INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Resilin: Elastic MapReduce for Private and Community Clouds

Pierre Riteau — Ancuţa Iordache — Christine Morin

N° 7767

Octobre 2011

Distributed and High Performance Computing

ISSN 0249-6399 ISRN INRIA/RR--7767--FR+ENG

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

INRIA

centre de recherche
RENNES - BRETAGNE ATLANTIQUE

Resilin: Elastic MapReduce for Private and Community Clouds

Pierre Riteau*† , Ancuţa Iordache‡ , Christine Morin†

Theme : Distributed and High Performance Computing
Équipe-Projet Myriads

Rapport de recherche n° 7767 — Octobre 2011 — 18 pages

Abstract: The MapReduce programming model, introduced by Google, offers a simple and efficient way of performing distributed computation over large data sets. Although Google's implementation is proprietary, MapReduce can be leveraged by anyone using the free and open source Apache Hadoop framework. To simplify the usage of Hadoop in the cloud, Amazon Web Services offers Elastic MapReduce, a web service enabling users to run MapReduce jobs. Elastic MapReduce takes care of resource provisioning, Hadoop configuration and performance tuning, data staging, fault tolerance, etc. This service drastically reduces the entry barrier to perform MapReduce computations in the cloud, allowing users to concentrate on the problem to solve. However, Elastic MapReduce is restricted to Amazon EC2 resources, and is provided at an additional cost. In this paper, we present Resilin, a system implementing the Elastic MapReduce API with resources from other clouds than Amazon EC2, such as private and community clouds. Furthermore, we explore a feature going beyond the current Amazon Elastic MapReduce offering: performing MapReduce computations over multiple distributed clouds.

Key-words: Cloud computing, MapReduce, Elasticity, Hadoop, Execution platforms

* Université de Rennes 1, IRISA, Rennes, France – Pierre.Riteau@irisa.fr
† INRIA Rennes – Bretagne Atlantique, Rennes, France – firstname.lastname@inria.fr
‡ West University of Timişoara, Timişoara, Romania – ancuta.iordache@info.uvt.ro

Centre de recherche INRIA Rennes – Bretagne Atlantique
IRISA, Campus universitaire de Beaulieu, 35042 Rennes Cedex
Téléphone : +33 2 99 84 71 00 — Télécopie : +33 2 99 84 71 71

Resilin: Elastic MapReduce pour nuages informatiques privés et communautaires

Résumé : Le modèle de programmation MapReduce, introduit par Google, offre un moyen simple et efficace de réaliser des calculs distribués sur de large quantités de données. Bien que la mise en œuvre de Google soit propriétaire, MapReduce peut être utilisé librement en utilisant le framework Hadoop. Pour simplifier l'utilisation de Hadoop dans les nuages informatiques, Amazon Web Services offre Elastic MapReduce, un service web qui permet aux utilisateurs d'exécuter des travaux MapReduce. Il prend en charge l'allocation de ressources, la configuration et l'optimisation de Hadoop, la copie des données, la tolérance aux fautes, etc. Ce service rend plus accessible l'exécution de calculs MapReduce dans les nuages informatiques, permettant aux utilisateurs de se concentrer sur la résolution de leur problème plutôt que sur la gestion de leur plate-forme. Cependant, Elastic MapReduce est limité à l'utilisation de ressources de Amazon EC2, et est proposé à un coût additionnel. Dans cet article, nous présentons Resilin, un système mettant en œuvre l'API Elastic MapReduce avec des ressources provenant d'autres nuages informatiques que Amazon EC2, tels que les nuages privés ou communautaires. De plus, nous explorons une fonctionnalité additionnelle comparé à Amazon Elastic MapReduce: l'exécution de calculs MapReduce sur plusieurs nuages distribués.

Mots-clés : Informatique en nuage, MapReduce, élasticité, Hadoop, plates-formes d'exécution

1 Introduction

The MapReduce programming model [15], proposed by Google in 2004, offers a simple way of performing distributed computation over large data sets. Users provide a map and a reduce function. The map function takes a set of input key/value pairs, and produces intermediate key/value pairs. The reduce function merges intermediate key/value pairs together to produce the result of the computation. This programming model became popular because it is simple yet expressive enough to perform a large variety of computing tasks. It can be applied to many fields, from data mining to scientific computing. This programming model is backed by a proprietary framework developed by Google that takes care of scheduling tasks to workers, sharing data through a distributed file system, handling faults, etc.

The Apache Hadoop [1] project develops a free and open source implementation of the MapReduce framework. The Hadoop MapReduce framework works with the Hadoop Distributed File System (HDFS) [32], designed to be highly-reliable and to provide high throughput for large data sets used by applications. Hadoop is heavily used by companies such as Yahoo!, Facebook and eBay to perform thousands of computations per day over petabytes of data [35, 33]. However, managing a Hadoop cluster requires expertise, especially when scaling to a large number of machines. Moreover, users who want to perform MapReduce computations in cloud computing environments need to instantiate and manage virtual resources, which further complicates the process.

To lower the entry barrier for performing MapReduce computations in the cloud, Amazon Web Services provides Elastic MapReduce (EMR) [30]. Elastic MapReduce is a web service to which users submit MapReduce jobs. The service takes care of provisioning resources, configuring and tuning Hadoop, staging data, monitoring job execution, instantiating new virtual machines in case of failure, etc.

However, this service has a number of limitations. First, it is restricted to Amazon EC2 resources. Users are not able to use Elastic MapReduce with resources from other public clouds or from private clouds, which may be less expensive or even free of charge. This is especially true for scientists who have access to community clouds administrated by their institution and dedicated to scientific computing [21, 27]. Moreover, Elastic MapReduce is provided for an hourly fee, in addition to the cost of EC2 resources. This fee ranges from 17% to 21% of the price of on-demand EC2 resources. It is impossible to use a different virtual machine image than the one provided by Amazon, which is based on Debian Lenny 5.0.8. Finally, some users may have to comply with data regulation rules, forbidding them from sharing data with external entities like Amazon.

In this paper, we present Resilin, a system implementing the Elastic MapReduce API with resources from other clouds than Amazon EC2, such as private and community clouds. Resilin allows users to perform MapReduce computations on other infrastructures than Amazon EC2, and offers more flexibility: users are free to select different types of virtual machines, different operating systems or newer Hadoop versions. The goal of Resilin is not only to be compatible with the Elastic MapReduce API. We also explore a feature going beyond the current Amazon Elastic MapReduce offering: performing MapReduce computations over multiple distributed clouds.

This paper is organized as follows. Section 2 presents the Amazon Elastic MapReduce service in more details. Section 3 covers the architecture and implementation of Resilin. Section 4 presents our experiments and analyzes their results. Section 5 reviews related work. Finally, Section 6 concludes and discusses future work.

2 Amazon Elastic MapReduce

Amazon Elastic MapReduce [30] is one of the products of Amazon Web Services. After signing up and providing a credit card number, users can submit MapReduce jobs through the AWS management console (a web interface), through a command line tool, or by directly calling the Elastic MapReduce API (libraries to access this API are available for several programming languages, such as Java and Python).

Users execute Hadoop jobs with Elastic MapReduce by submitting job flows, which are sequences of Hadoop computations. Before starting the execution of a job flow, Elastic MapReduce provisions a Hadoop cluster from Amazon EC2 instances. Each job flow is mapped to a dedicated Hadoop cluster: there is no sharing of resources between job flows. A Hadoop cluster created by Elastic MapReduce can be composed of three kinds of nodes:

- the master node, which is unique, acts as a meta data server for HDFS (by running the NameNode service) and schedules MapReduce tasks on other nodes (by running the JobTracker service),
- core nodes provide data storage to HDFS (by running the DataNode service) and execute map and reduce tasks (by running the TaskTracker service),
- task nodes execute map and reduce tasks but do not store any data.

By default, a Hadoop cluster created by Elastic MapReduce is composed of one master node and several core nodes[1]. At any time during the computation, a Hadoop cluster can be resized. New core nodes can be added to the cluster, but core nodes cannot be removed from a running job flow, since removing them could lead to HDFS data loss. Task nodes can be added at any time, and removed without risk of losing data: only the progress of MapReduce tasks running on the terminated task nodes will be lost, and the fault tolerance capabilities of Hadoop will trigger their re-execution on other alive nodes.

After the Hadoop cluster has booted, the service executes bootstrap actions, which are scripts specified by users. These actions allow users to customize an Elastic MapReduce cluster to some extent. Amazon provides several predefined bootstrap actions, to modify settings such as the JVM heap size and garbage collection behavior, the number of map and reduce tasks to execute in parallel on each node, etc. Users can also provide custom bootstrap actions by writing scripts and uploading them in S3. Once a Hadoop cluster is ready for computation, Elastic MapReduce starts executing the associated job flow. A job flow contains a sequence of steps, which are executed in order. Internally, each

[1] If a user requests a Hadoop cluster composed of only one node, the same virtual machine performs the roles of master node and core node.

step corresponds to the execution of a Hadoop program (which can itself spawn multiple Hadoop jobs). After all steps are executed, the cluster is normally shut down. Users can ask for the cluster to be kept alive, which is useful for debugging job flows, or for adding new steps in a job flow without waiting for cluster provisioning and paying extra usage (in Amazon EC2, instance cost is rounded up to the hour, which means a 1-minute and a 59-minute job flow cost the same price).

Typically, input data is fetched from Amazon S3, a highly scalable and durable storage system provided by Amazon. Intermediate data is stored in HDFS, the Hadoop Distributed File System. Output data is also saved in Amazon S3, since the termination of a Hadoop cluster causes its HDFS file system to be destroyed and all the data it contains to become unavailable.

3 Architecture and Implementation

Like the Amazon Elastic MapReduce service, Resilin provides a web service acting as an abstraction layer between users and Infrastructure as a Service (IaaS) clouds. Figure 1 presents how Resilin interacts with other components of a cloud infrastructure. Users execute client programs supporting the Elastic MapReduce API to interact with Resilin (for instance, a Python script using the boto [2] library). Resilin receives Elastic MapReduce requests and translates them in two types of actions:

- interactions with an IaaS cloud using the EC2 API, to create and terminate virtual machine instances,
- remote connections to virtual machines using SSH, to configure Hadoop clusters and execute Hadoop programs.

To retrieve input data and store output data, Hadoop clusters interact with a cloud storage repository through the S3 API.

Resilin implements most actions supported by the Amazon Elastic MapReduce API. Users are able to execute the following actions:

RunJobFlow Submits a job flow.

DescribeJobFlows Queries job flow statuses.

AddJobFlowSteps Adds new steps to an existing job flow.

AddInstanceGroups/ModifyInstanceGroups Change the number of resources by adding or modifying instances groups.

TerminateJobFlows Terminates job flows.

However, only JAR-based and streaming Hadoop jobs are currently available. Job flows written using Hive and Pig, two SQL-like languages for data analysis, are not yet supported by Resilin.

We now describe how each type of Elastic MapReduce action is handled. When a new job flow request (RunJobFlow) is received by Resilin, its parameters are validated, and the service contacts an EC2-compatible IaaS cloud on behalf of the user to provision a Hadoop cluster. By leveraging the EC2 API,

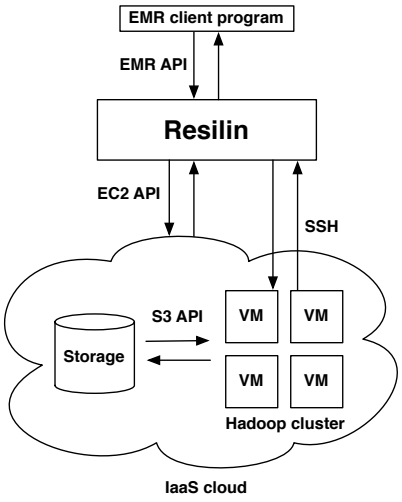

Figure 1: Overview of Resilin and its interactions with other cloud components

Resilin can make use of resources from clouds managed by open source toolkits supporting the EC2 API, such as Nimbus [7, 19], OpenNebula [8, 34], and Eucalyptus [3, 26].

Once a Hadoop cluster is provisioned, Resilin connects to the virtual machines, using SSH, to execute bootstrap actions specified by the user, configure the Hadoop cluster (specifying the HDFS NameNode address, the MapReduce JobTracker address, etc.), and start the Hadoop daemons.

After Hadoop has been configured, the associated job flow is started and each step in the job flow is executed by invoking the `hadoop` command on the master node. Each step specifies the locations of its input and output data. These references can be S3 URLs: Hadoop support reading and writing directly to Amazon S3. It would be technically possible to configure the Hadoop clusters created by Resilin to access input data from Amazon S3, like in Amazon Elastic MapReduce. However, this is ineffective for two reasons. First, bandwidth limitations between Amazon S3 and the private or community IaaS cloud running the Hadoop cluster drastically limit performance. Second, outgoing traffic from Amazon S3 is charged to customers, which would incur a high cost when performing MapReduce computations on large data sets.

We assume that clouds used by Resilin will have a storage service available. Both Nimbus and Eucalyptus provide their own S3-compatible cloud storage (respectively called Cumulus [11] and Walrus [6]). Furthermore, Cumulus can be installed as a standalone system, making it available for any cloud deployment. Resilin can use this kind of repository to provide Hadoop with input data and to store output data. To make it possible, we extended the S3 protocol support of Hadoop 0.20.2. In addition to being able to specify URLs in the form of `s3n://bucket/key`, users can also provide URLs such as `cumulus://bucket.host:port/key`. When such URLs are detected, the library used by Hadoop to interact with S3 (JetS3t) is set up to contact the Cumulus server running on `host:port`. Additionally, we had to implement support for

partial GET requests (HTTP Range header) in Cumulus. Hadoop uses this type of HTTP request to download subsets of input files on each node.

Two types of steps are supported by Resilin. The first type, a custom JAR, is simply the location of a JAR and its required arguments. When executed, this JAR submits jobs to Hadoop. The JAR URL can be a S3 or Cumulus location. In this case, Resilin first downloads the JAR to the master node, using the `hadoop fs -copyToLocal` command, as Hadoop does not support directly executing a JAR from a remote location. The second type, streaming jobs, is defined by a mapper program, a reducer program, and the locations of input and output data. Both programs can be stored in S3 or Cumulus, and can be written in any programming language (Java, Ruby, Python, etc.): they apply their computation on data coming from standard input and stream their result to standard output. To run a streaming step, the Hadoop command is invoked with the hadoop-streaming JAR included in the Hadoop MapReduce framework. We determined that Amazon made modifications to Hadoop to be able to fetch the mapper and reducer programs from S3 and add them to the Hadoop distributed cache (in order to provide them to all nodes). We did not make such modifications to the Hadoop source code, and currently rely on bootstrap actions to download the mapper and reducer programs to the master node.

Resilin monitors the execution of the Hadoop computation. When the execution of a step is finished, the status of the job flow is updated, and the next step starts running. When all steps have been executed, the job flow is finished, which triggers termination of the cluster (unless the user requested the cluster to be kept alive).

Resilin is implemented in Python. It uses the Twisted [10] framework for receiving and replying to HTTP requests, the boto [2] library to interact with clouds using the EC2 API, and the paramiko [9] library for executing commands on virtual machines using SSH.

3.1 Multi-Cloud Job Flows

Besides targeting compatibility with the Amazon Elastic MapReduce API, we are experimenting with an additional feature in Resilin: the ability to execute job flows with resources originating from multiple clouds. With Amazon Elastic MapReduce, executing MapReduce computations across multiple Amazon EC2 regions does not present a lot of interest. The large number of resources available in each region makes it possible to deploy large Hadoop clusters in a single data center[2]. However, in the usage context of Resilin, many users would have access to several moderately sized private or community clouds. For example, the FutureGrid [4] project, a distributed, high-performance testbed in the USA, contains 4 scientific clouds based on Nimbus, each having from 120 to 328 processor cores. Federating computing power from several such clouds to create large scale execution platforms has been proposed as the *sky computing* approach [20].

[2]When scaling to very large clusters, Amazon EC2 can start running out of resources. In March 2011, the Cycle Computing company published details about how they provisioned a 4096-core cluster in the Amazon EC2 US East region [14]. When they scaled up the number of resources, 3 out of the 4 availability zones of the region reported Out of Capacity errors.

Although creating MapReduce clusters distributed over several clouds may not be efficient because of the large amounts of wide area data transfer that it generates [12], it is interesting for some types of MapReduce jobs. For example, Matsunaga et al. [24] have shown that multi-cloud BLAST computations with MapReduce can scale almost linearly.

Resilin supports multi-cloud job flows by allowing users to dynamically add new resources from different clouds to a running job flow. As an example, let us assume that a user has access to two different clouds, *Cloud A* and *Cloud B*, and that *Cloud A* is her default choice. After a job flow has been started on *Cloud A*, Resilin will accept requests for adding new instances (AddInstanceGroup) with an instance type also specifying the cloud to use: instead of `m1.small`, the instance type of the request would be `m1.small@CloudB`. After new virtual machines are provisioned from *Cloud B*, they are configured to become resources of the cluster running in *Cloud A*. This addition is managed seamlessly by Hadoop. When Hadoop daemons are started in the newly provisioned virtual machines of *Cloud B*, they contact the master node in *Cloud A* to join the cluster and start receiving MapReduce tasks to execute.

4 Evaluation

For validating our implementation, we compare executions of the same job flows on Amazon Elastic MapReduce, using Amazon EC2 resources, and on Resilin, using resources provisioned from a cloud deployed on the Grid'5000 experimental testbed with Nimbus Infrastructure version 2.8.

In Amazon EC2, we used High-CPU Medium instances (c1.medium) from the US East (Northern Virginia) region. These instances are supplied with 2 virtual cores whose speed is equivalent to 2.5 EC2 Compute Units each (one EC2 Compute Unit corresponds to a 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor), 1.7 GB of memory, 350 GB of instance storage, and are running 32-bit operating systems. As mentioned in other performance studies of Amazon EC2 [17], the same type of instances can be hosted on physical machines with different processors. By examining the content of `/proc/cpuinfo`, we determined that the physical processors of our c1.medium instances were either Intel Xeon E5410 running at 2.33 GHz or Intel Xeon E5506 running at 2.13 GHz.

On Grid'5000, we used physical machines part of the *genepi* cluster in Grenoble. These machines have 2 Intel Xeon E5420 QC processors (each with 4 cores running at 2.5 GHz), 8 GB of memory, and a Gigabit Ethernet connectivity. They were running a 64-bit Debian Lenny 5.0.4 operating system with a Linux 2.6.32 kernel, and were using QEMU/KVM version 0.12.5 as hypervisor. The virtual machines created on these physical machines were dual-core virtual machines, with 2 GB of memory, and were running a 32-bit Debian Squeeze 6.0.1 with a Linux 2.6.32 kernel.

We evaluated the respective performance of each virtual machine type by running the benchmark tests of `mprime` [5]. Table 1 reports the mean, standard deviation, minimum and maximum of the average time required for computing a single thread 8192K FFT on each type of virtual machine. The results were obtained by running 5 iterations of the `mprime` benchmark on 3 instances of each virtual machine type, for a total of 30 runs. In the case of the c1.medium

Instance type	c1.medium	Nimbus VM
Mean (ms)	206.587	177.241 (14.2% faster)
Standard deviation	6.247	0.181
Minimum (ms)	197.998	176.821 (10.7% faster)
Maximum (ms)	220.829	177.55 (19.6% faster)

Table 1: CPU performance of the virtual machine types used in our experiments, measured with `mprime` (single-thread 8192K FFT)

virtual machines, 2 instances had a 2.33 GHz processor and 1 had a 2.13 GHz processor.

These measurements show the lack of performance predictability of Amazon EC2 instances, already reported in other studies [17]. Not only did we observed different performance between instances of the same type, but we also experienced substantial performance variability within the same instance. For example, one instance performed 2 consecutive runs of `mprime` with the following results for the 8192K FFT: 220.829 ms and 199.572 ms, a performance loss of 9.626%, even though these benchmarks runs were executed seconds apart. This behavior is likely caused by other instances sharing the same physical machine, which produced noise in the system and influenced the performance of our own instances.

Contrarily to Amazon EC2, the VMs executing in a Nimbus cloud on Grid'5000 showed stable performance measurements, both within one VM and among multiple VMs. When performing these benchmark runs, each VM was executed on a dedicated physical machine, which minimized the noise produced in the system.

To compare Resilin with Amazon Elastic MapReduce, we separate the evaluation of a job flow execution in two parts. First, we compare the deployment time, which includes provisioning a virtualized Hadoop cluster and performing its configuration. Second, we compare execution time, from the submission of a Hadoop computation until its completion. These experiments are run with various cluster sizes. In each experiment, we had a dedicated master node. We used the following numbers of core nodes: 1, 2, 4, and 8. Thus, the total number of instances were 2, 3, 5, and 9.

Additionally, we perform a synthetic benchmark of the cloud storage repository by measuring its read performance. The cloud storage repository is an important part of the system, as it is used to store input and output data. Finally, we report the cost of the instances used on Amazon Elastic MapReduce.

4.1 Deployment Time

The deployment time corresponds to the time taken from the submission of a RunJobFlow request to the API until the Hadoop cluster is ready to start executing an application. This includes provisioning a virtualized Hadoop cluster and performing its configuration. The provisioning of the cluster contains two main phases: propagating the virtual machine images to the hypervisor nodes and starting the virtual machines. The deployment time of a Hadoop cluster is independent of the application submitted for execution.

To compare the deployment performance of Elastic MapReduce and Resilin, we submit job flows on both services and compare the time from the submission

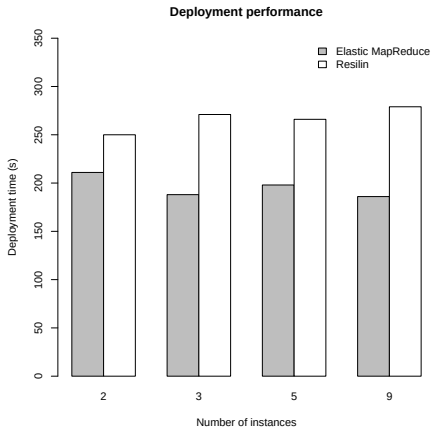

Figure 2: Deployment time of Hadoop clusters on Amazon Elastic MapReduce and Resilin with different number of instances

of the job flow until the start of the Hadoop computation. This time is computed as the difference between the CreationDateTime and the ReadyDateTime of the job flow, as reported by the DescribeJobFlows API request.

In the case of Amazon, the EC2 instances have a 5 GB root file system (the instances also have 350 GB of storage from local disks). However, since details of the Amazon EC2 architecture are not available, we cannot know how much of this data is propagated on the network, or if hypervisors have a local cache of popular virtual machine images.

In the case of the Nimbus cloud, the deployment time is mostly dependant of the propagation of virtual machine images to the hypervisor nodes. The size of our virtual machine image is 4 GB. LANTorrent is used for propagating these images to hypervisors. LANTorrent is a file distribution protocol integrated into the Nimbus IaaS toolkit. It is optimized to propagate the same virtual machine images from a central repository across a LAN to many hypervisor nodes.

Figure 2 shows the deployment time for different cluster sizes. Deployment time in Amazon EC2 is fairly stable, even though it can be perturbed by other users of the infrastructure, like the CPU performance presented earlier. Resilin also presents a stable deployment time. This can be explained by the efficient propagation mechanism used by LANTorrent, which scales almost linearly. However, Resilin is constantly slower than Elastic MapReduce by approximately one minute. We believe that the deployment process of Resilin can be improved by several techniques:

- propagation performance can be improved by optimizing the size of the virtual machine image, or by using compression or caching,
- configuration performance can be improved by reducing the number of SSH connections to the virtual machines (Resilin currently does one connection per configuration action, while all actions could be done by executing a single script).

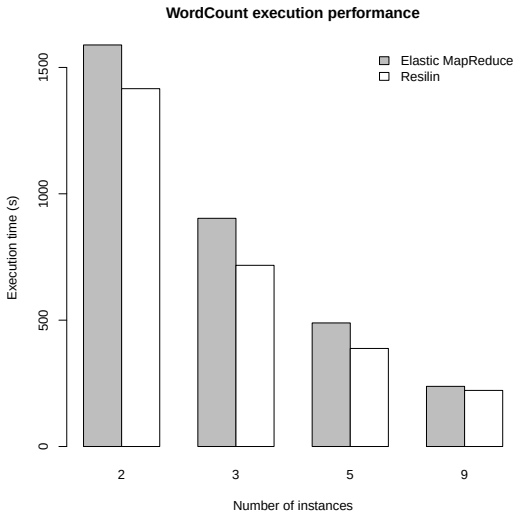

Figure 3: Execution time of Word Count on Amazon Elastic MapReduce and Resilin with different number of instances

4.2 Execution Time

To compare the execution performance of Elastic MapReduce and Resilin, we submit the same job flows, each composed of only one step, on both services and compare the time from the start of the requested Hadoop computation until its completion. This time is computed as the difference between the StartDateTime and the EndDateTime of the step, as reported by the DescribeJobFlows API request. We evaluate 2 different applications: Word Count and CloudBurst. We evaluated Resilin with Hadoop settings similar to those used by Elastic MapReduce. For instance, the number of map and reduce tasks were chosen to be the same: 4 map tasks and 2 reduce tasks per instance.

4.2.1 Word Count

To evaluate the performance of a streaming job, we ran a Word Count program on a collection of text files. Word Count computes the number of occurrences of each word in a text collection. The Amazon Elastic MapReduce Word Count sample [31] uses a copy of *The World Factbook* as input data. This book is split in 12 files, for a total of 18 MB. However, the content is not split evenly: the last 3 files are small in comparison to the first 9 files which contain most of the data. This text collection is small: Hadoop was created to process data at a much larger scale. We copied 100 times the first 9 files to create a larger collection of 900 files, for a total of 1.8 GB.

Figure 3 presents the results of the execution time of Word Count. For all cluster sizes, Resilin is faster than Amazon Elastic MapReduce. This is not surprising, since we determined that the virtual machines used on Grid'5000 had higher CPU performance. As the cluster size is increased, the difference of performance between Elastic MapReduce and Resilin becomes smaller. We will analyze if this can be explained by contention to access the Cumulus server.

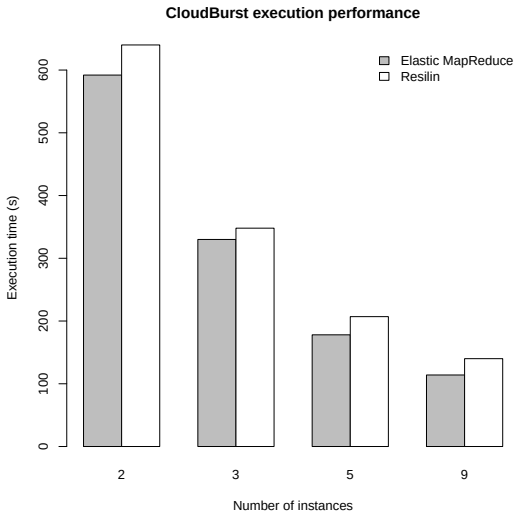

Figure 4: Execution time of CloudBurst on Amazon Elastic MapReduce and Resilin with different number of instances

4.2.2 CloudBurst

CloudBurst [28] is one of the sample workloads provided by Amazon for Elastic MapReduce. It implements a parallel read-mapping algorithm optimized for mapping next-generation sequence data to the human genome and other reference genomes. It is invoked as a custom JAR (`cloudburst.jar`), with the locations of input and output data and parameters for the execution (including the number of map and reduce tasks). We use the exact same program, input data and parameters as those provided by Amazon as a sample application (240 maps and 48 reduces). All input and output data is stored in the cloud storage repository.

Figure 4 presents the results of the execution time of CloudBurst. For this application, Resilin is consistently slower than Elastic MapReduce. Since Amazon uses a modified version of Hadoop with many tuned parameters, we will analyze if this difference in the execution environment is the source of the performance.

4.3 Cloud Storage Performance

In this experiment, we compare the performance of the cloud storage repository used in Amazon Elastic MapReduce and Resilin. We created a 1.8 GB file, uploaded it to S3 and to a Cumulus repository in Grid'5000, and compared the time required to read its content from Hadoop. The read operation was performed with the `fs -cat` option of the `hadoop` command using the `s3n` protocol. Output was sent to `/dev/null` to avoid any slowdown from I/O operations on the local hard drive.

Figure 5 compares the performance results obtained in Elastic MapReduce with one client and in Resilin (with Cumulus acting as a storage repository),

Number of instances	2	3	5	9
Instance cost (per hour)	$0.34	$0.51	$0.85	$1.53
Elastic MapReduce cost (per hour)	$0.06	$0.09	$0.15	$0.27
Total cost (per hour)	$0.40	$0.60	$1	$1.8

Table 2: Per-hour financial cost of Elastic MapReduce computations on Amazon with c1.medium instances in the US East region (not including S3 cost)

first with one client and then with 2 parallel clients. Results were obtained by executing 10 iterations of the read process. The plot reports the minimum, lower quartile, median, upper quartile, and maximum values observed. As with CPU benchmarks, we observed important performance variations when reading data stored in Amazon S3 from an EC2 instance. The slowest read operation (4.2 MB/s) took more than 6 times longer than the fastest one (26.4 MB/s), while the mean bandwidth of the 10 iterations was 10.14 MB/s. When scaling up the number of parallel readers, we did not notice any strong change of performance behavior, as Amazon S3 is presumably backed by thousands of machines serving data to clients, with data replicated in several locations.

With Resilin and the Cumulus storage repository, reading the file from one client at a time lead to a very stable bandwidth averaging 83.37 MB/s. Contrarily to Amazon S3, the Cumulus server is implemented with a single machine, which becomes a bottleneck when more clients are accessing the cloud storage in parallel. Figure 5 shows that for 2 parallel readers, the per-client bandwidth is reduced. This result was expected, since the resources of the Cumulus server are shared between the 2 clients. A notable behavior of Cumulus is that the bandwidth was not shared fairly between the 2 clients. While one client was averaging a bandwidth of 24.76 MB/s, the other was receiving data at 51.42 MB/s. Also, the cumulative bandwidth with 2 clients (76.18 MB/s) shows some performance degradation compared to the one-client scenario.

If more than 2 clients are doing parallel requests to a single Cumulus server, the performance of each client will degrade further. We note that Cumulus can be configured to scale horizontally across many nodes [11] by using parallel file systems such as GPFS. Using this setup, it should be possible to retain a good level of performance even when increasing the number of Cumulus clients. However, we did not test such a configuration in our experiments.

4.4 Financial Cost

An important difference between a public cloud like Amazon Web Services and a private or community cloud is the charging model. A public cloud directly charges its customers for resources used, while a private or community cloud is paid for by organizations (companies, universities, laboratories, etc.) and is free of charge for its direct users.

In Table 2, we report the price of the Elastic MapReduce clusters used in our experiments. The price is composed of the cost of the Amazon EC2 instances plus the cost of the Elastic MapReduce service. In our case, the EC2 instances were c1.medium instances of the US East region, billed at $0.17 per hour. For this type of instance, the Elastic MapReduce service costs $0.03 per instance

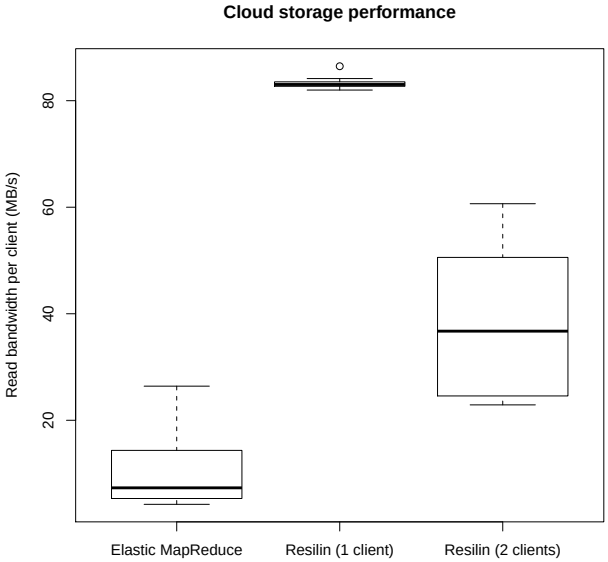

Figure 5: Comparison of cloud storage performance between Amazon Elastic MapReduce and Resilin (with the Cumulus cloud storage repository)

per hour. We note that each partial hour is fully billed: a job flow executing for 4 minutes costs a full hour.

The prices reported in Table 2 do not include the cost of the Amazon S3 storage. Amazon charges for three types of resource consumption in S3. Users are charged for the amount of storage space they use, for the number of requests to the S3 service, and for outgoing data transfer. There is no charge for data transferred between S3 and EC2 in the same region. In the case of Elastic MapReduce, this is generally the case: users provision instances in the data center where their data is available. However, for users with very large data sets, the storage space consumption and the number of requests could lead to significant costs. For instance, storing 1 PB of data in the US Standard Region of S3 costs more than $100,000 per month.

We cannot compare the cost of Elastic MapReduce computations with the real cost of resources of a private or community cloud, since the latter depends on many variables, including the costs of hardware resources, power, maintenance contracts, administrative staff, Internet network connectivity, etc.

5 Related Work

Related work on running MapReduce computations in the cloud has been focused on adapting MapReduce to cloud characteristics, either by creating completely new implementations of MapReduce, or by modifying existing systems. For instance, Gunarathne et al. [16] proposed AzureMapReduce, a MapReduce implementation built on top of Microsoft Azure cloud services [25]. They lever-

age several services offered by Azure, such as Azure Queues for scheduling tasks, Azure blob storage for storing data, etc. This allows their implementation to be decentralized and more elastic than a Hadoop-based cluster, leading to increased levels of fault-tolerance and flexibility. Similarly, Liu and Orban [23] created Cloud MapReduce, a MapReduce implementation leveraging services from the Amazon Web Services platform, with data stored in Amazon S3, and synchronization and coordination of workers performed with Amazon SQS and Amazon SimpleDB. These contributions are orthogonal to our objectives. While their goal is to build MapReduce implementations taking advantage of features offered by specific cloud services, we aim to bring an easy to use MapReduce execution platform to many different cloud implementations. Furthermore, to our knowledge, no open source cloud implements the Azure API or the SQS and SimpleDB interfaces, making it impossible to run these MapReduce implementations outside of Microsoft Azure and Amazon Web Services.

Another type of proposition has been to take advantage of spot instances available in Amazon EC2 [29]. Spot instances are virtual machines that leverage unused capacity of the Amazon EC2 infrastructure. The price of spot instances varies over time, depending on the infrastructure load. Users bid for a maximum price they are willing to pay for spot instances. As long as the current spot instance price stays lower than their bid, they are charged with the current spot price. When the spot instance price rises above their bid, their instances are terminated. Chohan et al. [13] propose using spot instances to reduce the cost of execution of MapReduce jobs. They study bidding strategies to optimize the effectiveness of using spot instances. Liu [22] uses spot instances in the Cloud MapReduce system, showing its efficiency as it handles instance terminations more gracefully than a Hadoop cluster. On August 18, 2011, Amazon unveiled spot instance support for Elastic MapReduce. However, no automatic bidding strategy or extended fault-tolerance is provided: customers are responsible for specifying how they want to use spot instances, knowing that it can lead to job failure. We do not yet support spot instances in Resilin. However, since Nimbus provides an implementation of spot instances, we consider implementing this feature in the future.

Another axis of research has been focusing on optimizing the number, type and configuration of resources allocated for a MapReduce computation. Kambatla et al. [18] study how to configure the number of map and reduce tasks running in parallel on each compute node. They propose to analyze the behavior of an application and to match it against a database of application signatures containing optimal Hadoop configuration settings, in order to derive efficient configuration parameters.

6 Conclusion and Future Works

In this paper, we presented Resilin, an implementation of the Elastic MapReduce API that allows users to run MapReduce computations on resources originating from other clouds than Amazon EC2, such as private and community clouds. Resilin takes care of provisioning Hadoop clusters and submitting jobs, allowing users to focus on writing their MapReduce applications rather than managing cloud resources. Resilin uses the EC2 API to interact with IaaS clouds, making it compatible with most open source IaaS toolkits. We evaluated our implemen-

tation of Resilin using a Nimbus cloud deployed on the Grid'5000 testbed and compared its performance with Amazon Elastic MapReduce, both for deployment and execution time. Results show that Resilin offers a level of performance similar to Elastic MapReduce.

As future work, we plan to complete compatibility with the Amazon Elastic MapReduce API, by providing Hive and Pig job flows. We will also evaluate Resilin with other cloud implementations than Nimbus, such as Eucalyptus and its Walrus cloud storage repository. We will optimize the deployment process and analyze reasons for executions slower than Elastic MapReduce. For users having access to multiple clouds, we will investigate how Resilin could automatically select which cloud infrastructure to use, instead of relying on users to choose where they want to provision resources. Finally, we plan to release the Resilin software as open source in the near future.

Acknowledgment

The authors would like to thank Kate Keahey for discussions on early versions on this work, and John Bresnahan for his help related to the Cumulus implementation. Experiments presented in this paper were carried out using the Grid'5000 experimental testbed, being developed under the INRIA ALADDIN development action with support from CNRS, RENATER and several Universities as well as other funding bodies (see `https://www.grid5000.fr`).

References

[1] Apache Hadoop. `http://hadoop.apache.org/`.

[2] boto: A Python interface to Amazon Web Services. `http://boto.cloudhackers.com/`.

[3] Eucalyptus. `http://www.eucalyptus.com/`.

[4] FutureGrid. `https://portal.futuregrid.org/`.

[5] Great Internet Mersenne Prime Search. `http://www.mersenne.org/freesoft/`.

[6] Interacting with Walrus (2.0). `http://open.eucalyptus.com/wiki/EucalyptusWalrusInteracting_v2.0`.

[7] Nimbus. `http://www.nimbusproject.org/`.

[8] OpenNebula. `http://www.opennebula.org/`.

[9] paramiko. `http://www.lag.net/paramiko/`.

[10] Twisted. `http://twistedmatrix.com/trac/`.

[11] John Bresnahan, Kate Keahey, David LaBissoniere, and Tim Freeman. Cumulus: An Open Source Storage Cloud for Science. In *2nd Workshop on Scientific Cloud Computing*, pages 25–32, 2011.

[12] Michael Cardosa, Chenyu Wang, Anshuman Nangia, Abhishek Chandra, and Jon Weissman. Exploring MapReduce Efficiency with Highly-Distributed Data. In *MapReduce '11*, 2011.

[13] Navraj Chohan, Claris Castillo, Mike Spreitzer, Malgorzata Steinder, Asser Tantawi, and Chandra Krintz. See Spot Run: Using Spot Instances for MapReduce Workflows. In *2nd USENIX Workshop on Hot Topics in Cloud Computing*, pages 1–7, 2010.

[14] Cycle Computing. Lessons learned building a 4096-core Cloud HPC Supercomputer for $418/hr. `http://blog.cyclecomputing.com/2011/03/cyclecloud-4096-core-cluster.html`.

[15] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *6th Symposium on Operating Systems Design and Implementation*, pages 137–149, 2004.

[16] Thilina Gunarathne, Tak-Lon Wu, Judy Qiu, and Geoffrey Fox. MapReduce in the Clouds for Science. In *Proceedings of the 2010 IEEE Second International Conference on Cloud Computing Technology and Science*, pages 565–572, 2010.

[17] Keith R. Jackson, Lavanya Ramakrishnan, Krishna Muriki, Shane Canon, Shreyas Cholia, John Shalf, Harvey J. Wasserman, and Nicholas J. Wright. Performance Analysis of High Performance Computing Applications on the Amazon Web Services Cloud. In *Proceedings of the 2010 IEEE Second International Conference on Cloud Computing Technology and Science*, pages 159–168, 2010.

[18] Karthik Kambatla, Abhinav Pathak, and Himabindu Pucha. Towards optimizing hadoop provisioning in the cloud. In *USENIX Workshop on Hot Topics in Cloud Computing*, pages 1–5, 2009.

[19] Katarzyna Keahey, Ian Foster, Tim Freeman, and Xuehai Zhang. Virtual Workspaces: Achieving Quality of Service and Quality of Life in the Grid. *Scientific Programming*, 13(4):265–275, 2005.

[20] Katarzyna Keahey, Maurício Tsugawa, Andréa Matsunaga, and José A. B. Fortes. Sky Computing. *IEEE Internet Computing*, 13(5):43–51, 2009.

[21] Kate Keahey and Tim Freeman. Science Clouds: Early Experiences in Cloud Computing for Scientific Applications. In *First Workshop on Cloud Computing and its Applications*, pages 1–5, 2008.

[22] Huan Liu. Cutting MapReduce Cost with Spot Market. In *3rd USENIX Workshop on Hot Topics in Cloud Computing*, pages 1–5, 2011.

[23] Huan Liu and Dan Orban. Cloud MapReduce: a MapReduce Implementation on top of a Cloud Operating System. In *11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 464–474, 2011.

[24] Andréa Matsunaga, Maurício Tsugawa, and José Fortes. CloudBLAST: Combining MapReduce and Virtualization on Distributed Resources for Bioinformatics Applications. In *4th IEEE International Conference on e-Science*, pages 222–229, 2008.

[25] Microsoft. Windows Azure Platform. `http://www.microsoft.com/windowsazure/`.

[26] Daniel Nurmi, Rich Wolski, Chris Grzegorczyk, Graziano Obertelli, Sunil Soman, Lamia Youseff, and Dmitrii Zagorodnov. The Eucalyptus Open-source Cloud-computing System. In *9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 124–131, 2009.

[27] Lavanya Ramakrishnan, Piotr T. Zbiegel, Scott Campbell, Rick Bradshaw, Richard Shane Canon, Susan Coghlan, Iwona Sakrejda, Narayan Desai, Tina Declerck, and Anping Liu. Magellan: Experiences from a Science Cloud. In *2nd Workshop on Scientific Cloud Computing*, pages 49–58, 2011.

[28] Michael C. Schatz. CloudBurst: highly sensitive read mapping with MapReduce. *Bioinformatics*, 25(11):1363–1369, 2009.

[29] Amazon Web Services. Amazon EC2 Spot Instances. `http://aws.amazon.com/ec2/spot-instances/`.

[30] Amazon Web Services. Amazon Elastic MapReduce. `http://aws.amazon.com/elasticmapreduce/`.

[31] Amazon Web Services. Word Count Example. `http://aws.amazon.com/articles/2273`.

[32] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The Hadoop Distributed File System. In *Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, pages 1–10, 2010.

[33] Konstantin V. Shvachko. Apache Hadoop: The Scalability Update. *;login:*, 36(3):7–13, June 2011.

[34] Borja Sotomayor, Rubén S. Montero, Ignacio M. Llorente, and Ian Foster. Virtual Infrastructure Management in Private and Hybrid Clouds. *IEEE Internet Computing*, 13(5):14–22, 2009.

[35] Ashish Thusoo, Zheng Shao, Suresh Anthony, Dhruba Borthakur, Namit Jain, Joydeep Sarma, Raghotham Murthy, and Hao Liu. Data Warehousing and Analytics Infrastructure at Facebook. In *2010 ACM SIGMOD International Conference on Management of Data*, pages 1013–1020, 2010.

Centre de recherche INRIA Rennes – Bretagne Atlantique
IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)

Centre de recherche INRIA Bordeaux – Sud Ouest : Domaine Universitaire - 351, cours de la Libération - 33405 Talence Cedex
Centre de recherche INRIA Grenoble – Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier
Centre de recherche INRIA Lille – Nord Europe : Parc Scientifique de la Haute Borne - 40, avenue Halley - 59650 Villeneuve d'Ascq
Centre de recherche INRIA Nancy – Grand Est : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex
Centre de recherche INRIA Saclay – Île-de-France : Parc Orsay Université - ZAC des Vignes : 4, rue Jacques Monod - 91893 Orsay Cedex
Centre de recherche INRIA Sophia Antipolis – Méditerranée : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399