML diagrams. Chapter **??** gathers an analysis on performance in a real testing cluster. Chapter cap:conclusiones analyzes related papers highlighting how they compare to this solution. Finally, the main contributions of this project are discussed in addition to proposing future improvements to the implementation.

Two annexes have also been included. Annex **??** guides the reader throughout a quick single node installation. Annex **??** covers the definition of some of the concepts and technologies referred to in this text.

Chapter 2

Background

This second chapter tries to acquaint the reader with the key concepts that define Cloud Computing as well as the MapReduce archetype. Later successive elaborations to the project will lay on top of them.

2.1 Cloud Computing

In essence, Cloud Computing, or Cloud for short, is a distributed computing model that attempts to ease the consumption on-demand of that distributed infrastructure, by exporting it as virtual computational resources, platforms or services. However it may seem, the Cloud is no new technology but it introduces a new manner to exploit idle computing capacity. What it intends is to make orchestration of enormous data centers more flexible, so as to allow a user to start or destroy virtual machines as required — Infrastructure as a Service (*IaaS*) —, leverage a testing environment over a particular Operating System or software platform — Platform as a Service (*PaaS*) — or use a specific service like remote backup — Software as a Service (*SaaS*). Figure 2.1 shows the corresponding high level layer diagram of a generic Cloud.

Different IaaS frameworks will cover the functionality that is required to drive the cloud-defining *physical* infrastructure. Nonetheless, an effort to analyze, design, configure, install and maintain the intended service will

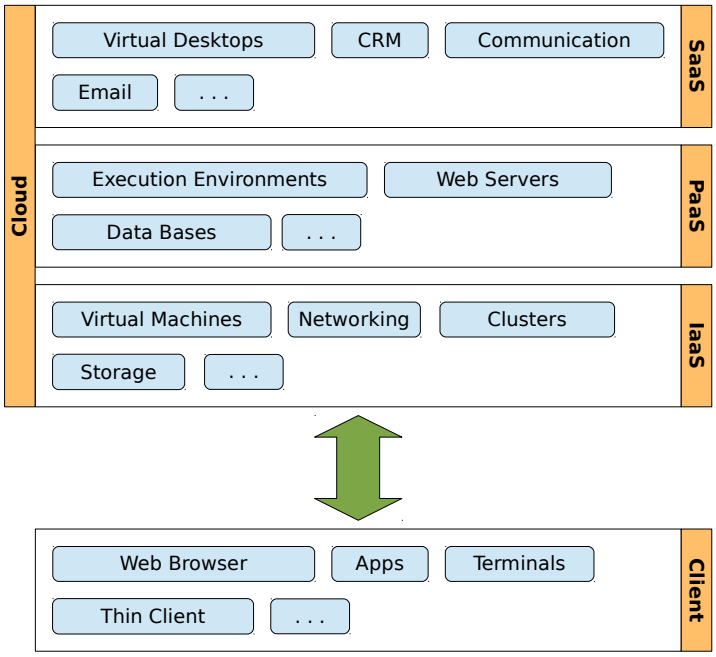

Figure 2.1: Layers in a cloud in production

be needed, bearing in mind that the degree of elaboration grows from IaaS services to SaaS ones. In effect, PaaS and SaaS layers are lied supported by those immediately under — software is implemented over a particular platform which, in turn, is also build upon a physical layer. Every Cloud Framework focuses on giving the option to configure a stable environment in which to run virtual machines defined by four variables: Virtual CPU count, virtual RAM, virtual persistent memory and virtual networking devices. Such an environment makes it possible to deploy virtual clusters upon which to install platforms or services to be subsequently consumed by users, bringing up the software layers that give form PaaS and SaaS paradigms respectively.

No less important cuestions like access control, execution permissions, quota or persistent or safe storage will also be present in all of the frameworks.

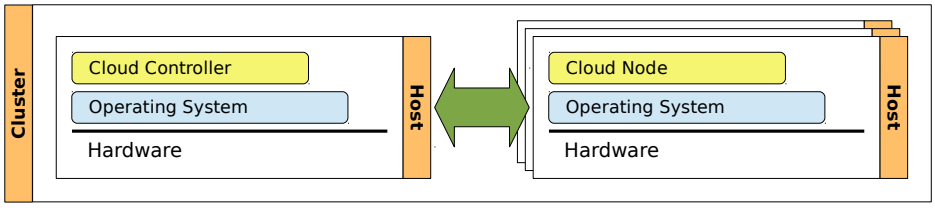

Figure 2.2: Cloud Controller and Cloud Node

2.1.1 Architecture

Figure 2.1 showed possible layers that could be found in a cloud deployment. Depending on the layers that are implemented, the particular framework and the role played by the cluster node, different particular modules will appear to make possible the consumption of configured services. These modules may be though of as Cloud subsystems that connect each one of the parts that are required to execute virtual machines. Those virtual machines' capabilities are defined by the four variables previously discussed — VCPUS, RAM, HDD and networking. As there is no methodology dictating how those subsystems should be in terms of size and responsibility, and thus, each framework makes its own modular partition regarding infrastructure management.

Setting modularity apart, one common feature among different clouds is the separation of responsibility in two main roles: *Cloud Controller* and *Cloud Node*. Figure 2.2 shows a generic Cloud deployment in a cluster with both roles defined. The guidelines followed for having this two roles lies close to *Master-Slave* architectures' approach. In those, in the abstract, there's a set of computers labeled as coordinators which are expected to control execution, and another set made up with those machines that are to carry out the actual processing.

Within this general role distribution in a cluster, host computers or cluster nodes — labeled as Cloud Controllers or Cloud Nodes — cooperate in a synchronized fashion through *NTP* (*Network Time Protocol*) and communicate via message passing supported by asynchronous queues. To store

services' metadata and status they typically draw upon a *DBMS* (*Data Base Management System*) implementation, which is regularly kept running in a dedicated cluster node set sharded (distributed) between the members of the set.

Although there is no practical restriction to configuring both Cloud Controller and Cloud Node within a single computer in a cluster, this approach should be limited to development environments due to the considerable impact in performance that it would carry.

Cloud Controller

The fundamental task for a Controller is to maintain all of the cloud's constituent modules working together by coordinating their cooperation. As an example, it is a Controller's duty to:

- Authentication and authorization control.
- Available infrastructure resources recount.
- Quota management.
- Usage balance.
- User and project inventory.
- API exposure for service consumption.
- Real time cloud monitoring.

Being an essential part of a cloud as it is, the Controller node (not to be mistaken for the Cloud Node) is usually replicated in physically distinct computers. Figure 2.3 shows a Cloud Controller's architecture from a high level perspective.

As a general rule, clients will interact with clouds through a Web Service API — mostly *RESTful* APIs (*REpresentational State Transfer*). Those APIs vary slightly from company to vendor as usual, which forces clients to

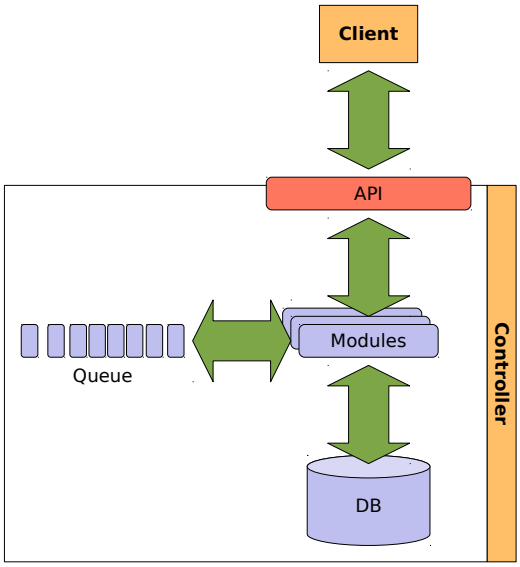

Figure 2.3: Cloud Controller in detail

be partially coupled to clouds. That is why there has been an increasing trend for unifying and standardizing those APIs in order to guarantee compatibility inter-framework. Of special mention is the cloud standard proposed by the *Open Grid Forum*: *OCCI* (*Open Cloud Computing Interface* [6]).

Los módulos constitutivos del cloud soportan el grueso funcional del controlador. Cada uno de ellos tendrá un fin bien definido, y así aparecen módulos de red, módulos de control de acceso y seguridad, módulos de almacenamiento, etc. Muchos de ellos ya existían previa aparición de los cloud, sólo que funcionaban de manera aislada. Los módulos se comunican entre sí usando una cola de mensajes asíncrona que garantiza la comunicación también con el exterior del controlador, es decir, con el resto de los nodos del clúster que participa en el cloud. Y por último, la base de datos, soporte de almacenamiento de información compartida por los módulos de la nube, jugará un papel central, ya que la información global de estado determina el comportamiento general del cloud. Si estos datos se corrompiesen, el funcionamiento general podría verse gravemente afectado.

Los requisitos operativos varían según el framework en cuestión y el tipo de servicio que se pretenda garantizar, pero generalmente, algo que se aproxime a 10 GB de RAM, cuádruple núcleo, *Gigabit Ethernet* y algún TB de almacenamiento sería más que suficiente.

Cloud Node

Si el controlador de la nube es quien se encarga de mantenerla funcionando haciendo de pegamento de las partes, el soporte de procesado se lo llevan los nodos. Esto es, la CPU, la memoria de ejecución y la memoria de disco de cada servidor virtual se van a tomar prestadas de las correspondientes variables CPU, RAM y disco de los anfitriones (nodos) del clúster real.

Los nodos del cloud pueden ser, en cierto grado, heterogéneos en cuanto a características hardware. Ellos van a configurar un grupo de recursos que visto desde fuera será como un todo homogéneo, donde la suma de las capacidades de cada nodo es la dimensión de la infraestructura de la nube. Además, este espacio homogéneo podrá ser provisto, como ya se ha comentado, bajo demanda. Es tarea del controlador del cloud determinar el modo óptimo de reparto de los servidores virtuales entre los nodos, atendiendo, entre otras, a las características técnicas de ambos —nodo *físico* y máquina virtual.

El subsistema más importante de los nodos del cloud es el *hipervisor* o monitor de máquina virtual (*VMM*). Se encarga de hacer posible la ejecución de servidores virtuales (instancias) creando la arquitectura virtual necesaria y un *dominio virtual de ejecución* que será manejado por el hipervisor y el kernel del sistema operativo del nodo del cloud. Para generar esta arquitectura existen fundamentalmente tres técnicas: *emulación*, *paravirtualización* y *virtualización hardware* o *virtualización completa*. Los diferentes hipervisor soportarán en distinta medida cada una de ellas; siendo mayoritaria la implementación de una sola de esas técnicas.

2.1.2 Técnicas de virtualización

A continuación se hará un repaso conciso sobre los principales métodos de virtualización.

Emulación

La emulación es la técnica de virtualización más general, en sentido de que no requiere nada especial en el hardware subyacente. En contrapartida, la penalización de rendimiento es la más acusada. En la virtualización por emulación, todas las estructuras que sostienen la ejecución de los servidores virtuales se generan como copias software que emulan el comportamiento de sus análogas hardware. Es decir, cada instrucción máquina que haya de ser ejecutada en la estructura emulada tendrá que ser traducida *en el momento* a otra instrucción que pueda procesar la arquitectura real. En función de la implementación del intérprete y de las diferencias entre hardware emulado y hardware real, aparecerá una siempre elevada sobrecarga que en la mayoría de los casos impide que la emulación sea utilizada en entornos de alto rendimiento. Sin embargo, gracias a su flexibilidad operativa, sí es un buen mecanismo de soporte de arquitecturas *legacy*. Además, el kernel del huésped —el kernel del sistema operativo que corre en el servidor virtual— no precisa modificación, y el kernel del anfitrión sólo cargar un módulo, normalmente.

Virtualización hardware

La virtualización hardware, por contra, permite que los procesos del huésped corran encima del hardware real directamente, sin necesidad de intermediarios. Lógicamente, esto propicia una aceleración considerable con respecto a la emulación, pero impone que el hardware dé un trato especial a esos procesos virtuales. En el caso de la CPU, tanto las de AMD como las de Intel

soportan la ejecución virtual de procesos —o dicho de otra forma, la ejecución de procesos pertenecientes al dominio virtual con reducida penalización al rendimiento— siempre que estén presentes las extensiones convenientes (*SVM* para AMD y *VT-X* para Intel [7]). Igual que en la emulación, no es necesario modificar el kernel del huésped, lo cual es importante ya que reduciría la diversidad de sistemas operativos instalables, y el kernel anfitrión sólo ha de cargar un módulo, generalmente. Apuntar, por último, que la arquitectura hardware se presenta a las máquinas virtuales tal y como es, es decir, sin ninguna alteración software intermedia.

Paravirtualización

La paravirtualización utiliza una aproximación diferente. Para empezar, es indispensable que el kernel huésped sea modificado, de forma que sea consciente de estar corriendo en un entorno paravirtualizado. A la hora de ejecutar, el hipervisor marcará aquellas regiones del código que sean comprometidas, esto es, que deban correrse en modo kernel en la CPU, dejando al margen las partes en modo usuario que podrán ser ejecutadas como procesos del anfitrión. A continuación, el hipervisor gestionará un contrato de ejecución entre el huésped y el anfitrión, de forma que este último procesará esas partes en modo kernel como si se tratasen de secciones del dominio real de procesado, y no como pertenecientes al dominio virtual con su correspondiente penalización de rendimiento. La paravirtualización, en cambio, no requiere ninguna extensión especial en el hardware.

2.1.3 Frameworks para cloud IaaS

Los frameworks para cloud IaaS son aquellos sistemas software encargados de abstraer la complejidad asociada al aprovisionamiento dinámico y la administración de infraestructura genérica susceptible de fallar. Casi en su totalidad en código abierto bajo licencia Apache —lo que facilita las aportaciones y la

reutilización de módulos de unos a otros—, sin embargo han tenido marcos de evolución diferentes. Este hecho ha condicionado su falta de interoperabilidad hacia el exterior —con clientes consumidores del cloud— cuajando APIs de gestión no estandarizadas; aunque hoy por hoy empieza a no ser cierto. Estos frameworks no dejan de ser producto del esfuerzo por mejorar y facilitar la administración de los clusters concretos sobre los que se fueron gestando. Por tanto, no es de extrañar que sus avances se hayan producido en paralelo con la infraestructura que gestionaban, sin importar demasiado la compatibilidad.

Poco a poco, estos sistemas de manejo fueron creciendo en alcance y responsabilidad impulsados por el creciente interés del sector. Al final, sucedió que la propia ingeniería del software y de sistemas los fue haciendo cada vez más genéricos, de manera que terminaron por solaparse funcionalmente. La aparición de los AWS terminó de fraguar la necesidad latente de estandarización, y así, la mayoría de estos frameworks ofrecen APIs propios cada vez más próximos al de Amazon, hoy por hoy estándar de facto, y que además están siendo remodelados para dirigirse hacia el estándar OCCI [6].

2.2 Paradigma MapReduce

El origen del paradigma se centra en el artículo [2]. En él se investigaba la posibilidad de abstraer las partes comunes de la utilización de recursos computacionales distribuidos, que tanto aumentaban la complejidad de algunos problemas simples que se planteaban cotidianamente en Google. Así fue tomando forma la primera implementación de un modelo que se ocupaba de todos los apartados necesarios para aprovechar las capacidades de cálculo y almacenamiento de los clusters de que disponían. Una concisa definición dicta que MapReduce es un "*modelo de procesado de datos y entorno de ejecución que corre en grandes clusters de computadores personales convencionales*" [8].

2.2.1 Modelo de programación

El modelo del programación del paradigma requiere que el desarrollador exprese su problema como un contrato de procesado en dos partes fundamentales. La primera parte se encarga de leer los datos de entrada y de producir unos resultados intermedios que serán distribuidos por la red. Estos resultados intermedios se agruparán según el valor de una clave intermedia. Una segunda fase comienza con los resultados intermedios agrupados y finaliza cuando se han *reducido* todas las operaciones pendientes sobre las agrupaciones de entrada. Visto desde otra perspectiva, la primera fase se corresponde, a grandes rasgos, con el comportamiento de la función *map* y la segunda con la función *fold*; ambas pertenecientes al modelo de programación funcional.

En el vocabulario del framework MapReduce, estos conceptos de la programación funcional dan origen a las funciones `Map` y `Reduce` que definen los trabajos de procesado. Tanto Map como Reduce han de ser escritas por parte del programador, lo que puede obligar a cambiar el modo de expresión del problema original. A cambio, el framework se encarga de paralelizar la computación, distribuyendo los datos de entrada, manejando los errores que pudiesen producirse y recogiendo los resultados a la salida; todo de forma transparente.

Función Map

El típico map funcional toma una función F cualquiera y una lista de elementos L o, en general, cualquier estructura recursiva, para devolver la lista resultado de aplicar F a cada elemento de L. La figura 2.4 muestra su firma y un ejemplo de aplicación.

En su realización MapReduce, la función Map recibe un par de valores a la entrada y produce otro par `(clave, valor)` como salida intermedia. La librería MapReduce se encarga entonces de agrupar esos pares intermedios,

$$\textbf{map} : (\alpha \rightarrow \beta) \rightarrow \alpha\, list \rightarrow \beta\, list$$
$$\textbf{map}\ (\textbf{pow}\ 2)\ [1,2,3] \Rightarrow [1,4,9]$$

Figure 2.4: Ejemplo de aplicación de la función map (versión funcional)

$$\textbf{map} : (k1, v1) \rightarrow (k2, v2)\, list$$
$$\textbf{k} : clave$$
$$\textbf{v} : valor$$
$$(\textbf{kn}, \textbf{vn}) : par\ (clave, valor)\ en\ un\ dominio\ n$$

Figure 2.5: Firma de la función Map (versión MapReduce)

por valor de clave, antes de pasárselos a Reduce como entradas. Los tipos de entrada y salida de Map se corresponden con la firma de la figura 2.5. Es el framework MapReduce quien se encarga de alimentar la función Map, transformando los datos de los ficheros de entrada en pares `(clave, valor)` correctos.

Función Reduce

El típico fold de la programación funcional recoge una función G cualquiera, una lista L, genéricamente cualquier estructura recursiva, y un elemento inicial I, del mismo tipo que los elementos de L y sobre el que se acumulan los resultados intermedios, para devolver el valor final de I, resultado de aplicar G a cada elemento de L. La figura 2.6 presenta su firma y un ejemplo de aplicación.

$$\textbf{fold} : (\alpha \rightarrow \beta \rightarrow \alpha) \rightarrow \alpha \rightarrow \beta\, list \rightarrow \alpha$$
$$\textbf{fold}\ (+)\ 0\ [1,2,3] \Rightarrow 6$$

Figure 2.6: Ejemplo de aplicación de la función fold

reduce : $(k2, v2\,list) \rightarrow v2\,list$
k : *clave*
v : *valor*
$(\mathbf{kn}, \mathbf{vn})$: *par* $(clave, valor)$ *en un dominio* n

Figure 2.7: Firma de la función Reduce

A la función Reduce, en cambio, se le pasan las agrupaciones intermedias formadas por una clave y su conjunto de valores asociado, la salida de Map, para producir en su salida un conjunto menor de valores, ya que cada invocación de Reduce ocasionará cero o un único valor por cada clave, *reduciendo* o plegando (fold) esos valores intermedios. La firma de Reduce se presenta en la figura 2.7. Del mismo modo a como sucede en Map, es el framework quien se encarga de gestionar la transferencia de los resultados intermedios, salida de Map, a Reduce. A este respecto decir que algunas implementaciones del modelo permiten al desarrollador controlar el modo en que gestionan los resultados intermedios, pudiendo definir el comportamiento de las agrupaciones en una función `Combiner` que actúa antes de Reduce.

Wordcount en MapReduce

Como ejemplo, la figura 2.8 contiene el pseudocódigo de una aplicación MapReduce para contar el número de apariciones de distintas palabras en una serie de documentos.

En una ejecución de `wordcount` sucederá lo siguiente: a la entrada de Map se situarán tanto el nombre de cada documento como su contenido en texto plano. Map recorrerá entonces cada documento al completo emitiendo el par `(<palabra>, ‘‘1’’)` por cada palabra de cada documento. De este modo, se producirá una explosión de pares intermedios que representan ya la solución del problema; sólo resta agrupar los pares por palabra y sumar el número de ocurrencias de cada par —la función Reduce. Al Reduce de `wordcount`

```
Map (String key, String value):
// key:  nombre del documento
// value:  contenido del documento
for each word w in value:
EmitIntermediate (w, "1");

Reduce (String key, Iterator values):
// key:  una palabra
// values:  un iterable sobre un número de palabras
int result = 0;
for each v in values:
Emit (AsString (result));
```

Figure 2.8: Pseudocódigo de wordcount para MapReduce. Fuente: [2]

se le presentará, por cada palabra de cada documento, la lista, o en general el *iterable*, que contenga todos los valores asociados a esa palabra, es decir, una lista con tantos "1" como veces haya aparecido la palabra en todos los documentos. Al concluir la ejecución, MapReduce devolverá un listado con todas las palabras visitadas y el número de veces que aparece cada una de ellas.

2.2.2 Aplicabilidad del modelo

La diversidad de problemas que pueden expresarse usando este paradigma aparece reflejado en [2], por ejemplo:

- Grep distribuido: map emite cada línea que concuerde con el patrón introducido. Reduce sólo copia sus entradas a la salida (función *identidad*).
- Recuento de la frecuencia de acceso a un URL: muy parecido al wordcount.

- Gráfico inverso web-enlace: para cada URL destino contenido en una web origen, Map genera el par `(<URLdestino>, <URLorigen>)`
- Índice invertido: Map parsea cada documento y emite una serie de pares de la forma `(<palabra>, <iddocumento>)`. Todos ellos son pasados como entrada a Reduce que elabora la secuencia de pares `(<palabra>, list(iddocumento))`.

2.2.3 Modelo de procesado

Además de definir la estructura que han de seguir los programas que quieran apoyarse en la librería MapReduce, en vez de escribir sus propios mecanismos para el empleo de computación distribuida, con [2] se propuso una implementación del modelo que permitiese mejorar el modo de utilización de los recursos computacionales de que disponían en Google:

- Linux, doble procesador, 2-4 GB de RAM.
- 100 Mb/s ó 1 Gb/s de ancho de banda de red en las máquinas.
- Cientos o miles de máquinas por clúster, es decir, se producirán fallos de ejecución ineludiblemente.
- Discos duros IDE y sistema de ficheros distribuido manejando la información en ellos.

Así evitamos el uso de protocolos, arquitecturas, sistemas operativos o software de comunicación propietarios: ideal para despliegues genéricos.

Cada petición de ejecución MapReduce se denomina *trabajo*. Cada trabajo está compuesto por múltiples *tareas*. La finalización de un trabajo requiere la conclusión del procesamiento de todas las tareas. La máxima del modelo de procesado es paralelizar la ejecución de las tareas en cada nodo participante. De forma general, se puede afirmar que las tareas de cada fase —las tareas de la fase Map y las de la fase Reduce— se procesan en paralelo

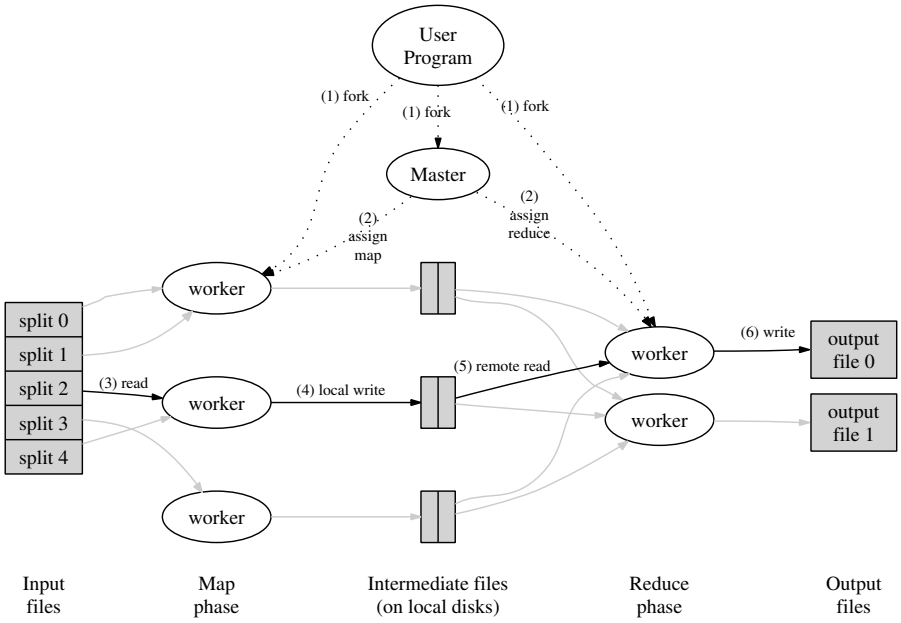

Figure 2.9: Diagrama de ejecución MapReduce. Fuente: [2]

y que las fases se suceden en serie. Sin embargo, no es necesario que haya terminado *completamente* la fase Map para que comience la Reduce.

La figura 2.9 muestra un resumen del flujo de ejecución típico de procesado MapReduce. Es interesante describirlo ya que los distintos frameworks que se adhieren al modelo MapReduce van a presentar aproximaciones muy similares.

1. MapReduce divide los ficheros de entrada en M partes, de tamaño configurable usando un parámetro, y distribuye tantas copias del programa del usuario —típicamente sólo las implementaciones concretas de Map y Reduce— como nodos participen en la computación.

2. Ahora cada copia del programa reside en un nodo. Se define entonces que una de esas copias sea especial —la réplica *Master*— de forma que el resto quedan residiendo en nodos que, en adelante, son considerados

trabajadores o *esclavos*. O dicho de otro modo, desde el momento en que se nombra la copia maestra, se determina el rol *maestro* del clúster, que recae sobre el nodo que posee la copia maestra; los demás nodos tendrán en su poder copias no maestras o esclavas, fijando, igualmente, su rol a esclavos. Estos nodos esclavos son los que reciben las tareas concretas de ejecución dirigidos por el maestro. Habrá M tareas Map y R tareas Reduce que el maestro ha de repartir entre sus trabajadores inactivos y que podrán ser procesadas en paralelo.

3. Los trabajadores a los que se les hayan asignado tareas Map leen las porciones de los ficheros de entrada correspondientes, *parsean* los documentos generando pares `(clave, valor)` que redirigen a la función Map del usuario. El contenido de salida de Map se mantiene en memoria a modo de búffer.

4. Periódicamente, esos pares en memoria son volcados a *disco local* y particionados en R regiones. Su posición en disco es enviada al maestro, responsable de informar de la localización de esos pares intermedios a los trabajadores con tareas Reduce.

5. En el momento en que un trabajador Reduce es notificado de que sus datos de ejecución, esto es, los datos de una partición que le corresponde, están disponibles, éste lee la información del disco del trabajador Map a través de *RPC* (*Remote Procedure Call*) y, antes de invocar la función Reduce del usuario, ordena los pares intermedios por clave.

6. Finalmente, el trabajador Reduce itera sobre los pares ordenados por clave, enviando, a la función Reduce del usuario, la clave y el conjunto de valores asociados a esa clave. La salida de la función Reduce para esa partición se va escribiendo en un archivo de salida sobre el sistema de ficheros distribuido.

Cuando se hayan completado todas las tareas Map y Reduce, el espacio particionado de salida —los ficheros de salida de cada partición— se envía al

programa que haya hecho la llamada a MapReduce. El modelo de ejecución es lo suficientemente genérico como para aplicarse a la resolución de problemas de distinto tamaño, corriendo sobre clusters indeterminadamente grandes.

2.2.4 Tolerancia a fallo

La idea de proporcionar un marco de ejecución de trabajos lo suficientemente grandes como para necesitar enormes conjuntos de máquinas de procesado para mantener los tiempos de latencia en valores razonables, pasa, forzosamente, por definir una política que asegure cierta resistencia a unos fallos que seguramente aparecerán. De no ser considerados, estos fallos derivarían en errores de distinto grado de severidad: unos provocarían que se perdiesen tareas ya completadas, otros, menos problemáticos, inhibirían los datos intermedios, siendo improbable la conclusión con éxito de la ejecución. Sea como fuere, el propio modelo MapReduce prevee una serie de problemas en el procesamiento e implementa una serie de actuaciones ante cada uno de ellos.

Fallo en trabajador

El menos problemático. Para controlar que todos los trabajadores al cargo de un maestro funcionan correctamente, cada cierto tiempo el maestro envía una petición de respuesta de eco —el clásico *ping*— a todos estos trabajadores. Si alguno de ellos no contestase de forma reiterada, éste quedaría registrado en el maestro como *inaccesible*.

Un trabajador marcado inaccesible no recibirá nuevas tareas, ni tampoco podrá ser accedido remotamente por los trabajadores Reduce para cargar los resultados de sus Map intermedios, lo que puede suponer un problema insalvable para completar la ejecución. El acceso a estos datos intermedios se resuelve marcando, de nuevo en el maestro, las tareas asignadas al trabajador caído como *inactivas*. Así, éstas serán programadas para ser procesadas de nuevo en algún trabajador activo, que colocará los resultados intermedios

en algún punto accesible por los nodos Reduce. Además, en el momento de detectar la incapacidad operativa de un trabajador, el maestro informará de este hecho al resto de trabajadores esclavos para evitar, por ejemplo, errores en la lectura de los datos intermedios.

Fallo en maestro

El fallo en el nodo maestro es el más problemático. La aproximación propuesta consiste en ordenar que el maestro cree periódicamente puntos de restauración desde los que pueda reanudar su ejecución en caso de que se terminase inesperadamente. Ya que el maestro es único, la probabilidad de fallo es mucho menor que para el caso general de fallo en un nodo; de hecho, lo suficientemente baja como para que se proponga en [2] que se aborte la ejecución del trabajo al completo si el maestro fallase. Como primera solución es aceptable. No obstante, dado que no tiene ningún sentido dejar un *punto de fallo singular* fácil de abordar al descubierto, frameworks posteriores proponen replicar el comportamiento del maestro en otros nodos.

2.2.5 Características adicionales

A continuación se resumen otras cuestiones adicionales estudiadas para este framework MapReduce.

Localidad

El cuello de botella en un despliegue típico es el ancho de banda de red. En una ejecución cualquiera, la información se propaga desde el cliente hacia el sistema de ficheros distribuido del clúster MapReduce. Como se ha comentado, cada nodo posee una cierta capacidad de almacenamiento y ejecución de tareas sobre los datos de entrada. De esta forma, cada paso del procesado global requiere la transmisión por la red de enormes cantidades de información —recordar, por ejemplo, la E/S de la función Map en wordcount— colapsando rápidamente su ancho de banda y limitando, por tanto, la veloci-

dad de ejecución. Para mitigar este problema, se afina el sistema de ficheros distribuido para que aproxime la información al nodo del clúster que ha de procesar esa información, reduciendo considerablemente las transferencias acometidas para completar los trabajos, minimizando así el tráfico de red requerido.

Complejidad

A priori, las variables M y R de las ejecuciones MapReduce —recordemos, la dimensión de las particiones del espacio de entrada e intermedio respectivamente— pueden tomar valores cualesquiera configurables por el usuario. Sin embargo, existen ciertos límites prácticos. Por cada trabajo MapReduce que se esté gestionando, el maestro ha de hacer $O(M + R)$ decisiones de planificación —suponiendo que no se produjese ningún error que forzase replanificar tareas— ya que cada partición del espacio de entrada, M, habrá de repartirse entre los trabajadores Map, y cada partición del espacio de salida de Map, R, entre los trabajadores Reduce; de ahí la expresión de la *complejidad temporal*. En cuanto a la *complejidad espacial*, el maestro tendrá que mantener $O(M \cdot R)$ de memoria de estado. La expresión se optiene al razonar que, en el peor de los casos, cada parte del fichero de entrada se mapeará sobre cada parte del espacio intermedio.

Tareas secundarias

Podría caber la situación en la que un nodo del clúster estuviese ejecutando tareas Map o Reduce a una velocidad más lenta de lo habitual; por ejemplo, si tuviese el disco rígido dañado, las lecturas y escrituras se sucederían más lentamente, y como el trabajo MapReduce no está completado hasta que todas sus tareas componentes hayan finalizado con éxito, el nodo problemático estaría ocasionando una limitación en el rendimiento global. Para aliviar el problema, se propone que cuando falte por concluir la ejecución de pocas tareas del trabajo, éstas sean enviadas a dos nodos trabajadores: una como tarea convencional y otra como tarea secundaria de algún nodo inactivo. En

el momento en que concluya la ejecución de alguna de las dos, esa tarea se marcará como terminada, reduciendo la latencia que añaden esos nodos con dificultades operativas.

Función Combiner

En muchas ocasiones sucede que existe un gran número de pares intermedios generados que se repiten. Tomando como ejemplo el wordcount expuesto antes, se puede deducir que cada trabajador Map generará ingentes cantidades de registros de la forma (‘‘a’’, ‘‘1’’), que serán enviados por la red hacia los trabajadores Reduce. Un mecanismo para reducir este colapso es permitir al usuario escribir una función de combinación de resultados intermedios, que se lanzará previa ejecución de la función Reduce. Esta función combinadora irá recogiendo los resultados intermedios, dentro del mismo trabajador Map, para hacer una primera reducción (o combinación) de los datos antes de ser marcados como disponibles para los trabajadores Reduce.

Típicamente, el código fuente de ambas funciones —Reduce y Combiner— es el mismo; lo que sí cambia es el modo en que la librería MapReduce gestiona su ejecución: la salida de la combinación se mantiene en disco local y no en el sistema de ficheros distribuido, para reducir la sobrecarga de red que supondría.

2.2.6 Frameworks MapReduce

Desde 2004 han ido apareciendo multitud de frameworks que codifican, considerando distintas optimizaciones, la funcionalidad del paradigma. Por citar algunos nombres:

Hadoop [8] Uno de los primeros en cubrir el modelo de procesado propuesto para MapReduce y principal implementación de referencia para los demás frameworks. Es el más utilizado, probado y desplegado con diferencia en la actualidad.

GridGain [9] Comercial y centrado en el procesado en memoria para acelerar la ejecución: menor latencia de acceso al dato a costa de un menor espacio de E/S.

Twister [10] Desarrollado como proyecto de investigación en la Universidad de Indiana, pretende dar un giro de tuerca más al modelo MapReduce, abstrayendo las partes comunes y necesarias para lanzar ejecuciones, aislándolas y almacenándolas en memoria distribuida como objetos estáticos, reduciendo así el tiempo necesario para configurar las funciones Map y Reduce previo procesado.

GATK [11] Utilizado en investifación genética para evaluar y secuenciar fragmentos de ADN de múltiples especies.

Qizmt [12] Escrito en C# y puesto en funcionamiento para MySpace.

misco [13] Escrito 100% en Python y basado en trabajos previos de Nokia, se propone como implementación MapReduce capaz de correr en dispositivos móviles.

Peregrine [14] Optimizando el modo en que se manipulan los resultados intermedios y pasando toda operación de E/S a una cola asíncrona, entre otras alteraciones, han llegado a acelerar bastante la ejecución de tareas. Todavía en beta a junio de 2013.

Mars [15] Implementado en *NVIDIA CUDA*, se centra en extraer el máximo rendimiento de mapeo y reducción moviendo la operativa a la tarjeta gráfica. En seis pruebas realizadas con Mars se han observado incrementos de rendimiento superiores a un orden de magnitud frente a ejecuciones en CPU.

De todos ellos Hadoop es, incuestionablemente, el framework MapReduce más utilizado en la actualidad. Su naturaleza de código abierto y su flexibilidad, tanto de procesado como de almacenamiento, le han reportado un creciente interés desde la industria de la IT, que se ha venido traduciendo en

la aparición de numerosas herramientas acoplables a Hadoop que extienden su funcionalidad.

Bibliography

[1] Google Apps: Energy Efficiency in the Cloud. `http://static.googleusercontent.com/external_content/untrusted_dlcp/www.google.com/en/us/green/pdf/google-apps.pdf`. Accedido: junio 2013.

[2] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified Data Processing on Large Clusters. *Common. ACM*, 51(1):107–113, 2008.

[3] Pierre Riteau, Ancuta Iordache, and Christine Morin. Resilin: Elastic MapReduce for Private and Community Clouds. Research Report RR-7767, INRIA, October 2011.

[4] Huan Liu and Dan Orban. Cloud MapReduce: A MapReduce Implementation on Top of a Cloud Operating System. In *Proceedings of the 2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, CCGRID '11, pages 464–474, Washington, DC, USA, 2011. IEEE Computer Society.

[5] Steve Loughran, Jose Maria Alcaraz Calero, Andrew Farrell, Johannes Kirschnick, and Julio Guijarro. Dynamic Cloud Deployment of a MapReduce Architecture. *IEEE Internet Computing*, 16(6):40–50, November 2012.

[6] Andy Edmonds, Thijs Metsch, Alexander Papaspyrou, and Alexis Richardson. Toward an Open Cloud Standard. *IEEE Internet Computing*, 16(4):15–25, July 2012.

[7] Rich Uhlig, Gil Neiger, Dion Rodgers, Amy L. Santoni, Fernando C. M. Martins, Andrew V. Anderson, Steven M. Bennett, Alain Kagi, Felix H. Leung, and Larry Smith. Intel Virtualization Technology. *Computer*, 38(5):48–56, May 2005.

[8] Tom White. *Hadoop, the Definitive Guide*. O' Reilly and Yahoo! Press, 2012.

[9] Nikita Ivanov. GridGain and Hadoop: Differences and Sinergies. `http://www.gridgain.com/blog/gridgain-hadoop-differences-synergies/`. Accedido: junio 2013.

[10] Jaliya Ekanayake, Hui Li, Bingjing Zhang, Thilina Gunarathne, Seung hee Bae, Judy Qiu, and Geoffrey Fox. Twister: A Runtime for Iterative MapReduce. In *The First International Workshop on MapReduce and its Applications*, 2010.

[11] Aaron McKenna, Matthew Hanna, Eric Banks, Andrey Sivachenko, Kristian Cibulskis, Andrew Kernytsky, Kiran Garimella, David Altshuler, Stacey Gabriel, Mark Daly, and Mark A. DePristo. The Genome Analysis Toolkit: A MapReduce Framework for Analyzing Next-generation DNA Sequencing Data. 2010.

[12] Quizmt project web page. `http://qizmt.myspace.com/`. Accedido: junio 2013.

[13] Adam Dou, Vana Kalogeraki, Dimitrios Gunopulos, Taneli Mielikainen, and Ville H. Tuulos. Misco: a Mapreduce Framework for Mobile Systems. In *Proceedings of the 3rd International Conference on Pervasive Technologies Related to Assistive Environments*, PETRA '10, pages 32:1–32:8, New York, NY, USA, 2010. ACM.

[14] Peregrine project web page. `http://peregrine_mapreduce.bitbucket.org/`. Accedido: junio 2013.

[15] Bingsheng He, Wenbin Fang, Qiong Luo, Naga K. Govindaraju, and Tuyong Wang. Mars: A MapReduce Framework on Graphics Processors. In *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, PACT '08, pages 260–269, New York, NY, USA, 2008. ACM.

[16] *Eucalyptus — 3.1.1 Installation Guide*, 2012.

[17] *CloudStack Basic Installation Guide for CloudStack Version 3.0.0 — 3.0.2*, 2012.

[18] Citrix Unveils Next Phase of Cloudstack Strategy. `http://www.citrix.com/news/announcements/apr-2012/citrix-unveils-next-phase-of-cloudstack-strategy.html`, 2012. Accedido: junio 2013.

[19] How to Use CloudStack without Hardware Virtualization. `http://support.citrix.com/article/CTX132015`, 2012. Accedido: junio 2013.

[20] *Apache CloudStack 4.0.0 — Incubating CloudStack Installation Guide*, 2012.

[21] *CloudStack Advanced Installation Guide for CloudStack Version 3.0.0 — 3.0.2*, 2012.

[22] *Citrix XenServer 6.0 Installation Guide*, 2012.

[23] OpenNebula 3.8.1 QuickStart. `http://wiki.centos.org/Cloud/OpenNebula/QuickStart`, 2012. Accedido: junio 2013.

[24] Getting Started with Openstack on Fedora 17. `http://fedoraproject.org/wiki/Getting_started_with_OpenStack_on_Fedora_17`, 2012. Accedido: junio 2013.

[25] *OpenStack Install and Deploy Manual — Red Hat — Folsom*, 2012.

[26] *OpenStack Network (Quantum) Administration Guide — Folsom*, 2012.

[27] *OpenStack Object Storage Administration Manual — Folsom*, 2012.

[28] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The Google File System. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, SOSP '03, pages 29–43, New York, NY, USA, 2003. ACM.

[29] *Amazon Elastic Compute Cloud — User Guide — API Version 2012-12-01*, 2013.

[30] QEMU Internals. `http://qemu.weilnetz.de/qemu-tech.html`, 2012. Accedido: junio 2013.

[31] Jaromír Hradílek, Douglas Silas, Martin Prpič, Stephen Wadeley, Eliška Slobodová, Tomáš Čapek, Petr Kovář, John Ha, David O'Brien, Michael Hideo, and Don Domingo. *Fedora 17 System Administrators' Guide. Deployment, Configuration and Administration of Fedora 17.* Red Hat Inc., 2012.

[32] Christopher Curran and Jan Mark Holzer. *Red Hat Enterprise Linux 5.2 – Virtualization Guide.* Red Hat Inc., 2008.

[33] Michael Hideo Smith. *Red Hat Enterprise Linux 5.2 — Deployment Guide.* Red Hat Inc., 2008.

[34] Johan De Gelas. Hardware Virtualization: the Nuts and Bolts. `http://www.anandtech.com/show/2480`, 2012. Accedido: junio 2013.

[35] *OpenStack Install and Deploy Manual — Red Hat — Essex*, 2012.

[36] *OpenStack Compute Administration Manual — Essex*, 2012.

[37] *OpenStack Compute Administration Manual — Folsom*, 2012.

[38] Jacek Artymiak. *Programming OpenStack Compute API.* Rackspace US Inc., 2012.

[39] *OpenStack API Reference*, 2013.

[40] OpenNebula OCCI Specification 3.8. `http://opennebula.org/documentation:archives:rel3.8:occidd`, 2012. Accedido: junio 2013.

[41] DEISA. Deisa glossary. `http://www.deisa.eu/references`. Accedido: junio 2013.

[42] Peter Mell and Timothy Grance. The NIST Definition of Cloud Computing. `http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf`, 2011. Accedido: junio 2013.

[43] *CloudStack API Documentation (v3.0)*, 2012.

[44] Simon Kelley. dnsmasq — A Lightweight DHCP and Caching DNS Server — ManPage. `http://www.thekelleys.org.uk/dnsmasq/docs/dnsmasq-man.html`, 2012. Accedido: junio 2013.

[45] HDFS Users' Guide. `http://hadoop.apache.org/docs/r1.0.4/single_node_setup.html`, 2010. Accedido: junio 2013.

[46] Single Node Setup. `http://hadoop.apache.org/docs/r1.0.4/single_node_setup.html`, 2008. Accedido: junio 2013.

[47] MapReduce Tutorial. `http://hadoop.apache.org/docs/r1.0.4/mapred_tutorial.html`, 2008. Accedido: junio 2013.

[48] Cluster setup. `http://hadoop.apache.org/docs/r1.0.4/hdfs_user_guide.html`, 2008. Accedido: junio 2013.

[49] Dhruba Borthakur. HDFS Architecture Guide. `http://hadoop.apache.org/docs/r1.0.4/hdfs_design.html`, 2012. Accedido: junio 2013.

[50] Jimmy Lin and Chris Dyer. *Data–Intensive Text Processing with MapReduce*. 2010.

[51] Sriram Rao, Raghu Ramakrishnan, Adam Silberstein, Mike Ovsiannikov, and Damian Reeves. Sailfish: A Framework for Large Scale Data Processing. In *Proceedings of the Third ACM Symposium on Cloud Computing*, SoCC '12, pages 4:1–4:14, New York, NY, USA, 2012. ACM.

[52] Ahmed Metwally and Christos Faloutsos. V-SMART-join: A Scalable MapReduce Framework for All-pair Similarity Joins of Multisets and Vectors. *Proc. VLDB Endow.*, 5(8):704–715, April 2012.

[53] Amr Awadallah. Apache Hadoop in the Enterprise — Keynote. Cloudera Inc., 2011.

[54] Mendel Cooper. Advanced Bash-Scripting Guide. An In-depth Exploration of the Art of Shell Scripting. `http://tldp.org/LDP/abs/html/`, 2012. Accedido: junio 2013.

[55] Bruce Barnett. Sed — An Introduction and Tutorial. `http://www.grymoire.com/Unix/Sed.html`, 2012. Accedido: junio 2013.

[56] MapReduce Wikipedia entry. `http://en.wikipedia.org/wiki/MapReduce`, 2012. Accedido: junio 2013.