Processing Wikipedia Dumps: A Case-Study comparing the XGrid and MapReduce Approaches

Dominique Thiébaut* Yang Li Diana Jaunzeikare Alexandra Cheng
Ellysha Raelen Recto Gillian Riggs Xia Ting Zhao Tonje Stolpestad
Cam Le T Nguyen

{dthiebau,yli2,djaunzei,acheng,erecto,griggs,xzhao,tstolpes,lnguyen}@smith.edu
Department of Computer Science, Smith College
Northampton, MA, 01060, USA

February 2011

Abstract

We present a simple comparison of the performance measured as the total execution time taken to parse a 27-GByte XML dump of the English wikipedia on three different cluster platforms: Apple's XGrid, and Hadoop the open-source version of Google's MapReduce. We use a local hadoop cluster of Linux workstation, as well as an Elastic MapReduce cluster rented from Amazon. We show that for selected benchmark, XGrid yields the fastest execution time, with the local Hadoop cluster a close second. The overhead of fetching data from Amazon's Simple Storage System (S3), along with the inability to skip the reduce, sort, and merge phases on Amazon penalizes this platform targetted for much larger data sets.

Keywords: Grid Computing, XGrid, Hadoop, wikipedia, data mining, performance

1 Introduction

The aim of this paper is to find the fastest parallel computer cluster for processing Wikipedia dumps and for generating word statistics. We implement the parsing of the XML dump of the English wikipedia, and the gathering of word usage an Apple XGrid system, as well as on two Hadoop clusters, one local, the other hosted by Amazon, and we report on the performance obtained on the different systems.

Apple's XGrid [8] was released in 1994 and is part of the standard OS X distribution. It is a loosely coupled system consisting of a controller and many agents. The agents advertise their availability to the controller. The XGrid offers a low-cost, easy, versatile, and powerful infrastructure for executing data-parallel programs on tens, or hundreds of Mac computers. Judging from the published literature reporting computational results on it, it is used mostly in academia and for scientific computing, less so in commercial applications.

Hadoop [15] is an open-source software framework inspired by Google's MapReduce [6], and runs mostly on Linux machines, but can also run on other platforms. It supports data-intensive distributed applications running on thousands of nodes processing petabytes of data, and it is resilient to node failure. Hadoop has a strong presence in the commercial world, with Yahoo! its main contributor [5].

The two different frameworks target different data processing needs, hadoop for large scale applications processing petabytes of data where node failure is an expected way of life, and XGrid in a generally smaller context of a research or eductional institution, on highly data-parallel scientific data. While the XGrid is a natural choice for parsing the 30 GBytes of a dump of the English wikipedia, in this paper we investigate whether

*contact author

hadoop is a contender, and under which conditions the execution time of the parsing process is comparable under both frameworks.

When low-cost, powerful, and easily accessible parallel computational platforms are available, it is important to better understand their source of performance and pick the best one for a solving a given problem.

Our results show that the XGrid is the fastest platform for our need, with the local Hadoop cluster a close second. The size of our data set is too low, and our computation too simple to offset Amazon's high storage access overhead. However, Amazon can still provide acceptable support in its multicore single-machine options.

In the next section we review background and state of the art information of comparative studies involving XGrid and/or Hadoop clusters. Then we set the conditions of our experiments in precise terms. In Section 4, we present the XGrid cluster and report its performance on our workload. In Section 5 we concentrate on the local Hadoop cluster first, and then on Amazon's Hadoop infrastructure. We present our analysis in Section 6, and conclude in Section 7.

2 Background

The individual performance of the XGrid and Hadoop systems under various loads and conditions have been reported extensively, but few reports compare them to other systems or to each other. Different studies offer comparison of grids using different metrics, such as evaluating the scheduling policy only [10].

Because Hadoop follows the approach of Google's MapReduce, it is well suited to process large data sets in text format, and its performance and tuning for sorting large text files has been carefully studied [11].

Among the very few published comparisons of hadoop/MapReduce against other computing solutions, that of Pavlo et al. [12] along with a rebutal by Google's Dean and Ghemawat [7] stand out. Pavlo et al. compare the MapReduce paradigm to parallel and distributed SQL database management systems (DBMS), and heralds the superiority of DBMS in terms of their faster response times and need for fewer CPUs, but concedes on the superiority of Hadoop for its fault tolerance and ease of installation. Dean and Ghemawat's rebutal published in *CACM* is an indication that the fair comparison of different approaches to solve identical parallel problems is nontrivial, and the assumptions made in setting up the experiments can greatly influence the results. In the case of [12], one system requires long pre-processing of the data before storage in the database, but uses efficient indexing methods to return quick results to multiple queries, while the other does not use preprocessing and generates one output set with all the results.

Iosup and Epema [9] propose the creation and use of synthetic benchmarks to rate the performance of grids, and evaluate their system on one system only, which unfortunately uses neither XGrid nor Hadoop.

The work in [13] is a very comprehensive framework for measuring detailed performance measures of various grids, but their comparative study included world-wide network of research computers, or centralized networks of Linux workstation, which didn't include Hadoop or XGrid.

The evaluation of MapReduce on multicore systems presented by Ranger [14] is worth mentioning, and although it does not compare Hadoop's performance to that of other grids, it deserves special mention, as machines with up to 26 cores can currently be rented from Amazon Web services, and present an alternative to small clusters. We come back to this study in the conclusion section.

As many of the research cited above have stressed, it is a challenging task to provide a fair comparison of the computational power of systems that operate at different clock speeds, with different memory support, using different file systems, and with different network bandwidth. None the less, we feel that our attempt to answer the question of which of XGrid or Hadoop yields the fastest execution time for processing Wikipedia is worth investigating, and we hope our results will help others in their search for the best computing platform for their need.

3 Problem Statement

This paper reports on the findings of a simple exploration: find among the available platforms the parallel approach that provides the fastest parsing of the entire contents of a dump of the English Wikipedia, where word-frequency statistics is gathered for all pages.

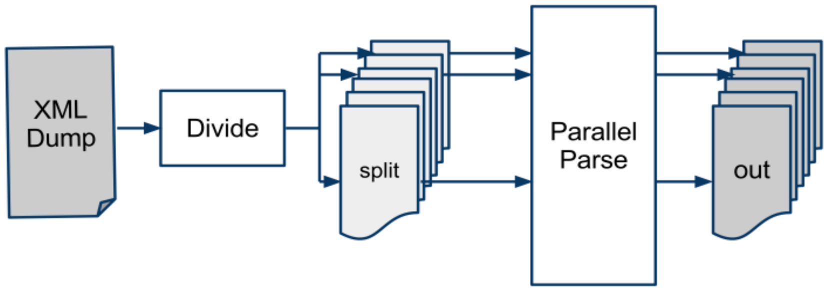

Figure 1: Main block diagram of the processing of the XML dump.

The complete process is illustrated in Figure 1. The data dump (pages-articles, dated 10/17/2009) is first retrived from Mediawiki [3], and then parsed by a serial compiled C++ program, *divide*, that formats 9.3 million XML entries into 431 XML files, each containing as many whole entries as can fit in 64 Mbytes. We refer to these 431 files as *splits*, according to the hadoop syntax. In a split, each of the individual entries represents a wikipedia entry, and is formatted as a single line of text. This format allows the use of hadoop's default input filter that feeds individual split lines of text to the map tasks.

The 431 splits are then processed in parallel (*parallel parse*), stop words are removed and word statistics (most frequent words and groups of words) are computed and stored in some number N of result files (*out files*). N varies depending on the platform and configuration used. The contents of these files are then further processed (not shown in Figure 1) and stored in an SQL database. We do not include this phase in the analysis reported here.

Our intent is to implement the *parallel parse* module with hadoop and with XGrid, to compare their relative performance, and to specify as simply and cleanly as possible the input and output interface so that the comparison is meaningful. We pick three different frameworks that are easily accessible: a local XGrid cluster of iMacs, a local cluster of Fedora-workstation running Hadoop, and Amazon's *Elastic MapReduce* framework [1]. The input and output files may reside in different storage systems, that are be selected so as to make the parallel computation of Wikipedia an easily interchangeable stage of a larger computational pipeline. For this reason, we include setup, upload and download times of input and output files in our comparison.

The target execution time to beat is 15 hours 34 minutes and 28 seconds (56,068 seconds) for the serial execution of *parallel parse* on one of the 2-core iMac workstations used as standard agent in our XGrid, with all files (executable, input and output) stored on the iMacs local disk. This execution involves only one core, due to the lack of multitasking of the compiled C++ program.

In the next section we present the XGrid infrastructure and the performance measured on this network.

4 XGrid setup

4.1 Hardware Setup

The XGrid is a collection of iMac workstations located on the same 1-gigabit LAN in the department of computer science at Smith College. The XGrid controller is an 2.8 GHz 8-core MacPro3.1, with 10 GB Ram. The agents are 3.06 GHz Intel 2-core duos iMacs, with a maximum of 42 processors available in the grid. The experiments are conducted at times where the computers are free of users.

4.2 Workload Setup

We use a compiled C++ program to process each of the 431 input files and to generate the output files. The program uses Nokia's Qt library [4] for its efficient and versatile string and XML processing capabilities.

Table 1: XGrid Execution Times as a function of Processors

Number of Processors	Execution Times (sec)	
25	45 min 11 sec	(2,711 sec)
32	41 min 3 sec	(2,463 sec)
34	35 min 2 sec	(2,102 sec)
41	36 min 0 sec	(2,160 sec)

Because not all the nodes in the grid sport the Qt library, a bundle of binaries is created that includes the C++ executable and the set of all the libraries on which it depends.

Because of the number of input files (431) and their size ($\sim$ 64MB), we elect to use *batch mode* processing.

Unfortunately, due to the size of the binary bundle, and the size of input data files, no more than 4 split files can be included in a batch script before a dynamic memory allocation error is generated upon submission to the grid controller. Even if the appropriate limit parameter is modified, the 27 GB of input data would generate a batch file twice that size, which would block the paging system of our controller.

For this reason, the input files are stored on a Web server on the same subnet as the XGrid, from where they are downloaded by the XGrid agents. While the output files can easily be generated as temporary file that are automatically retrieved by the XGrid controller, we elect to have the XGrid agents upload the result files back to the same local Web server. Our measurements show that the upload to the Web server is less than 1% longer in relative execution time compared to the automatic handling and downloading by XGrid. The resulting batch file is 31 MBytes in size, and contains the binary bundle described above, along with multiple single-line commands specifying the name of the C++ executable followed by the URL of the input split.

4.3 Execution Times

Table 1 presents the average execution times observed when processing the 431 input files on the XGrid. The best speedup observed is 26.6 for 34 cores, corresponding to the fastest execution time of 2,102 seconds, or 35 min 2 sec. The different node configurations illustrate the variation in node availability typical of workstations located in public lab spaces.

5 Hadoop Setup

Two different hadoop platforms are used for this test. One is a local cluster of 10 Fedora 2 GHz, quad-core Intel Linux workstations with 8 GB Ram, on a 1-gigabit LAN. The other is the AWS platform offered by Amazon [1].

5.1 Hadoop on Local Cluster

5.1.1 Experiment Setup

Cloudera's patched Hadoop 0.18 distribution [2] is selected to run the experiments because of its robustness and available support. To same C++ program used on the XGrid is modified to become the *map task* in Hadoop's Map-Reduce organization. In this case we do not use a *reduce task*, which has the advantage of forcing Hadoop to automatically upload the local result files generated by the map tasks to distributed file storage. They form the result files of our computation. The C++ executable is run in *streaming mode*.

The ten quad-core CPUs form a set of 40 processors to run the Map and Reduce tasks, comparable in number of cores to the XGrid cluster, with an aggregated processing power of 80 GHz for Hadoop, versus 125.5 GHz for the XGrid cluster.

The 1 gigabit LAN linking the Fedora workstations is the same network linking the iMacs of the XGrid cluster. Each cluster resides in a lab of its own, in the same building.

The number of map tasks running concurrently (mapred.tasktracker.map.tasks,maximum) is set to 4, as it yields the fastest execution time. This corresponds to one map task running on each core. This generates 4 x

Table 2: Execution times on local Hadoop Fedora cluster.

Operation	Execution Times	
T0: Upload to HDFS	14 min 0 sec	(840 sec)
T1: Computation only (result in HDFS)	22 min 37 sec	(1,357 sec)
T2: Download result from HDFS	1 min 53 sec	(113 sec)
T3: Computation + I/O (T1+T2)	24 min 30 sec	(1,470 sec)
T4: Concurrent Computation and Download	23 min 20 sec	(1,400 sec)

431 = 1724 output files which are stored directly in the *Hadoop Distributed File System*, or HDFS, at the end of the MapReduce computation since the number of reduce tasks (mapred.reduce.tasks) is set to 0.

5.1.2 Performance

The execution times for setting up, executing, and downloading the results of the MapReduce program are presented in Table 2, and the total time is 1,400 seconds, or 23 min 20 sec. T0 represents the time taken to upload the input splits to HDFS. This is a required condition for MapReduce applications. This step involves creating 2 copies (*replication factor* = 2) of the same split on the distributed (local) disks of cluster workstations. T1 represents the execution of the MapReduce program, including replication of the binaries to the compute nodes, execution, and storage of the resulting files in HDFS. T2 is the time taken to download the files from HDFS to the local disk of a workstation on the same subnet. T3 is the sum of T1 and T2. T4 represents the time taken if the output files are downloaded from HDFS as soon as they are created, in essence overlapping execution and I/O. While T4 is slightly lower than T3, we elect to use T3 as the representative quantity, which will make our comparison easier in Figures 2 and 3 below.

5.2 Hadoop on Amazon Web Services

The C++ map task used in the previous section is first compiled on an Amazon *Elastic Compute Cloud* (EC2) [1] *m1-large instance* initialized with the required applications (qt, qt-devel, emacs, g and make), so that it can easily be distributed to an elastic MapReduce cluster of m1-large instances later. The m1-large instance consists of 4 virtual 64-bit cores, 7.5 GB Ram, and 850 GB of disk space. A static bundle of the executable and all the libraries it depends upon is then loaded in an Amazon *Simple Storage Service* (S3) *bucket* where it will be available to the hadoop infra-structure in the form of *cache files*.

Because the Amazon platform does not allow for reduce-free MapReduce programs, we modify the map function (the C++ core compute function) to output each record with the same Id. These records are fed to an *identity* reduce function that simply passes the records through unchanged. Note that because the reduce step must be used, the sorting and merging of the output records are automatically performed by hadoop, a time consuming step we avoided in the local cluster. This is the reason for our choice of the same Id for all output records shortens the sorting phase.

Before the computation, the 431 splits are uploaded to S3 storage using Smith College's high speed network connection. Then 10 m1-large instances are harnessed into an Elastic MapReduce cluster of 10 nodes, or 40 cores. The resulting files are then downloaded from S3 to a local disk. The 10-node cluster contains 40 cores, and is the closest match to the XGrid and local Hadoop setups.

5.3 Performance Measurements

The average times observed for the upload, execution, and download operations on the 431 splits on the 10-node Elastic MapReduce cluster are presented in Table 3. Note that we use the ability of Hadoop to automatically read compressed data files as well as generate compressed output files, so as to minimize the upload and download times.

We also measure the execution times of our hadoop program on 5, 10, and 15-node clusters, and record the execution times in Table 4. The table includes the number of *instance hours*, which is the number of instances

Table 3: Execution times on Amazon Web Services.

Operation	Execution Times	
T0: Upload splits to S3	49 min 17 sec	(2,957 sec)
T1: Computation only (result in HDFS)	1 hr 19 min 0 sec	(4,740 sec)
T2: Download results from S3	16 min 0 sec	(960 sec)

Table 4: Measured Execution times as a function of cluster size on Amazon Web Services.

Number of Nodes	Number of cores	Execution Times		Instance Hours	Efficiency
5	20	2 hr 12 min	(7,920 sec)	60	0.19
10	40	1 hr 19 min	(4,740 sec)	80	0.16
15	60	1 hr 6 min	(3,960 sec)	120	0.12
20	80	1 hr 1 min	(3,660 sec)	160	0.10

used per hour of computation, the quantity used by Amazon for billing users. The MapReduce application is modified for each case so that there can be as many number of reduce tasks as there are cores. We note that even with twice the number of cores as the local Hadoop cluster, the computation on 80 core takes 3,660 seconds, close to three times the 1,357 seconds it takes on the 40-core local cluster. We further note that the 5-node, 20-core AWS cluster is the most *efficient* of the four sizes considered, with an *efficiency* (speedup/number of cores) of 0.19, as opposed to 0.16, 0.12, and 0.10 for the other configurations.

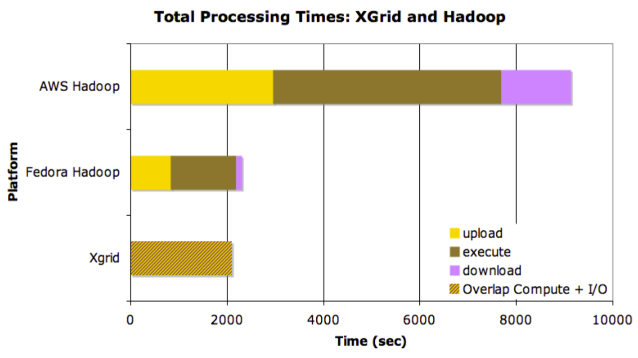

Figure 2: Execution times on the three platforms, XGrid, Fedora Hadoop and AWS Hadoop.

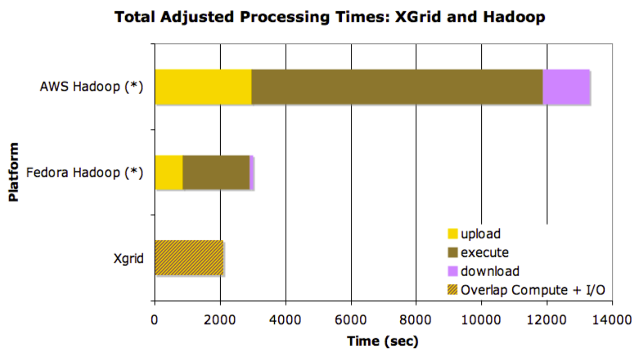

Figure 3: Adjusted execution times for XGrid, Fedora Hadoop, and AWS Hadoop.

6 Comparative Analysis

6.1 One-Core Processing

The purpose of this section is normalize the times measured on the different networks. We compute the speed of the three infrastructures relative to each other by processing all 431 splits on a single core of each platform, with the input and output files retrieved from and stored back on the same Web Server. For the XGrid workstation, we simply run the compiled, single-thread, C++ program in a shell of the OSX 10.6 workstation. For the hadoop workstations, we run the program directly from the command line, by-passing the hadoop java framework. In all cases the splits are fetched from a Web server at Smith College, and the result uploaded back to the same server, favoring the on-campus computers. The *m1-large* EC2 instance on Amazon performs the fastest, 1.88 times faster than the XGrid processor, and 1.23 times faster than the Fedora Hadoop core, as illustrated in Table 5.

Table 5: Serial execution times and multiplicative factors.

Platform	Execution Times		X/A	X/B	X/C
A. XGrid	15 h 57 min 47 sec	(57,467 sec)	1.00	1.52	1.88
B. Local Hadoop	10 h 28 min 23 sec	(37,703 sec)	0.66	1.00	1.23
C. AWS Hadoop	8 h 28 min 35 sec	(30,515 sec)	0.53	0.81	1.00

Table 6: Multithreaded Execution on a 26-core m1-large AWS EC2 Instance.

Operation	Execution Time	
T0: Download from Web Server to EC2 Instance	28 min	(1,680 sec)
T1: Unzip Split and Computation	36 min 22 sec	(2,182 sec)
T2: Upload to Web Server	9 min 40 sec	(580 sec)
T3: Total (T0+T1+T2)	1 hr 14 min 02 sec	(4,442 sec)

6.2 Comparing the Performance of the Parallel Program on all Three Platforms

Figure 2 shows the time taken to transfer 431 input files to the selected platform, process them in parallel, and get the resulting files back. These times are nicely separated for the Hadoop platforms, but because the XGrid platform allows individual nodes to download, process, and upload back to the server in parallel with other nodes, the three different actions are merged into one solid block.

The first observation is that the XGrid performs faster than the other two platforms.

The second observation is that the local Hadoop cluster performs much faster than the AWS cluster. This can be attributed primarily to the overhead of the S3 storage, as well as the sort and merge phase that follows the reduce steps in the AWS case. Remember that while we could implement a reduce-less MapReduce program on the local Hadoop cluster, Amazon does not allow for it. As a result, 40 of the the fastest cores used in this study take the longest processing time.

We see that for the computation only, the local Fedora hadoop infrastructure is more efficient, with XGrid a close second. However, because download and upload are integrated in the measured execution time, XGrid performs actually better than reported.

To reduce the disparity in the architectures and provide a more meaningful comparison, we show in Figure 3 the same data as in Figure 2, but we multiply the execution times on the hadoop platforms by the factors of Row 1 of Table 5, 1.52 for the local hadoop, and 1.88 for the AWS hadoop. In effect we simulate the hadoop platforms on XGrid cores. Here again we observe the fastest execution on the XGrid, with the local Hadoop close to 1.5 times slower, and the AWS hadoop more than 6 times slower.

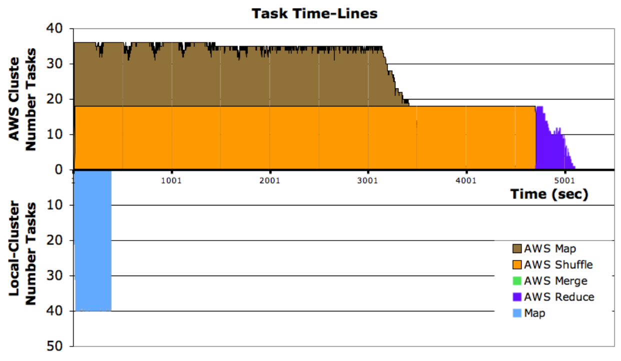

Figure 4: Time-Lines of the map-reduce-shuffle tasks on AWS (positive axis) and the map-only tasks on the local Hadoop cluster. The upper part of the stacked-area graph shows the number of map, reduce, shuffle and sort tasks executing as a function of time on the AWS cluster, while the loower part of the graph shows the number of map tasks (there are no reduce, shuffle or sort tasks needed) on the local cluster.

The performance of Amazon's Elastic MapReduce is disappointing when compared to that of the local Hadoop cluster because of several reasons. First, it is ill-suited for the small-scale problem we have applied it to. Second, it fetching its data from S3 which is a wide-area attched storage, as compared to the locally stored data of the local Hadoop cluster. Finally because the algorithm does not require a reduce phase, which is required on Amazon MapReduce clusters, and must be implemented, eating way significant processing power from the nodes. Figure 4 which shows the time-lines for the Map-Reduce-Shuffle computation on Amazon, and the Map-only computation on the local hadoop cluster illustrates this difference in performance.

One shouldn't underestimate the power of the variety EC2 instances available. As an experiment we wrapped our C++ executable into a multithreaded Python program (using the multiprocessing module, and not the threads module, which only runs one thread at a time) and ran it on Amazon's largest instance, a 26-core, 68.4 GB *m2.4x-large* instance, and recorded its execution time. The transfer of data to and from the Smith Web server and the threaded computation takes only three times longer that the same operations on the local Hadoop cluster with almost twice as many cores (40), and with a total execution time of 1 hour 14 minutes it is a viable option for our purpose. This reinforces the observation of Ranger [14] who observes similar performance for MapReduce programs on multicores with shared memory as with P-Threads, indicating that HDFS and network communication are responsible for significant overheads.

7 Conclusions

We have compared three different parallel platform for gathering statistics from an XML dump of the English Wikipedia. The three platforms include Apple's XGrid, Hadoop on a local 10-workstation cluster, and Hadoop on Amazon. Our goal is to pick the fastest platform for processing data of importance in another research project requiring the collection of statistics from Wikipedia XML dumps. The Wikipedia dumps is divided into 431 input files whose size is selected to contained the maximum number of wikipedia entries while the total size remains under 64 MB.

Our efforts thus provide one data-point in the comparison of these three platforms, and not a comprehensive comparison or evaluation of the different parameters influencing the execution times. Such a study, incuding the tuning of various parameters, such as the size of the input splits, the number of reduce tasks, the C++ library used is left for future work.

Of the three platforms considered, XGrid is found to provide the fastest delays for gathering the input data, processing them, and storing the result back in a place where it can easily be reached for the next step of the process. The Local hadoop cluster, with its distributed file system spread over the local disks of the ten workstations performs almost as well.

While the AWS cluster performs worst of the three, one should remember that this commodity-computing solution has great appeals, in particular that of requiring no physical or manpower infrastructures to host it or manage it, and that users pay only for the number of hours the instances are involved in computation. The 2010 cost of performing the processing of Wikipedia on all of the different clusters (5, 10, 15, and 20 nodes) was less than US-$60.

Another advantage provided by Hadoop clusters is that they tolerate hardware and software failures, and the parallel computation will continue and succeed even if one or several nodes disappear from the network. This was clearly evident during one set of our experiments when we had to discard a set of data after discovering that one of the nodes in the local hadoop cluster had ceased working and did not participate in an unknown number of experiments. XGrid is not as hardened, and the whole computation will fail if one node fails or is disconnected during a parallel run.

While AWS hadoop is clearly not the most performant in this case, we strongly suspect that the 27 GB size of the input data is simply too small to make this platform shine, and that its true power is best expressed on much larger data sets.

Acknowledgements

We are happy to acknowledge Kurt Messersmith of Amazon.com for the grant of free compute time on Amazon Web Services. This time was used primarily in the class CSC352, Introduction to Distributed and Cloud Computing, and for this research.

References

[1] Amazon web services. `http://aws.amazon.com/`.

[2] Cloudera. `http://cloudera.com/`.

[3] Mediawiki foundation. `http://www.mediawiki.com/`.

[4] Nokia Qt cross-platform application and UI framework library. `http://qt.nokia.com/`.

[5] Eric Baldeschwieler. Yahoo! launches world's largest hadoop production application. `http://developer.yahoo.net/blogs/hadoop/2008/02/yahoo-worlds-largest-production-hadoop.html`, 2008.

[6] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, 2008.

[7] Jeffrey Dean and Sanjay Ghemawat. MapReduce: A flexible data processing tool. *Communications of the ACM*, 53(1):72–77, 2010.

[8] Baden Hughes. Building computational grids with apple's xgrid middleware. In Rajkumar Buyya and Tianchi Ma, editors, *Fourth Australasian Symposium on Grid Computing and e-Research (AusGrid 2006)*, volume 54 of *CRPIT*, pages 47–54, Hobart, Australia, 2006. ACS.

[9] Alexandru Iosup and Dick Epema. Grenchmark: A framework for analyzing, testing, and comparing grids. In *CCGRID '06: Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid*, pages 313–320, Washington, DC, USA, 2006. IEEE Computer Society.

[10] Majd Kokaly, Issam Al-Azzoni, and Douglas G. Down. Mgst: A framework for performance evaluation of desktop grids. *Parallel and Distributed Processing Symposium, International*, 0:1–8, 2009.

[11] Owen O'Malley and Arun Murthy. Hadoop sorts a petabyte in 16.25 hours and a terabyte in 62 seconds. `http://developer.yahoo.net/blogs/hadoop/2009/05/hadoop_sorts_a_petabyte_in_162.html`, 2009.

[12] Andrew Pavlo, Erik Paulson, Alexander Rasin, Daniel J. Abadi, David J. DeWitt, Samuel Madden, and Michael Stonebraker. A comparison of approaches to large-scale data analysis. In *SIGMOD Conference*, pages 165–178, 2009.

[13] Ioan Raicu, Catalin Dumitrescu, Matei Ripeanu, and Ian Foster. The design, performance, and use of diperf: An automated distributed performance testing framework. In *the Journal of Grid Computing, Special Issue on Global and Peer-to-Peer Computing*, 2006.

[14] Colby Ranger, Ramanan Raghuraman, Arun Penmetsa, Gary Bradski, and Christos Kozyrakis. Evaluating mapreduce for multi-core and multiprocessor systems. In *HPCA '07: Proceedings of the 2007 IEEE 13th International Symposium on High Performance Computer Architecture*, pages 13–24, Washington, DC, USA, 2007. IEEE Computer Society.

[15] Tom White. *Hadoop: The Definitive Guide*. O'Reilly, first edition, june 2009.