VMR: Volunteer MapReduce over the Large Scale Internet

Fernando Costa, Luís Veiga, and Paulo Ferreira
Distributed Systems Group, INESC-ID
Technical University of Lisbon
R. Alves Redol, 9
1000-029 Lisboa, Portugal

ABSTRACT

Volunteer Computing systems (VC) harness computing resources of machines from around the world to perform distributed independent tasks. Existing infrastructures follow a master/worker model, with a centralized architecture, which limits the scalability of the solution given its dependence on the server. We intend to create a distributed model, in order to improve performance and reduce the burden on the server.

In this paper we present VMR, a VC system able to run MapReduce applications on top of volunteer resources, over the large scale Internet. We describe VMR's architecture and evaluate its performance by executing several MapReduce applications on a wide area testbed.

Our results show that VMR successfully runs MapReduce tasks over the Internet. When compared to an unmodified VC system, VMR obtains a performance increase of over 60 % in application turnaround time, while reducing the bandwidth use by an order of magnitude.

Categories and Subject Descriptors

H.5.3 [**Information Interfaces and Presentation**]: Group and Organization Interfaces—*Collaborative Computing*; D.1.1 [**Programming Techniques**]: Applicative (Functional) Programming; C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Cloud Computing*

General Terms

Experimentation, Performance

Keywords

Volunteer Computing, MapReduce, Large Scale Distributed Systems

1. INTRODUCTION

The use of volunteer PCs across the Internet to execute distributed applications has been increasing in popularity since its inception, with projects like Seti@home[2] or Folding@home[12]. These Volunteer Computing (VC) systems harness computing resources from machines running commodity hardware and software, and perform highly parallel computations that do not require any interaction between network participants (also called bag-of-tasks).

Existing VC systems support over 60 scientific projects,[1] and have over a million participants, rivaling supercomputers in computing power. The number of Internet connected devices is expected to increase exponentially with the advent of mobile devices [6]. Furthermore, Moore's law continuous relevance shows that we can expect a sustained evolution of the hardware in the last mile of the Internet. This translates to an incredible amount of untapped computing and storage potential in machines spread throughout the world.

However, current VC systems have a centralized architecture that follows a master/worker model, as a small number of servers is responsible for task distribution and result validation. This limitation has prevented Volunteer Computing from reaching its true potential. In addition, the single point of failure inevitably creates a bottleneck, as projects expand and storage and network requirements become more demanding.

1.1 Goal

Our goal is to improve the performance of Volunteer Computing systems, by decentralizing some of the mechanisms of existing systems that place an excessive burden on the central server. There are several challenges and requirements to consider, in order to achieve our objective.

First and foremost, our solution must be able to take advantage of the huge amount of VC resources that we previously mentioned. We must consider both the hardware capabilities of individual machines and the network bandwidth that is at our disposal, at the last mile of the Internet.

Our system must also be compatible with existing VC solutions (e.g. BOINC [1]). Developing a whole new platform from scratch would be of no practical use once the research was finished. In fact, our solution would undoubtedly bring significant disadvantages if it required that only our system's clients were attached to a project.[2] To avoid this situation we must guarantee compatibility with existing projects. Any

MGC2012, December 3, 2012, Montreal, Quebec, Canada.

[1]List of active VC projects.
`http://www.distributedcomputing.info/projects`

[2]A VC Project runs on top of existing middleware (e.g. BOINC) by developing an application and defining all parameters concerning its execution. Project developers only have to make sure their tasks are properly configured and provide a publicly accessible machine to act as the VC server.

client must be able to run any project application. On the other hand, our solution must support existing applications, and successfully schedule tasks on existing clients.

Additionally, the execution of our system on unreliable, non-dedicated resources requires fault tolerance mechanisms. This means it must account for unreachable clients, which have disconnected from the server, or are simply offline.

Another potential problem is caused by byzantine behavior. Clients may maliciously return incorrect results, or inadvertently produce an incorrect output.

Finally, our solution must be able to withstand transient server failures. This is particularly important in our case because we will be dealing with long running applications, with a potentially high level of server interactions. We need to prevent task execution on the clients from coming to a halt, as they wait for the server to come back up.

1.2 Programming Paradigm

As parallel and distributed computing becomes the answer for increased scalability for varied computational problems, several paradigms and solutions have been created during the last decade. In our first attempt at running data-intensive applications on top volunteer resources, we decided to select a popular paradigm, representative of different tasks. Among the potential candidates, MapReduce [9] has taken its place as one of the most widely used paradigms in cloud computing environments, such as Amazon's EC2. [3]

MapReduce leverages the concept of Map and Reduce commonly used in functional languages: a map task runs through each element of a list and produces a new list; reduce applies a new function to a list, reducing it to a single final value or output. In MapReduce, the user specifies a map function that processes tuples of key/values given as input, and generates a new intermediate list of key/value pairs. This map output is then used as input by a reduce function, also predefined by the user, that merges all intermediate values that belong to the same key. Therefore, all reduce inputs are outputs from the previous map task. Throughout the rest of the paper, we will be referring to them as map outputs. The adaptation of MapReduce to a volunteer environment is an interesting challenge because current implementations are limited to cluster environments.

1.3 Our Solution

Existing solutions do not fulfill the goal we described, and are inadequately prepared to meet the requirements mentioned above.

Most Volunteer Computing systems have a centralized architecture, with communication going through a single server. There are few exceptions and they were created with a smaller scope or environment in mind [5]. In BOINC [1], XtremWeb [3] and Folding@home[12], the server or coordinator must fulfill the role of job scheduler, by handling task distribution and result validation. This architecture creates too much overhead on the server when considering more complex data distribution or storage. Existing projects such as Climateprediction.net and MilkyWay@home have encountered problems when dealing with large files or having the same data shared by many clients [7].

Fault tolerance is strictly confined to the client-side in current VC systems. Although some projects do have a set of mirrors that act as data repositories, all client requests and task scheduling goes through the central server. Therefore, any server fault that prevents it from communicating with clients has a very high probability of disrupting clients and stopping further task execution.

Finally, a considerable limitation of existing VC systems is their focus on bag-of-tasks applications, without communication or dependencies between the tasks. None of the current platforms support MapReduce, a widely used programming model that adapts well to a data-intensive class of applications. Supporting MapReduce requires fundamental changes on existing algorithms, and the introduction of on-the-fly task creation.

In this paper we present VMR, a prototype that is able to execute MapReduce tasks over the large scale Internet, on top of volunteer resources. Our system is compatible with existing solutions (in particular BOINC). VMR is able to decentralize the existing architecture, by using client to client transfers, thus allowing VMR to tolerate transient server failures, as the clients depend merely on other peers for data. Our system is also capable of tolerating VC clients' failure by using replication (i.e. running the same task on several VC machines). Finally, byzantine behavior is controlled through the use of task validation in the server. By replicating each task at least twice, it is possible to compare the outcome and accept only the results in which a quorum has been reached.

This paper is organized as follows: VMR is presented in more detail in Section 2; Section 3 presents experimental results; related work is discussed in Section 4; and Section 5 concludes.

2. VMR

VMR's architecture consists of a central server, and clients which can assume two different roles: mappers, which are responsible for bag-of-tasks in the map stage; and reducers, which perform the aggregation of all map output in the reduce step. VMR is compatible with BOINC (Berkeley Open Infrastructure for Network Computing), the most successful and popular VC middleware to date. Consequently, it is able to borrow its mechanisms and algorithms to deal with many of the challenges of VC systems.

However, BOINC suffers from the fundamental drawback of overloading the server by following a master/worker model, in which a central server is responsible for scheduling and validating tasks. Although it is possible to use mirrors to hold data, many projects use a single machine for both data storage and scheduling. Furthermore, these mirrors act as web servers, as all data is transferred through HTTP, making this impractical to implement on VC clients. Therefore, BOINC projects do not fully exploit users' increasing bandwidth, and deploy compute intensive applications, since a few problems arise in data-intensive scenarios [8, 10].

The parameters of the MapReduce job to run on top of VMR are defined by the user, and stored in the server. This includes the number of map and reduce tasks, the executable files used by map and reduce tasks, as well as their hardware and software requirements. Once all the MapReduce job characteristics have been defined, the VMR server creates the map tasks, and stores this information in the its database – the VMR database is responsible for holding all persistent information on tasks, clients, and applications being run.

The overall VMR execution model is presented in Fig. 1. A group of mappers first requests work from the server (1). The server follows a simple scheduling procedure when

[3]Amazon EC2. `http://aws.amazon.com/ec2`

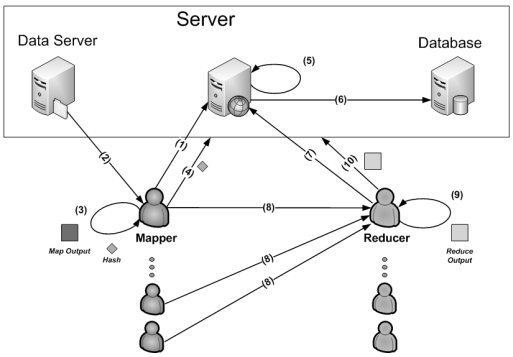

Figure 1: VMR Execution Model.

selecting which available task is assigned to each mapper or reducer: whenever it receives a work request, it matches each task's predefined hardware or software requirements to the client's machine characteristics. These requirements may include memory, disk space, CPU or Operating System specifications. If the client is suitable for the task, the server assigns it the task and saves this information in its database. After selecting an appropriate map task for the requesting mapper, the server sends back information on the task that the mapper must execute. This information includes the location of input and executable files, the deadline for task completion and the previously mentioned task requirements. The machines holding input and executable files are called data servers. Most VC projects store the data in the central server, as represented in Fig. 1.

The mapper must then download the required data from the data server (2) before starting the computation (3). After the task execution is completed, the mapper creates an MD5 hash for each of the map output files. Therefore, at the end of the computation, each mapper is left with both the map output files and the same number of corresponding hashes. These hash sums are sent back to the server in place of the output files (4) (so it is compatible with current VC solutions, e.g. BOINC). This greatly reduces the upload volume from mappers to the server.

The hashes are compared at the server in order to validate each corresponding task (5). If the result is valid, the mapper's address is stored in VMR's database (6). Each time a map result is validated, the VMR server checks if all map tasks have been executed and validated. When this condition is met, the server creates the predefined number of reduce tasks. A reducer may then send a work request to the project server (7), in order to be assigned a reduce task. The server follows task scheduling procedure defined earlier and looks through the database to find a task that can be assigned to the reducer. If the reducer meets all the hardware and availability requirements, the server replies with a reduce task that fits the request.

MapReduce jobs require communication between map and reduce stages since map outputs are used as input for reduce tasks. In the reduce step, each task performs join operations on the map outputs. Therefore, each reduce task must obtain all the map outputs that correspond to the key range it is responsible for. In order to achieve good performance in MapReduce jobs, we leverage clients' resources by moving as much of the communication as possible to the client-side. This helps reduce the load on the central server, and creates a more suitable decentralized model for data-intensive scenarios, typical of MapReduce.

Note that, as previously stated, in current VC systems all data would have to be uploaded and downloaded from the server. However, the VMR server stores the address of all mappers that returned valid map results. This information is included in the work request reply, and allows reducers to download the map output directly from the mappers, without having to go through the server (8). Once the input files have been downloaded, the reduce task is executed (9) and the final result is returned to the server (10) for validation.

3. EXPERIMENTS

We evaluate VMR by running several tests over the Internet, in a scenario that resembles a typical VC environment. We run experiments with 3 different applications (word count, inverted index, and N-Gram), in order to gauge our system's performance under different conditions. This section presents experimental results, describes the applications we use and reveals some of the implementation details.

3.1 Experimental Setup

Our goal is to improve the performance of VMR when running MapReduce applications. Therefore, we compare VMR with an existing VC system (BOINC). We use the VMR server in all our experiments, since current VC systems are unable to schedule MapReduce applications. As we previously mentioned, the server is compatible with clients from existing VC solutions. Therefore, we are able to use both VMR and unmodified BOINC clients in our experiments.

We measure application turnaround, while differentiating between map and reduce stages in order to pinpoint potential bottlenecks and areas that would benefit most from improvement. Additionally, we monitor network traffic on the server. This allows us to identify the benefits of reducing the dependence on the central server.

We run our experiments on PlanetLab [4], a wide-area testbed that supports the development of distributed systems and networks services. In all our experiments, we use 50 nodes that work as the clients, and one node to act as server.

We use an initial input text file of 1 GB, divided into 100 chunks (one 10 MB chunk per map task), in the word count and inverted index experiments. For the N-Gram application, we use 100 5 MB chunks, due to the larger size of intermediate files.

The applications we use to evaluate our system are packaged by many MapReduce implementations as benchmarks, and thus have the characteristics expected of data-intensive jobs. We describe each of them in turn.

3.1.1 Word Count

The word count application is a widely accepted benchmark in MapReduce implementations. Each map task receives a file chunk as input, counts the number of words in it and outputs an intermediate file with "word 1" pairs for each word found. The reduce step collects all the map intermediate outputs and aggregates them into one final output.

3.1.2 Inverted Index

This is another typical benchmark of MapReduce systems, in which the final output lists all the documents each word belongs to. The map task parses each chunk, and

emits a sequence of "word document ID" pairs. The reduce task merges all pairs for a given word, and emits a "word list(document ID)" pair.

3.1.3 N-Gram.

An N-Gram is a contiguous sequence of N items from a given input. Its output can be used in various research areas, such as statistical machine translation or spell checking. In our case, it is useful to extract text patterns from large size text and give statistical information on patterns' frequency and length.

Each map task receives its corresponding file chunk as input, and counts all sequences of words of length 1 to N. In our case, we defined N as 2, for reasons explained in the next subsection. As in the previous applications, a map output produces a "sequence 1" pair for each sequence with 1 or 2 words. The reduce step collects all the map intermediate outputs and aggregates all coinciding sequences into one final count, thus producing a pattern frequency result.

The map task from the two previous applications produces output files that are a little larger than the initial input. Therefore, the amount of data uploaded by mappers is similar to the volume downloaded as input. On the other hand, for each 5 MB input, N-Gram's map task creates around *30 MB* of intermediate files which must be transferred to reducers. Therefore, N-Gram is helpful in assessing the performance of our system with applications with large intermediate files.

3.1.4 Limitations of PlanetLab

Each node in PlanetLab may be shared by multiple virtual machines (slivers) at any time. For obvious reasons, users do not have access to slivers they do not own, and cannot predict when they will be executed. As such, our experiments were occasionally influenced by other slivers running at the same time. This was especially notorious in the node acting as the server, as the network bandwidth could reduce suddenly and drastically. We were able to identify these incorrect experiments due to their unusually long execution time, and through the use of HTTP commands to occasionally download files from the server.

PlanetLab has another significant limitation: disk space. Each node running our virtual machine has access to 8 GB in disk, which is very limited for our purpose. This was especially true when running a normal VC server, which has to hold the initial map input, map output files and the final reduce output. For this reason, we confined our input size to 1 GB for word count and inverted index, and 500 MB for N-Gram.

3.2 Experimental Results

Throughout this section, we refer to the existing Volunteer Computing system we use for comparison as *VCS*. We run tests on two versions of our system: *VMR* corresponds to our unmodified prototype; whereas *VMR-NH* is a VMR version that does not use hashes for map outputs.

The VMR-NH client returns the map output files to the server, exactly as the VCS clients. However, both VMR-NH and VMR use inter-client transfers, while all communication goes through the server in VCS. VMR-NH allows us to assess the impact of uploading and downloading map outputs from the central server, independently. As we mentioned previously, all our experiments are run with 50 clients.

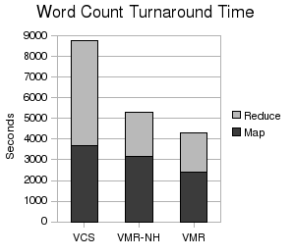

Figure 2: Turnaround Time of Word Count Application.

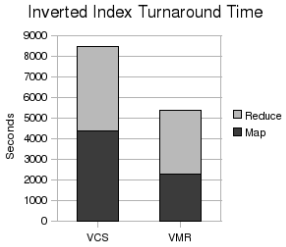

Figure 3: Turnaround Time of Inverted Index Application.

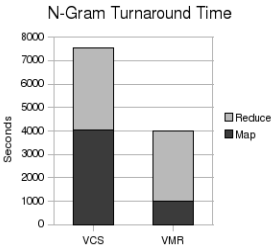

Figure 4: Turnaround Time (N-Gram).

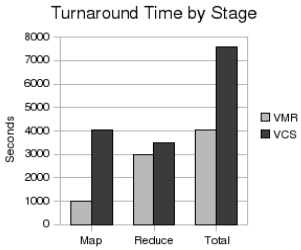

Figure 5: Turnaround Time by Stage (N-Gram).

3.3 Application Turnaround

We begin by measuring application turnaround on all three experiments. We measure the time it took each MapReduce job to finish, starting from the initial download of map input files, and ending with the upload of the last reduce output. We separate the map and reduce steps in order to identify their respective weight in regards to the overall application turnaround time. The map stage is considered to be finished once all its output has been validated in the server.

We run the word count application with VMR, VMR-NH and VCS. The turnaround time of the word count application for the three alternatives is shown in Fig. 2. For this application, the scenario with VMR clients has the lowest turnaround, followed by VMR-NH and then VCS. Both VMR and VMR-NH perform considerably better than VCS in the reduce step, with taking only 40% of VCS's time. This speedup can be attributed to the inter-client transfers, which reduce the communication with the central server. On the map stage, VMR-NH and VCS have similar results, with VMR-NH performing marginally better. This was expected as both clients download map inputs and return its output files to the server. The use of hashes yields a considerable improvement on the map step, with VMR reducing its execution interval to 65% of VCS's value. When considering the full job turnaround, with map and reduce execution, VMR is able to cut the time required by existing VC systems in more than half.

The results of the Inverted Index application support our previous findings, as we can see in Fig. 3. VMR's map tasks finished almost twice as fast as VCS. The reduce step is also faster with VMR, with an overall speedup of 1.25.

In the N-Gram experiments the server had to be deployed

on a faster node in order to make sure the jobs finished. The results obtained with N-Gram are shown in Fig. 4. The first conclusion we can gather from a first look at the graph is that VMR is able to finish the MapReduce job in half the time of VCS. This is consistent with previous results from the word count application. However, in this experiment we can also observe that the reduce stage on VMR is only slightly faster than VCS (Fig. 5). This can be explained by the better network connection of the node used as server specifically for this application. Despite its larger bandwidth, inter-client transfers still perform better than the centralized system. On the other hand, the differences in the map step are, as expected, much more significant. VMR is 4 times faster in executing the map stage, which translates to just a quarter of time needed by VCS to validate all its map tasks. This result shows us that VMR performs better with applications that create large intermediate files.

3.4 Network Traffic

We measure upload and download traffic in the server, for VMR and VCS while running the applications. Monitoring the network traffic on the server provides a more accurate measure of its overhead. It also allows us to quantify the impact of our solution concerning the decentralization of the VC model. We present the amount of data downloaded from the clients by the server, as well as the amount uploaded by the server to the clients.

Figure 6 shows the results for download traffic in the server while running the word count application. We include the results of VMR-NH to show the difference between returning hashes (VMR) or map outputs (VMR-NH). Since VMR-NH clients must return the map output back to the server, exactly as the regular VCS clients, the server has to download the same amount of data from the clients in both cases. This is clearly shown in Fig. 6, where we can see VMR-NH reaching the same value as VCS. VMR, on the other hand, has almost completely eliminated server downloads through the use of hashes. The VMR server receives a mere 250 MB from clients, a value 10 times smaller than VCS's 3GB.

The server upload traffic is presented in Fig. 7. We can see that up until around second 2000, both the VCS and the VMR server send the required map input files to the clients. However, once that step is completed, the VMR server is no longer responsible for uploading map outputs to reducers, unlike VCS which holds that responsibility. That explains the steep increase in the VCS line, around second 5000. Our server is required to upload 2.5 GB, to clients whereas VCS uploads more than double that amount.

The inverted index application experiments yielded very similar results, so they are not shown here. VMR is able to reduce the amount of data sent to to clients from 6.5 GB to 2.3 GB and cut downloaded data by 96%.

N-Gram presents a different scenario from the two other applications, so it is worthwhile to analyze its results. The upload traffic for a server running N-Gram is shown in Fig. 8. Note that, as mentioned in the previous section, GiGi-MR has a much lower application turnaround than VCS. This is why the GiGi-MR line in Fig. 8 stops around second 3000 (the same happens in Fig. 9), while VCS only finishes its execution much later. It is clear that there is a significant difference in the amount of data uploaded by GiGi-MR and VCS. This is due to the large size of intermediate files, which causes the VCS server to send almost 5 times more data to

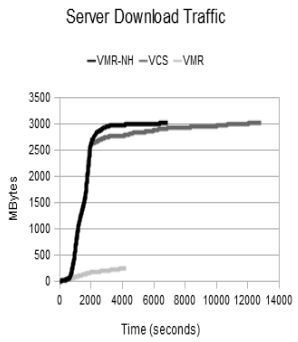

Figure 6: Server download traffic (Word Count).

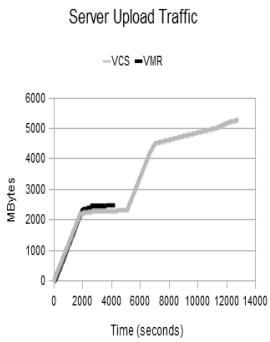

Figure 7: Server upload traffic (Word Count).

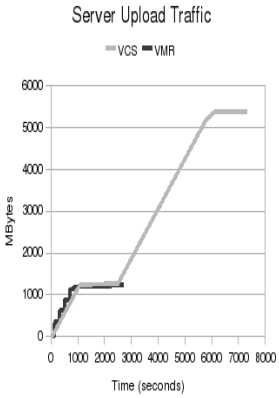

Figure 8: Server upload traffic (N-Gram).

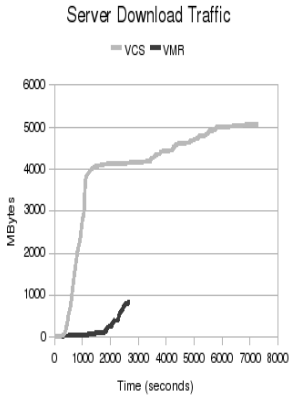

Figure 9: Server download traffic (N-Gram).

the clients than the VMR server in the reduce step.

The server's download traffic is exhibited in Fig. 9. Here, we can see the benefits of using hashes for map task validation. Up until second 2000, the VMR server has received almost no data from the clients. At around that time in the experiment, reducers that finished their task began sending their output back to the server. The VMR server downloads a total of 820 MB from the clients. On the other hand, the VCS server is responsible for downloading all map outputs from mappers, which corresponds to the steep increase up until second 4000. The VCS server is required to download 6 times more data than VMR. Therefore, we can conclude that VMR not only can perform better than VCS when running jobs with large intermediate files, but is also able to alleviate the server's network connection.

4. RELATED WORK

Combining the concepts of Cloud and VC was proposed in [11], in which the authors studied the cost and benefits of using clouds as a substitute for volunteers or servers.

In [14], the authors define a P2P model under the MapReduce framework. Their system is tailored to a dynamic cloud environment, creating a federation or cluster of data centers through a P2P overlay network.

MOON (MapReduce On opportunistic eNvironments) [13]

proposes an extension to the Hadoop[4] project that implements adaptive task scheduling in order to account for node failure. However, MOON is tailored for a cluster environment, such as a research lab, in which nodes are trusted or even dedicated.

MapReduce was also adapted to desktop grids in [15]. The authors claim it is able to run MapReduce jobs on XtremWeb [3], over the Internet. However, their experiments were conducted in a cluster interconnected by Gigabit Ethernet. This environment more closely resembles a common scenario of XtremWeb, which consists of a federation of research labs.

BOINC, on the other hand, has millions of users, and is actually tailored for a truly volunteer environment over the Internet. This allows us to state with more certainty what are the advantages and shortcomings of this paradigm on a Volunteer Computing environment.

5. CONCLUSION

We have presented VMR, a Volunteer Computing platform that leverages client resources in order to execute MapReduce applications over the Internet. Our system is able to tolerate volunteer faults, and transient server failures. Furthermore, it is compatible with existing VC systems (in particular BOINC). VMR significantly reduces the dependence on the central server, which is typically overburdened in current VC platforms, thus allowing it to obtain a better performance.

We evaluated VMR by measuring the application turnaround, server network traffic and overhead while running three different MapReduce applications. Our solution was able to improve the performance of all the MapReduce jobs we tested. The map stage was up to 4 times faster than in an existing VC system. The reduce step also showed an improvement, thus reducing each MapReduce job's execution time down to less than half.

Regarding the server's network traffic, VMR reduced server download traffic by an order of magnitude on the word count and inverted index applications. The N-Gram application provided a different scenario, due to its large intermediate data. Therefore, we were able to witness a decrease in uploaded data to 20% of the existing VC system server's value.

We were able to conclude that VMR not only performs better than VCS when running MapReduce jobs, but is also able to significantly remove the burden on the server's network connection.

6. ACKNOWLEDGMENTS

This work was partially supported by national funds through FCT - Fundação para a Ciência e a Tecnologia, under projects PTDC/EIA-EIA/102250/2008, PTDC/EIA-EIA/108963/2008, PTDC/EIA-EIA/113613/2009, and PEst-OE/EEI/LA0021/2011.

7. REFERENCES

[1] D. P. Anderson. Boinc: A system for public-resource computing and storage. In *Proc. of the 5th IEEE/ACM Int'l Workshop on Grid Computing*, GRID '04, pages 4–10, Washington, DC, USA, 2004.

[2] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer. Seti@home: an experiment in public-resource computing. *Commun. ACM*, 45:56–61, November 2002.

[3] F. Cappello, S. Djilali, G. Fedak, T. Herault, F. Magniette, V. Néri, and O. Lodygensky. Computing on large-scale distributed systems: Xtremweb architecture, programming models, security, tests and convergence with grid. *Future Gener. Comput. Syst.*, 21:417–437, March 2005.

[4] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman. Planetlab: an overlay testbed for broad-coverage services. *SIGCOMM Comp. Commun. Rev.*, 33:3–12, July 2003.

[5] W. Cirne, F. Brasileiro, N. Andrade, L. Costa, A. Andrade, R. Novaes, and M. Mowbray. Labs of the world, unite!!! *Journal of Grid Computing*, 4:225–246, 2006.

[6] I. Cisco. Cisco visual networking index: Forecast and methodology, 2011−2016. *CISCO White paper*, pages 2011–2016, February 2012.

[7] F. Costa, I. Kelley, L. Silva, and G. Fedak. Optimizing data distribution in desktop grid platforms. *Parallel Processing Letters*, 18(3):391–410, September 2008.

[8] F. Costa, L. Silva, G. Fedak, and I. Kelley. Optimizing the data distribution layer of boinc with bittorrent. *Parallel and Distributed Processing Symposium, International*, 0:1–8, 2008.

[9] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51:107–113, Jan. 2008.

[10] G. Fedak, H. He, and F. Cappello. Bitdew: a programmable environment for large-scale data management and distribution. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, SC '08, pages 45:1–45:12, Piscataway, NJ, USA, 2008.

[11] D. Kondo, B. Javadi, P. Malecot, F. Cappello, and D. P. Anderson. Cost-benefit analysis of cloud computing versus desktop grids. In *Proceedings of the 2009 IEEE International Symposium on Parallel&Distributed Processing*, IPDPS '09, pages 1–12, Washington, DC, USA, 2009.

[12] S. M. Larson, C. D. Snow, M. Shirts, V. S. P, and V. S. Pande. Folding@home and genome@home: Using distributed computing to tackle previously intractable problems in computational biology. *Computational Genomics*, 2002.

[13] H. Lin, X. Ma, J. Archuleta, W.-c. Feng, M. Gardner, and Z. Zhang. Moon: Mapreduce on opportunistic environments. In *Proc. of the 19th ACM Int'l Symposium on High Performance Distributed Computing*, HPDC '10, pages 95–106, New York, NY, USA, 2010.

[14] F. Marozzo, D. Talia, and P. Trunfio. Adapting mapreduce for dynamic environments using a peer-to-peer model. In *Proc. of the First Workshop on Cloud Computing and its Applications (CCA 2008)*, Chicago, USA, October 2008.

[15] B. Tang, M. Moca, S. Chevalier, H. He, and G. Fedak. Towards mapreduce for desktop grid computing. In *Proceedings of the 2010 International Conference on P2P, Parallel, Grid, Cloud and Internet Computing*, 3PGCIC '10, pages 193–200, Washington, USA, 2010.

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