SCADAMAR: Scalable and Data-Efficient Internet MapReduce

Rodrigo Bruno
INESC-ID
Instituto Superior Técnico
Lisboa, Portual
rodrigo.bruno@tecnico.ulisboa.pt

Paulo Ferreira
INESC-ID
Instituto Superior Técnico
Lisboa, Portual
paulo.ferreira.inesc-id.pt

ABSTRACT

Recent developments of popular programming models, namely MapReduce, have raised the interest of running MapReduce applications over the large scale Internet. However, current data distribution techniques used in Internet wide computing platforms to distribute the high volumes of information, which are needed to run MapReduce jobs, are naive, and therefore need to be re-thought.

Thus, we present a computing platform called SCADAMAR that runs MapReduce jobs over the Internet and provides two new main contributions: i) improves data distribution by using the BitTorrent protocol to distribute all data, and ii) improves intermediate data availability by replicating tasks or data through nodes in order to avoid losing intermediate data and consequently preventing big delays on the MapReduce overall execution time.

Along with the design of our solution, we present an extensive set of performance results which confirm the usefulness of the above mentioned contributions, improved data distribution and availability, thus making our platform a feasible approach to run MapReduce jobs.

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: General—*Data communications*; C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Distributed applications*; C.4 [**Performance of Systems**]: Reliability, availability, and serviceability

General Terms

Algorithms, Performance, Realiability

Keywords

Volunteer Computing, BitTorrent, BOINC, MapReduce

1. INTRODUCTION

With the ever growing demand for computational power, scientists and companies all over the world strive to harvest more computational resources in order to solve more and bigger problems in less time, while spending the minimum money possible. With these two objectives in mind, we think of utilizing available resources all over the world through the Internet as a viable solution to access huge amounts of computational resources.

With time, more computing devices (PCs, gaming consoles, tablets, mobile phones, ...) join the network. By aggregating all these resources in a global computational pool, it is possible to obtain huge amounts of computational resources that would be impossible, or impractical, for most grids, supercomputers, and clusters. For example, recent data from BOINC [2], a Volunteer Computing project, shows that, currently, there are 50 supported projects sustained by an average computational power of over 7 PetaFLOPS[1].

In addition, recent developments of popular programming models, namely MapReduce[2] [6], have raised the interest of using MapReduce applications over the Internet. Although increasing its attractiveness, it also brings the need to rethink and evolve current computing platforms' architectures, in particular, data distribution, to adapt to these new programming models.

Hence, we present a system called SCADAMAR, a computing platform which has as its ultimate goal to aggregate as many computational resources as possible in order to run MapReduce jobs in a scalable and efficient way, over the Internet. To be successful, SCADAMAR must fulfill several key objectives: i) scale up with the number of nodes, ii) collect nodes' resources such as CPU cycles, network bandwidth, and storage in an efficient way, iii) tolerate byzantine and fail-stop failures, and iv) support MapReduce, a particularly interesting programming model, given its relevance for a large number of applications.

MapReduce applications commonly have two key features: i) depend on large amounts of information to run and ii) may run for several cycles (where data can be processed and transformed several times). Therefore, in order to take full advantage of the MapReduce model, SCADAMAR must be able to distribute large amounts of data efficiently while allowing applications to perform several MapReduce cycles without compromising its scalability.

In order to understand the challenges inherent to building a solution like SCADAMAR, it is important to emphasize its target environment: computation pools. Computation pools are made of arbitrary computers, owned by individuals, institutions, or cloud providers around the world. Nodes

[1]Statistics from boincstats.com
[2]MapReduce is a popular programming model composed by two operations, Map and Reduce

have variable Internet connection bandwidth, node churn is very high (compared to clusters or grids), nodes are very asymmetrical in hardware and software and their computing resources may be focused on user tasks. Finally, as opposed to grids and clusters, nodes cannot be trusted since they may be managed by malicious users.

By considering all the available solutions, it is important to note that solutions based on clusters and/or grids do not fit our needs. Such solutions are designed for controlled environments where node churn is expected to be low, where nodes are typically well connected with each other, nodes can be trusted, and are very similar in terms of software and hardware. Therefore, solutions like HTCondor [16], Hadoop [17], XtremWeb [8] and other similar computing platforms are of no use to attain our goals.

When we turn to platforms that harness resources all over the Internet, we observe that most existing solutions are built and optimized to run Bag-of-Tasks applications. Therefore, solutions such as BOINC, GridBot [14], Bayanihan [13], and many others [1, 3, 4, 12] do not support the execution of MapReduce jobs, which is one of our main requirements.

With respect to the few solutions [11, 15, 7, 5] that support MapReduce, we are able to point out some issues (more details on Section 4): data distribution could be improved, intermediate data availability is overlooked, and lack of support for multiple cycle MapReduce applications.

To solve the aforementioned problems, we present SCADAMAR, a BOINC compatible computing platform that enables the deployment of MapReduce applications over the Internet. Besides supporting MapReduce jobs, SCADAMAR goes one step further by allowing nodes (mappers or reducers) to help distributing both the input, intermediate output and final output data, through the BitTorrent[3] protocol and not point-to-point protocols (such as HTTP and FTP). SCADAMAR will therefore benefit from nodes' network bandwidth to distribute data. Additionally, SCADAMAR allows sequences of MapReduce cycles to run without having to wait for the data server, i.e., data can flow directly from reducers to mappers (of the next cycle). Regarding the availability of intermediate data, SCADAMAR proposes an enhanced work scheduler that automatically replicates data or tasks to minimize the risk of loosing intermediate data.

By providing these functionalities, SCADAMAR achieves higher scalability (reducing the burden on the data server), reduced transfer time (improving the overall turn-around time), and augmented fault tolerance (since nodes can fail during the transfer without compromising the data transfer).

In short, the contributions of this work are the following: i) a BOINC compatible computing platform; ii) enhanced data distribution for MapReduce workflows using BitTorrent; iii) enhanced intermediate data availability using data and task replication.

To conclude, we envision that our project could be of great, use namely for cloud providers. Using SCADAMAR, a cloud provider could decide whether to use its own resources, or to export some MapReduce computation to other cloud providers, or even to some individual/volunteer desktop. Such cloud provider could use some resource market solution (that is out of the scope of this paper) that could enable users to choose where to place the computation depending, for example, on its price.

[3]Official BitTorrent specification can be found at www.bittorrent.org

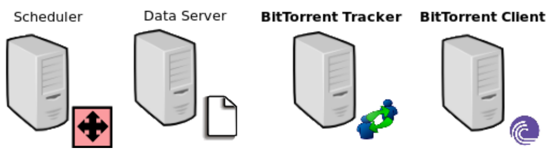

Figure 1: Server-Side Architecture

2. SCADAMAR

Along this section, we start by describing how and which techniques are used to achieve the proposed goals. Next, we present our data distribution algorithms (used to provide fast and scalable data transfers in the context of MapReduce jobs). We finish this section with some techniques that address the availability problem of intermediate data and some implementation notes.

2.1 Architecture Overview

SCADAMAR is divided into two major components: client and server. The server is responsible for delivering and managing tasks (work units) while the client is responsible for executing and reporting results. At a very high level, the server is composed by a scheduler (that manages and delivers tasks) and by a data server (that stages the input and the output of jobs).

BitTorrent is a peer-to-peer protocol that allows nodes around the world to exchange data. In our project, we use it to spread input, intermediate and output data. Using BitTorrent, the central server and all client nodes become part of several swarms, networks of nodes that share a particular file. This way, nodes can download each file from multiple sources at the same time (therefore alleviating the burden on the central server).

In order to introduce BitTorrent as the data distribution protocol, two new entities are added to the central server (see Figure 1): a BitTorrent Tracker and a BitTorrent Client. The tracker is essential to provide information so that new nodes are able to find others that have a particular file. The BitTorrent client (which is also added to all clients) is responsible for handling the BitTorrent protocol and all the necessary communication to perform the transfers.

At a very high level, our approach to introduce MapReduce in SCADAMAR can be described as follows: i) the client side executes the map tasks and reports hashes of the output files to the central server; ii) the central server validates the hashes and then shuffles intermediate data; iii) reduce tasks run and the results are sent to the central server.

The shuffle phase is a necessary step in the MapReduce workflow. As each mapper might produce output to every reducer, there is the need to organize the inputs for the reduce tasks. Figure 2 shows this operation. This operation is performed by the central server once it detects that all map tasks are validated. It is important to note that this operation manages hash files (known as `.torrent` files). This way, the shuffle phase is extremely fast and the scalability of the system is not compromised. Once this operation is finished, new reduce tasks can be delivered to new nodes.

2.2 Data Distribution Algorithm

We now detail how we use the BitTorrent file sharing protocol to coordinate input, intermediate and final output data transfers. Still on our data distribution algorithm, we show how SCADAMAR is able to run multiple MapReduce cy-

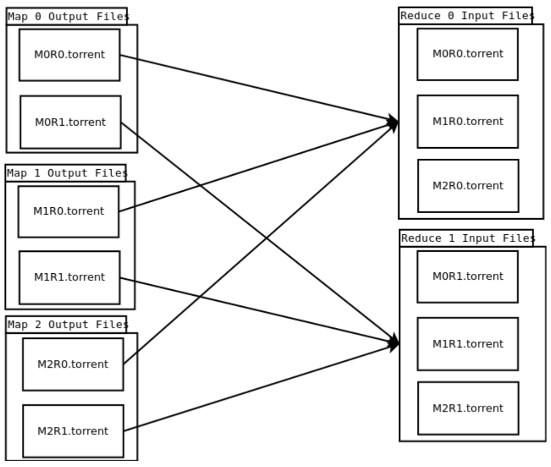

Figure 2: Shuffle Phase Example

cles without compromising its scalability (i.e. avoiding high burden on the data server).

2.2.1 *Input Distribution*

Input distribution is the very first step in every MapReduce application. Input must be split over multiple mappers. To do so, each mapper downloads an input file hash (the `.torrent` file) that identifies an input file at the central data server.

For each input file, the server plays as original seed. If we take into consideration that each map task is replicated over at least three nodes (for replication purposes), then, when a new map task begins, the node will have at least one seed (the data server) and possibly up to the task replication factor minus one, additional nodes sharing the file (each node shares all the input file chunks that it has using the BitTorrent protocol).

Therefore, we can leverage the task replication mechanisms to share the burden of the data server. Even if the server is unable to respond, a new mapper may continue to download its input data from other mappers. The transfer bandwidth will also be bigger since a mapper may download input data from multiple sources in parallel.

2.2.2 *Intermediate Output Distribution*

Figure 3 illustrates the steps for the intermediate data distribution. Once a map task is finished, the mapper has an intermediate output ready to be used. The first step is to create a hash of the output (1). Then, the BitTorrent client, running at the client node, automatically contacts the BitTorrent tracker (2), running at the central server, to report new data (the hash of the output is used as an identifier). The third step is to make the server aware of the map task finish. To this end, the mapper contacts the server and sends the hash file just created for the intermediate output (3).

As more intermediate file hashes arrive at the server, the server is able to decide (using a quorum of results) which mappers have the correct intermediate files (by comparing all hashes). When all the intermediate outputs are available and validated, the server shuffles all hash files and saves them at the data server(4 and 5). When new nodes request work, the scheduler starts issuing reducer tasks (6). These reducer tasks contain references to the hash files that were successfully validated and that need to be downloaded (7).

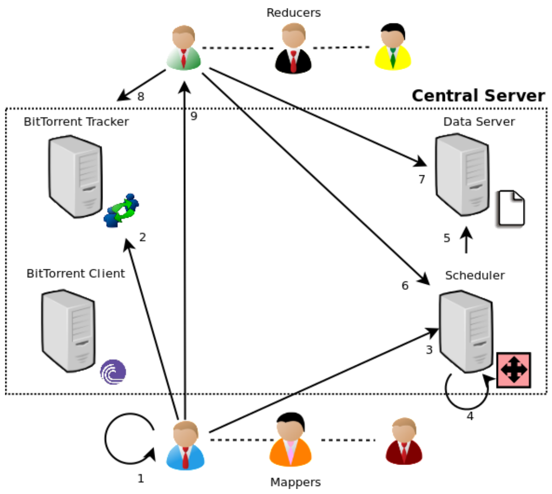

Figure 3: Intermediate Data Distribution

Once a reducer has access to these hash files, it contacts the BitTorrent Tracker (using the hashes as file identifiers) to know which nodes have the intermediate data (8), and starts transferring intermediate files (using the BitTorrent protocol) from all the nodes reported by the tracker (9). Reduce tasks start as soon as all the needed intermediate values are successfully transfered.

2.2.3 *Output Distribution*

Given that reduce tasks are replicated over at least three nodes, it is possible to accelerate the upload of the final output files from reducers to the data server.

The procedure is similar to the one used for intermediate outputs. Once a reduce task finishes, the reducer computes a hash for its fraction of the final output. Then, it informs the BitTorrent Tracker that some output data is available at the reducer node. The next step is to send a message to the central scheduler containing the hash of the output file and acknowledging the task completion. Once the scheduler receives enough results from reducers, it can proceed with validation and decide which hashes will be used to download the final output. All the trustworthy hashes are then used by the BitTorrent client at the central server to download the final output.

Using BitTorrent to transmit the final outputs results in a faster transfer from client nodes to the data server, a lower and shared bandwidth consumption from the client's perspective, and an increased fault tolerance (since a client node failure will not abort the file transfer).

2.2.4 *Multiple MapReduce Cycles*

Using the data distribution techniques just described, where the central server and all client nodes have a BitTorrent client and use the BitTorrent Tracker to find peers with data, it is very easy to use SCADAMAR for running applications that depend on multiple MapReduce cycles. The difference between our solution and previous ones (namely SCOLARS) is that output data need not to go to the central server before it is delivered to new mappers (i.e., data can flow directly from reducers to mappers from one cycle to the other).

From mappers's perspective, when a workunit is received, the BitTorrent tracker is asked for nodes with the required data. It does not differentiate between the single cycle scenario (where mappers download from the central server) or the multiple cycle scenario (where mappers download from reducers of the previous cycle). Regarding the central server's perspective, the scheduler only needs to know that some map tasks depend on the output of some reduce tasks.

2.3 Availability of Intermediate Data

Previous studies [10] show that the availability of intermediate data is a very sensitive issue for programming models like MapReduce. Note that, when using embarrassingly parallel applications, there is no intermediate data and therefore this problem does not apply.

The problem is that, for performance reasons, typical MapReduce implementations (targeted to clusters) do not replicate intermediate results. However, when applied (MapReduce) to Internet wide computing, where node churn is very high, such lack of replication leads to a loss of intermediate data. It was shown that loosing a single chunk of intermediate data incurs into a 30% delay of the overall MapReduce execution time. To cope with this problem, SCADAMAR presents two methods:

1. replicate map tasks aggressively when nodes designated to execute a particular map task take too long to answer the central server probes. By imposing a shorter interval time to report to the central server, we make sure that we keep at least a few replicas of the intermediate output. As soon as a mapper is suspected to be failing, a new map task will be delivered to replicate the failed one;

2. replicate intermediate date when there are intermediate outputs that have already been validated (by the central server) and some of the mappers that reported these results take too long to answer to the central server probes. Therefore, nodes might be used to replicate intermediate data to compensate other mappers that die while waiting for the reduce phase to start. These tasks would simply download hash files and use them to start downloading intermediate data. Once the reduce phase starts, these new nodes can also participate in the intermediate data distribution phase, just like the mappers that performed the map tasks.

Replicating only the intermediate output is much faster than replicating a map task since: i) the computation does not have to be performed; ii) intermediate data is normally smaller than input data. These two methods are applied before the reduce stage starts.

It is important to notice that if a map task is not validated, it is not be safe to replicate the intermediate output. If there is some intermediate output available (but not validated), replicating it would possibly replicate erroneous data which would get to wrongly validated data.

During the reduce stage, reducer nodes that take too long to answer the central server probes will also be replaced by other nodes until the computation is finished.

2.4 Implementation

SCADAMAR is implemented on top of BOINC. We did not change BOINC's core implementation since it would be impossible to have a single BOINC server hosting MapReduce and non MapReduce projects at the same time. In fact, SCADAMAR is implemented via alternative implementations of some project specific daemons, namely the daemon responsible for creating tasks and the one responsible for validating tasks.

Benchmark	BOINC	SCOLARS	SCADAMAR
grep	1500 sec	1425 sec	610 sec
Word Count	2121 sec	1590 sec	578 sec
Terasort	3482 sec	2100 sec	686 sec

Table 1: Benchmark Execution Times

To be and remain compatible with current BOINC clients, our project is implemented as a regular BOINC application. Therefore, all client nodes will be able to join a MapReduce computation without upgrading their client software. If it would not be a regular application, clients would have to upgrade their client software in order to fully explore SCADAMAR' capabilities namely, use BitTorrent to share files. Previous solutions (e.g. SCOLARS) do not use this approach and modify the BOINC client. Therefore, it cannot be used without forcing users to upgrade their client software.

SCADAMAR's client side application is meant to be used as a framework, i.e., developers would simple call SCADAMAR's code to register the map and reduce functions. All other issues related to managing map and reduce tasks execution, downloading and uploading data is handled by our implementation.

Notwithstanding, application developers might analyse and adapt the application code to specific application needs (e.g. if one needs to implement a special way to read/write input/output data). Other optimizations like intermediate data partitioning or combining intermediate results might be easily implemented as well.

3. EVALUATION

We now proceed with an extensive evaluation of our system. We compare SCADAMAR with SCOLARS (a BOINC compatible MapReduce computing platform), and BOINC, one of the most successful VC platforms, and therefore, a good reference for performance and scalability comparison. We use several representative benchmark applications and different environment setups that verify the performance and scalability of SCADAMAR.

To conduct out experiments, we used 60 university lab nodes, all very similar both in hardware and software. To create a more realistic environment, all tasks run with the lowest possible priority and had their upload bandwidth limited to 10Mbps (except for the experiment in Table 3). All MapReduce workflows used 16 map tasks and 4 reduce tasks replicated three times each.

3.1 Application Benchmarking

We start our evaluation with some typical MapReduce application benchmarks. It is important to understand that these benchmarks are IO intensive and not CPU intensive (as all Internet wide applications should be). We decided to use them to stress even more the data distribution performance. An input file of 512MB was used. By Table 1, it is possible to conclude that SCADAMAR outperforms both BOINC and SCOLARS by a vast margin.

For the next experiments, whenever it is not said in contrary, we are using the word count benchmark to produce results.

3.2 Varying the Input File Size and the Upload Bandwidth

File Size	BOINC	SCOLARS	SCADAMAR
256 MB	1150 sec	840 sec	340 sec
512 MB	2121 sec	1590 sec	578 sec
1024 MB	4127 sec	3100 sec	1020 sec
2048 MB	8951 sec	6240 sec	1960 sec

Table 2: Performance Varying the Input File Size

Bandwidth	BOINC	SCOLARS	SCADAMAR
5 Mbps	3974 sec	3022 sec	983 sec
10 Mbps	2121 sec	1590 sec	578 sec
25 Mbps	761 sec	553 sec	321 sec
50 Mbps	451 sec	304 sec	241 sec
100 Mbps	281 sec	165 sec	209 sec

Table 3: Performance Varying the Upload Bandwidth

In this next experiments, we change the input file size and the upload bandwidth to see how the platforms react. According to Tables 2 and 3, SCADAMAR is the system with better performance, the one that degrades the performance in a smoother way when the input file size increases and when the upload bandwidth decreases.

It is also interesting to note that for high upload bandwidths, some BitTorrent overhead starts notice. As the used file only has 512MB, for high bandwidths, the latency of contacting multiple nodes and the tracker and the time lost waiting in queues starts to notice.

3.3 Iterative MapReduce Applications

For this experiment we use another MapReduce application, one that benefits from multiple cycles: page rank. Results are shown in Table 4. SCADAMAR is clearly the best platform for applications with multiple cycles as it is approximately 10 times faster than BOINC and 5 times faster than SCOLARS.

It is important to note that the big performance discrepancy between BOINC and SCOLARS is caused by the way both systems handle intermediate data. As page rank's intermediate data is almost 50% bigger than input data, BOINC looses much time moving data to and from the data server.

3.4 Varying the Map Replication Factor

We now turn our attention to the intermediate data availability problem. With this experiment, we show an interesting fact that is confirmed by our results: SCADAMAR, that uses BitTorrent, does not have performance degradation when the map task replication factor is increased. Results from Table 5 confirm it.

The cause for this resides in the BitTorrent protocol. Since each group of replicas share the same file, the server can, in theory, send only one copy of each chunk of the file for each replica. All the replicas can then exchange chunks until all of them have all the chunks. By design, the BitTorrent protocol converges to this scenario.

Phase	BOINC	SCOLARS	SCADAMAR
Map 1	1937 sec	1879 sec	613 sec
Reduce 1	2807 sec	600 sec	148 sec
Map 2	1940 sec	1900 sec	142 sec
Reduce 2	2801 sec	603 sec	148 sec
Total	9485 sec	4982 sec	1051 sec

Table 4: Performance for Two Page Ranking Cycles

Repl. Factor	BOINC	SCOLARS	SCADAMAR
2	1575 sec	1225 sec	600 sec
3	2121 sec	1766 sec	591 sec
4	2600 sec	2220 sec	590 sec
5	3128 sec	2771 sec	621 sec
6	3576 sec	3228 sec	600 sec

Table 5: Varying the Map Task Replication Factor

Session Time	1 vol/min	10 vol/min	100 vol/min
1 hour	INF	2710 sec	2282 sec
2 hour	3922 sec	2527 sec	2114 sec
3 hour	3880 sec	2398 sec	2104 sec
4 hour	3754 sec	2354 sec	2104 sec
5 hour	3754 sec	2354 sec	2104 sec

Table 6: Real World Simulations

3.5 Real World Environment

In this last experiment we test our solution on a simulated realistic environment, i.e., with nodes coming and going in the middle of a MapReduce job. To that end, we used different churn rates by manipulating the average node session time and the new node rate.

Results from Table 6 shows that the time needed to complete a job is reduced by: i) increasing the node session time and ii) increasing the new node rate. The interesting result is found when the session time is 1 hour and there is one new node per minute. Our results show that there are not enough nodes and nodes fail too often to complete the job. For those reasons, most of our runs never ended.

From this experiment, we conclude that, as intermediate data is kept on volatile nodes, one should make sure that the job completion time is, at most, equal to the average node session time. Therefore, when running MapReduce on top of computation pools, one should use many small jobs (possibly with multiple cycles) instead of few large jobs.

4. RELATED WORK

In this section we analyse and discuss the systems that we found to be close to SCADAMAR. From all the existent Internet wide computing platforms, most of them are focused and optimized to run Bag-of-Tasks (embarrassingly parallel) applications and thus, cannot support MapReduce applications. Nevertheless, as MapReduce's popularity increased, some platforms decided to use available resources over the Internet to run MapReduce jobs. Therefore, solutions such as SCOLARS [5], MOON [11], Tang [15] and Marozzo [7] already support MapReduce applications.

MOON (MapReduce On Opportunistic Environments) is an extension of Hadoop (an open source implementation of MapReduce). MOON ports MapReduce to opportunistic environments mainly by: i) modifying both data and task scheduling (to support two types of nodes, stable and volatile nodes), and ii) performing intermediate data replication. However, although MOON was designed to run MapReduce tasks on volatile nodes, it still relies on a large set of dedicated nodes (mainly for hosting dedicated data servers). This assumption does not hold in a pure volatile computing setting (where one could not garantee the availability of such dedicated resources).

The solution presented by Tang [15] is a MapReduce implementation focused on desktop grids. It was built on top of a data management framework, Bitdew [9]. Even though BitDew supports BitTorrent, the authors do not mentioned it. Moreover, there is no performance evaluation with other applications than word count and no performance evaluation in realistic environments.

Marozzo [7] presents a solution to exploit the MapReduce model in dynamic environments. The major drawbacks of this solution are: i) data is distributed point-to-point (which fails to fully utilize the node's bandwidth), and ii) there is no intermediate output replication.

SCOLARS (Scalable Complex Large Scale Volunteer Computing) is a modified version of BOINC that supports MapReduce applications and presents two contributions: i) inter-client transfers (for intermediate data only), and ii) hash based task validation (where only a hash of the intermediate output is validated on the central server). However, it presents the same issues as the previous solution: only point-to-point transfers, and no intermediate data replication.

In conclusion, all the analysed solutions present problems that invalidate them as candidate solutions for the problem we are addressing. Additionally, no solution showed that it was able to fully utilize the node's upload bandwidth, and no solution showed that it was possible to run multiple MapReduce cycles, while avoiding a bottleneck on the data server.

5. CONCLUSIONS

SCADAMAR is a new, MapReduce enabled, computing platform. It presents a new data distribution technique for input, intermediate and final output using the BitTorrent protocol. Using BitTorrent, SCADAMAR is able to use node's upload bandwidth to help distributing data. Moreover, it presents an improved map task and data replication scheduler and is able to efficiently run MapReduce applications with multiple cycles.

From the experiments, we conclude that SCADAMAR is able to perform much better (performance and scalability wise) than current platforms. Hence, with our work, it is possible to improve MapReduce applications' execution time on large computation pools such as the Internet.

6. REFERENCES

[1] A. Alexandrov, M. Ibel, K. Schauser, and C. Scheiman. Superweb: towards a global web-based parallel computing infrastructure. In *Parallel Processing Symposium, 1997. Proceedings., 11th International*, pages 100–106, 1997.

[2] D. Anderson. Boinc: a system for public-resource computing and storage. In *Grid Computing, 2004. Proceedings. Fifth IEEE/ACM International Workshop on*, pages 4–10, 2004.

[3] A. Baratloo, M. Karaul, Z. Kedem, and P. Wijckoff. Charlotte: Metacomputing on the web. *Future Generation Computer Systems*, 15(5-6):559 – 570, 1999.

[4] A. Chakravarti, G. Baumgartner, and M. Lauria. The organic grid: self-organizing computation on a peer-to-peer network. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 35(3):373–384, 2005.

[5] F. Costa, L. Veiga, and P. Ferreira. Internet-scale support for map-reduce processing. *Journal of Internet Services and Applications*, 4(1):1–17, 2013.

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

[7] D. T. Fabrizio Marozzo and P. Trunfio. Adapting mapreduce for dynamic environments using a peer-to-peer model, 2008.

[8] G. Fedak, C. Germain, V. Neri, and F. Cappello. Xtremweb: a generic global computing system. In *Cluster Computing and the Grid, 2001. Proceedings. First IEEE/ACM International Symposium on*, pages 582–587, 2001.

[9] G. Fedak, H. He, and F. Cappello. Bitdew: A data management and distribution service with multi-protocol file transfer and metadata abstraction. *Journal of Network and Computer Applications*, 32(5):961 – 975, 2009. Next Generation Content Networks.

[10] S. Y. Ko, I. Hoque, B. Cho, and I. Gupta. Making cloud intermediate data fault-tolerant. In *Proceedings of the 1st ACM symposium on Cloud computing*, pages 181–192. ACM, 2010.

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

[12] V. Lo, D. Zappala, D. Zhou, Y. Liu, and S. Zhao. Cluster computing on the fly: P2p scheduling of idle cycles in the internet. In *Peer-to-Peer Systems III*, pages 227–236. Springer, 2005.

[13] L. F. Sarmenta and S. Hirano. Bayanihan: building and studying web-based volunteer computing systems using java. *Future Generation Computer Systems*, 15(5-6):675 – 686, 1999.

[14] M. Silberstein, A. Sharov, D. Geiger, and A. Schuster. Gridbot: execution of bags of tasks in multiple grids. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, SC '09, pages 11:1–11:12, New York, NY, USA, 2009. ACM.

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

[16] D. Thain, T. Tannenbaum, and M. Livny. Distributed computing in practice: the condor experience. *Concurrency and Computation: Practice and Experience*, 17(2-4):323–356, 2005.

[17] T. White. *Hadoop: the definitive guide.* O'Reilly, 2012.