2011 IEEE International Parallel & Distributed Processing Symposium

Volunteer Cloud Computing: MapReduce over the Internet

Fernando Costa, Luis Silva
CISUC, Dept. of Informatics Engineering
University of Coimbra, Portugal

Michael Dahlin
Laboratory for Advanced Systems Research
University of Texas Austin

***Abstract*— Volunteer Computing harnesses computing resources of machines from around the world to perform distributed independent tasks, following a master/worker model. Despite the recent increase in popularity and power in middleware such as BOINC, there are still several limitations in existing systems. Current research is oriented towards optimizing existing applications, while the number of active users and projects has reached a plateau.**

A programming paradigm that has been significantly popular and is used by several systems on the cloud is MapReduce. The main advantage of this paradigm is that it can be used to solve a vast amount of different problems, by breaking them into simple steps and taking advantage of distributed resources. Volunteer Computing provides these resources, and although it cannot match the conditions offered by a cluster, it has other advantages that can be leveraged. In this paper, we try to increase the computational power of Volunteer Computing systems by allowing more complex applications and paradigms such as MapReduce to be run, thus opening new avenues and possibilities for the use of computational devices scattered through the Internet.

We created a BOINC prototype that can run MapReduce jobs (BOINC-MR), using a pull-model in which communication is always initiated by the client. By running experiments on a small cluster, with multiple variables, we were able to evaluate a few initial scenarios with this paradigm. We used a simple MapReduce application, word count, as proof of concept, just to demonstrate a typical execution.

***Index Terms*—Cloud Computing, MapReduce, Volunteer Computing**

I. Introduction

VOLUNTEER Computing (VC) is a type of distributed computing in which research projects take advantage of volunteer resources donated by ordinary people spread throughout the Internet. The use of personal computers' computational power as a tool for science has steadily increased in popularity. Applications running on this infrastructure tackle problems from a wide range of scientific subjects, such as physics, molecular biology, or astronomy. These Desktop Grids are especially well suited for highly parallel computations that do not require any communication or coordination between network participants.

BOINC is the most popular software platform for Volunteer Computing, and is currently being used by many projects to analyze data, allowing it to rival the word's supercomputers in computing power. In the last decade, the VC community has grown significantly, with over 500 thousand active users, providing scientists with low cost petaFLOPS.

There are, however, worrying signs of a potential bottleneck in VC evolution, preventing it from reaching its true potential. In current implementations, the network topology is restricted to a strict master/worker scheme, usually with a small set of centrally managed project machines handling task and data distribution and retrieving results from users. This significant limitation – support exclusively embarrassingly parallel applications –, along with the notion that VC cannot be used for more complex scenarios or purposes has hampered its adaptability and use by new projects. The number of active users has reached a plateau, while scalability issues have also emerged in data distribution and storage [1].

In order to take full advantage of the potential at the disposal of VC platforms, one must first break free of existing limitations and constraints. This can be achieved by introducing new computing paradigms, and providing new mechanisms to support a whole new level of applications to run on Desktop Grids.

MapReduce is a broadly-useful computing paradigm that has recently gained prominence as robust, parallel implementations such as Google's MapReduce [2] and others [3] [4] have been widely used to handle data-intensive operations across clusters in data centers. It is able to represent a wide range of applications that can be broken down into map and reduce operations. Our goal is to adapt it to the insecure, unreliable VC environment, by taking advantage of the vast improvements in network infrastructure and disk storage in the last mile of the Internet.

In this paper, we developed a BOINC prototype that can run MapReduce jobs, and added inter-client data transfers to move data between map and reduce tasks. At this stage, we only ran experiments with a simple application, word count, as a proof-of-concept, to show its feasibility.

II. Volunteer Computing and MapReduce

This section introduces the system and paradigm we based our research on – BOINC and MapReduce – and discusses

This work was supported in part by the Fundação para Ciência e Tecnologia under Grant SFRH/BD/46034/2008, and UT Austin's Center for Information Assurance and Security.

1530-2075/11 $26.00 © 2011 IEEE
DOI 10.1109/IPDPS.2011.345

existing limitations and development opportunities.

Volunteer Computing harnesses computing resources of machines from around the world to perform distributed independent tasks, following a master/worker model. Desktop Grids have been extremely successful in gathering large numbers of donated computing cycles, to form a large-scale virtual supercomputer. The computing power provided by existing systems is considerable, with Folding@home [5] having reached 5 petaFLOPS and BOINC sustaining over 3,5 petaFLOPS, especially when considering the world's fastest supercomputer, Tianhe-1A, performs at 2,57 petaFLOPS.[6].

BOINC

One of the first and most popular VC projects was SETI@home [7], which has attracted over 5 million participants. Its success lead to the creation of a middleware platform, BOINC [8] – Berkeley Open Infrastructure for Network Computing –, to deal with all the complexities of running projects on distributed, unreliable resources.

Existing Desktop Grids such as BOINC or XtremWeb [9] have centralized architectures, in which a central server or coordinator is responsible for scheduling task execution, while taking into consideration specified deadlines and available resources. BOINC's current implementation follows a master/worker model, with no communication or coordination between clients, only supporting embarrassingly parallel applications.

This creates problems for existing projects when dealing with data distribution and storage. As volunteer computing projects gain in popularity and their user-bases expand, network requirements can easily become more demanding, forcing projects to upgrade both their computer hardware and network capacities to cope with increased demand. There are also examples of existing applications for projects such as MilkyWay@home [10] or ClimatePrediction.net [11] that could benefit from a distributed and scalable data management system, to share input or checkpoint files.

The exclusive support for embarrassingly parallel applications may, however, bring about a potentially more troubling issue: the absence of enough varied and enticing alternative use-cases for scientists to take advantage of volunteer computing, and for users to donate their resources.

The total number of BOINC users has been increasing, but there is also a constant stream of users leaving projects. Since they cancel each other out, the number of active users has stabilized, which means that the expansion of computing power is dependent on hardware upgrades in volunteer machines.

The introduction of GPU programming delivered the last boost in computational power, which allowed volunteer computing systems to surpass the 1 petaFLOPS barrier for the first time. While hardware optimizations will continue to drive further improvements and research, there is too much untapped potential to ignore the slump in new volunteers.

The user base is expected to increase if there are new projects to spark the interest of volunteers, and offer scientific goals that compel ordinary people to donate their resources. Unfortunately, the reputation that Volunteer Computing holds in the High Performance Computing (HPC) world does not attract potential developers. Most scientists outsource HPC decisions to IT personnel, who usually do not consider VC as an alternative since they believe it cannot handle meaningful or "serious" jobs [12].

By surpassing the master/worker model, and implementing new mechanisms and algorithms to handle more complex applications and paradigms, we can provide a two-fold solution: new projects may be executed on top of BOINC, potentially gathering more volunteers; Volunteer Computing will be able to demonstrate that it is useful for complex jobs, with significantly different requirements.

A. Cloud Computing and MapReduce

There has been a growing trend in the last decade to take advantage of Internet resources and the ease-of-access to remote computing sites. This use of web-based processing, referred to as Cloud Computing, constituted a paradigm shift from the typical client-server model to a more Service-Oriented Architecture (SOA). This paradigm shift is likely to continue in the near future, with services and applications being moved to the Internet.

Cloud Computing shares a few similarities with Volunteer Computing, specifically in terms of providing dynamically scalable over-the-Internet resources. Furthermore, commodity hardware develops faster than specialized hardware, while there have been significant improvements in network throughput on the last mile of the Internet. This means that some applications that were previously exclusive to cluster or data-centre environments may be able to run over the Internet (many are not be suited to a higher latency environment, if they have synchronization requirements, for example).

MapReduce is a software framework for parallel data-intensive computations recently popularized by Google [2], but it can be representative of a wide range of applications that can be reduced to map and reduce operations (present in Lisp and many other functional languages). It is useful in grids, clusters or Cloud Computing environments, due to the high volume of data transferred between nodes. There are several implementations available besides Google's original system, but the most popular one is Hadoop [3], developed by Apache.

MapReduce consists of a first "map" step, in which a master node (acting as a coordinator) divides the initial input into smaller chunks, each to be used in its own separate sub-task, and distributes them to worker nodes. Each worker processes the tasks it was given, and reports its completion to the master. At the next "reduce" step, the master node schedules one or more tasks on workers to perform join operations on the map outputs, to combine them in a way to get the output – the answer to the initial problem to be solved (Fig. 1).

In these environments, MapReduce relies on a shared, parallel file system at each step (such as Google File System [13]). For example, the map step reads its input files from the parallel file system. Map outputs, on the other hand, are

usually written on the local disk, since they are intermediate data files and are often discarded after the job is finished. In the reduce step workers must read these outputs from the map workers to perform the computation. Once the reduce tasks have been executed, their output is stored in the parallel file system, to be retrieved later by the user who submitted the job.

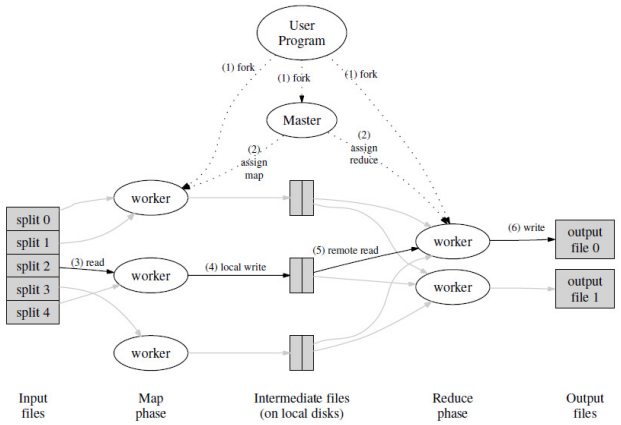

Fig. 1 – MapReduce job execution. Image taken from [2]

The advantage of MapReduce is the parallelism inherent to the map and reduction operations. Each mapping task is usually independent of the other, which makes it an embarrassingly parallel first step. A set of "reducers" can also perform the reduction phase – the only requirement is that all outputs of the previous map step within the same key range be presented to the same reducer.

Our objective is to adapt it to an unreliable, unsecure Internet environment, and obtain valid results with an application makespan that compensates the job distribution. MapReduce can be regarded as a good example of a complex framework, employed in several applications on diverse scientific fields, that can successfully be deployed as a volunteer computing project.

MapReduce can be considered as a gateway to allow other paradigms or more complex applications to be run on a VC system. There are several examples of MapReduce workflows, and one could consider other types of scientific workflows (e.g.: CancerGrid [14]) as candidates to run on desktop grids. On more data-intensive scenarios, it should be possible to deploy BOINC clients as distributed agents, such as web crawlers. Although numerous challenges would undoubtedly arise, the opportunity to leverage such a vast resource pool in volunteer PCs, Playstations or even digital TV receivers warrants further research in this area.

III.BOINC-MAPREDUCE

A.Overview

In order to create a BOINC prototype that would run MapReduce jobs, we had to take into account some of BOINC's characteristics, and design our system with them in mind. In our system, *BOINC-MR*, communication always starts from the client, never from the server, just as in the original system. This prevents connectivity issues arising from the use of firewalls or NAT by end-users, in client-server communication.

Since we introduced inter-client data transfers in our prototype, there are still difficulties in guaranteeing reliable communication between users. However, we decided not to tackle this problem at the moment, and instead focus on the evaluation of the system in different scenarios. For this version of the prototype, we used TCP sockets to transfer data between the map and reduce phase. This problem is further discussed in the limitations and future work sub-section, where we discuss a few potential solutions.

During the development of BOINC-MR, Hadoop was the system more thoroughly analyzed for several reasons: it is open source, which allowed us to have a better idea of how it works; it has its own implementation of a distributed file system, HDFS [15], which could be beneficial at a later stage in the prototype; and it is used by numerous important organizations such as Yahoo, eBay, or Facebook [16].

B.Map and Reduce Coordination and Execution

In order to run MapReduce jobs, changes had to be made both in the BOINC client and server. Since the map and reduce phases are significantly different, in terms of client-to-client coordination and data transfers, we handled each phase differently. In this sub-section we'll focus on the server changes and how both steps were coordinated, and we'll discuss client alterations in the ensuing sub-section.

When designing BOINC_MR, retro-compatibility was a major concern, since it would be counter-productive to require all clients to update their software to support MapReduce jobs. In our prototype, ordinary BOINC clients can be used to run these applications, although they obviously do not support inter-client data transfer (either serving or downloading files from other users).

As mentioned earlier, map tasks are completely independent of each other, with no dependencies between them or any shared data. Therefore, BOINC-MR follows the traditional protocol when scheduling work during the map phase (Fig. 2).

Users request work from the project's central server, while providing information on how many work units are currently being run and on hold. The scheduler takes into account the workload of each requester, as well as its hardware and availability information when choosing the work unit to send back.

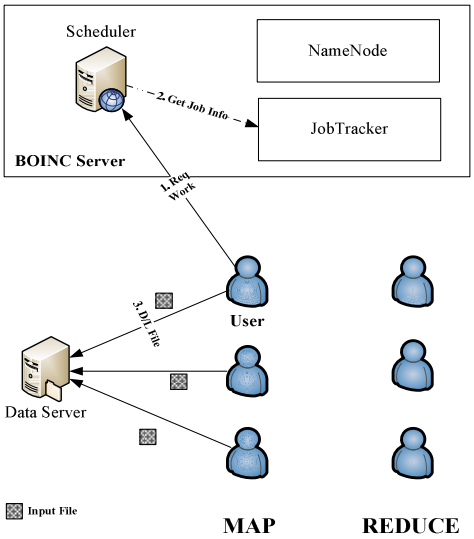

Fig. 2. BOINC-MR Map Phase. 1) Each user requests work from the server; 2) Scheduler checks for available map work units to send to client; 3) Users download input from project's data server, and execute the computation.

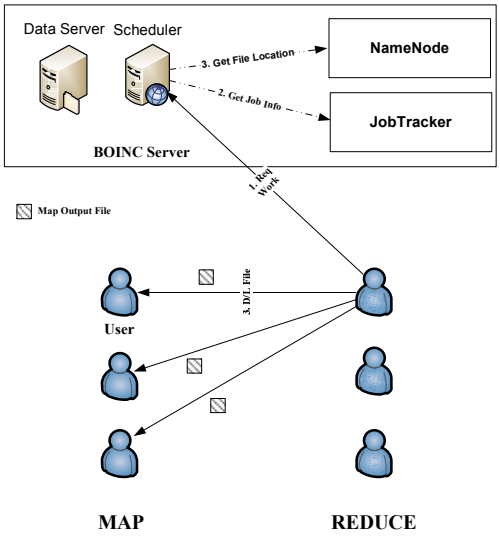

Fig. 3. BOINC-MR Reduce Phase. 1) Each user requests work from the server; 2) Scheduler checks for available reduce work units to send to client; 3) Scheduler uses JobTracker to identify which clients have finished map tasks for this job, and have their output data available for upload; 4) User downloads map output data from specified users, and executes the computation.

Map work units must be manually added to the central server, as any other applications, using specific scripts. Each work unit has two template files for input and output data. Additional information must be added to these templates, to identify the MapReduce job it belongs to (using a *<mapreduce>* tag). The transitioner and feeder daemons at the server create the results (work unit instances) and add them to the project's database. All map input data are saved on the project's data servers (BOINC has an option to specify the location of files, so they can be stored in any machine with a HTTP server). We created a general configuration file to the project's directory, *mr_jobtracker.xml*, which is used to specify MapReduce parameters, such as number of mappers and reducers.

Upon receiving a work request, the scheduler may choose to send map results, according to its normal matchmaking algorithm. *JobTracker*, a new module on the server, provides information on map or reduce tasks to be given to the client. Since all the input files are available on the project's data servers, the client downloads them as usual through HTTP. Once in possession of the executable and required inputs, the client executes the application and returns the results back to the central server – in the future, we intend to send hashes of the output data back to the server, instead of the files themselves to save bandwidth.

Since our prototype allows transfers between mappers and reducers, it is in our best interest to keep client-server communication to a bare minimum. To that end, map outputs should not be uploaded to the server; instead, each output's hash would be reported back, and validated using the usual quorum method.

Once all the map work units have been returned and the results have been validated, the system moves to the reduce phase (Fig. 3).

Reduce work units are automatically created by the BOINC-MR at this stage. Information on which users ran map tasks for each MapReduce job is saved on the central database, so the scheduler appends to each reduce result the address (IP and port) of mappers holding output for the same job.

Therefore, all reduce results sent to clients have the location of the required input data, as IP addresses of other users and, if map outputs were returned to the server, as the project's data server address. The latter allows non BOINC-MR clients to be able to execute MapReduce jobs without any changes required – this option is nowhere near optimal since all data must go through the server. After downloading all required input, the reduce task is executed, and the output is uploaded back to the server.

Data received by map users are trusted since all initial input is received from central server, as any other BOINC project. Map output data, on the other hand, require validation before being used as input by reduce tasks, since we have to consider byzantine behavior: malicious users or errors during the computation.

In this first version of the prototype, we took advantage of the existing BOINC mechanisms to validate jobs: replication. Therefore, each map work unit is sent to N different users (as results), and in order to be validated there must be a quorum of identical outputs – 2 out of the 3 users must return the same value, for example. This was also applied to reduce work units.

C.BOINC-MR Client and Word Count Application

Changes on the client were more comprehensive, since we had to introduce inter-client communication, and support the MapReduce framework.

For the prototype's first version, we did not think it would compensate to develop a "full-blown" MapReduce API and

add it to the existing BOINC API, used by developers to write their applications. Instead, when creating the word count example, we inserted MapReduce functionalities into the code.

The map word count task, for example, receives the number of reduce workers as input. Each map output's key (a word in our example) is hashed and the output file to it write to is decided based on the number of reduce tasks – modulo the number of reducers.

After moving all corresponding map outputs to each reduce worker location, the word count reduce application iterates through the values that are associated with each key (word) and outputs its sum. The final output from each reducer is uploaded back to the server, and can be merged into a single file, if necessary.

For inter-client communication, we used the TCP protocol, due to its simplicity and ease of testing. As mentioned previously, we did not address NAT and firewall traversal but the next sub-section describes some of the alternative solutions to be implemented and tested in the future.

We open a TCP for listening to incoming connections whenever a map task has finished and its output(s) is available. We dynamically adapt to the number of files being served, and stop accepting connections when there are no more files available for upload. This happens when the MapReduce job has finished (all reduce tasks have been returned and validated) or if the files have been served for too long – reduce application may have failed but server is still unaware. In case the server decides a reduce task should be restarted or scheduled on another client, the map outputs' timeout is reset (even if it has already been reached in the meantime), and the file becomes available for upload.

The timeout value must be chosen according to the expected execution time of a map task, and take into account possible client failures. Since we did not consider node failure in our tests, we used large enough value to allow all inter-client transfers to take place. Although failure tolerance was not our main goal for this first version, we have already implemented a fall back mechanism for failed inter-client downloads. After *n* failed attempts, the user resorts to downloading the file from the server. This requires map outputs to be always returned to the server, which is not an ideal solution, but guarantees that a job's execution will not be stopped due to transfer failures.

BOINC's HTTP transfers are handled by the curl library, which allows multiple simultaneous transfers to and from the server. In order to deal with concurrent file uploads and downloads to/from other clients, we had to introduce threading to BOINC. Threads were required due to the amount of data that had to be shared with ongoing transfers and to allow a quick termination of existing communications when needed (user needing the machine and BOINC exiting, for example). We kept a threshold for a maximum number of inter-client connections, so as to not overload the network. A further improvement would be to use TCP-Nice [17], which we mention in the following section.

D.Limitations and Future Work

With the introduction of inter-client transfers, we face some of the problems of Peer-to-Peer systems, especially in terms of communication. Network Address Translation (NAT) causes well-known difficulties in this respect, since peers may not be reachable from outside their personal network. Although we did not tackle this problem in the first version of the prototype, we present a few techniques to be integrated into the system as future work.

One of the most effective methods of NAT and firewall traversal is known as "hole punching" [18]. In this technique, two hosts, each in its own private own network, contact a public server with a reachable IP address that uncovers external and internal address information for them. Since the request was client-initiated, the server knows their IP addresses and port numbers for that session, which are then shared with both hosts, and used to establish communication. It's widely used in UDP-based applications and protocols [19], but it has also been shown to work on TCP [18] (less effectively). However, the technique is not applicable in all scenarios or with all types of NATs, as NAT operating characteristics are not standardized.

To overcome this limitation, one has to consider fail-over mechanisms after trying simpler solutions. In our case, if one client is behind a NAT, connection reversal could be used whereby the NATed node initiates communication regardless of which client requested the file. If both clients are behind NATs, we would then turn to the previously mentioned STUN-like NAT traversal [19] [18]. If all else failed, we would resort to a failover mechanism involving a publicly reachable relay node, as seen in the TURN protocol [20]. This tiered approach is used successfully by P2P systems such as Skype [21], and is quite robust to NAT and firewall connectivity issues.

In a volunteer computing environment the server could work as a relay node, but that would require all map output to be sent back to the project servers, thus minimizing the advantages of having inter-client communication. Another possibility would be to have a client fulfill that role, thus creating a supernode-based P2P network [22]. Such networks organize participants into two layers: supernodes and ordinary nodes. Supernodes are chosen from ordinary nodes (selection mechanism is usually based on connectivity and performance), and create an overlay network among themselves. Ordinary nodes must connect to a small number of supernodes and issue queries through them, in order to reach other peers. There are quite a few systems employing this type of network successfully such as Skype, KaZaA [23] or Gnutella [24], to bypass NAT and firewalls.

Even after solving the connectivity problem, it is still in our best interest to make good use of the available bandwidth. To that end, we intend to incorporate TCP-Nice [17], an algorithm that optimizes bandwidth consumption by proactively detecting congestion. It is a good fit for a volunteer computing environment because it has an end-to-end strategy optimized to support background transfers. The algorithm is currently being

used in a BitTorrent client (UDP version) [25].

IV. Experiments

To test the prototype, experiments of small scale were performed, and various parameters were considered when trying new scenarios. The results are presented in this section, as well as information on the testing infrastructure.

A. Experiment Setup

The area of this research requires many machines to achieve meaningful results without having to resort to a simulator. In this initial stage, however, we tried to get preliminary results on the performance of MapReduce jobs on top of BOINC and BOINC-MR. Therefore, we took advantage of UT Austin's Center for Information Assurance and Security (CIAS) Emulab network testbed [26]. The CIAS Emulab provides around 40 machines, each of which has 5 100Mbit interfaces through high-speed Cisco switches.

In our experiments we used 2 different node types: pc3001, with Dell PowerEdge 2850, 3GHz Intel Pentium IV Xeon, and 1GB RAM; and pcr200, with Dell PowerEdge r200, Quad Core Intel Xeon X3220, and 8GB RAM. Our BOINC-MR prototype was created on top of a BOINC client version 6.11.1, and server version 6.11.0. For the original BOINC clients, the latest version was used: 6.13.0.

To evaluate our scenarios, we developed a BOINC project to run the word count MapReduce application, and used the default parameters for the first tests. We tweaked some options to adapt to each scenario to obtain more realistic results. Each work unit is replicated into 2 results/instances, which means that when we mention 10 map work units or tasks, this means there are 20 map results to be executed. Each work unit is only validated if both results are identical.

The word count application is a typical example of a MapReduce job. The map function reads an input file word by word and outputs one line per word, with the format "*word 1*" – e.g.: "test 1". The reduce application reads one line at a time, and increments the count for each unique word. We set a fixed size of 1GB for the initial input file to be split into chunks (number of chunks is the same as the number of maps). Therefore, if we run a test with 20 map work units, each input file will have 50MB in size.

BOINC suggests using at most 15 input files, because of size limitations in the database. If more are needed, they should be compressed into single files, and sent to the client to uncompress before execution. We cannot do that since each reduce input file (created by map computation) is stored in a different user.

To compare different scenarios and versions we used application makespan, in order to be able to pinpoint if there is a slowdown in either the map or reduce stage. We ran all our tests with our BOINC-MR server, but used both the original BOINC client and our BOINC-MR client.

B. Map-Reduce Coordination

Table I has the results of our experiments. We varied the number of nodes, and tried different combinations of map and reduce work units. The first part of the table used normal BOINC clients, while BOINC-MR was used in the last row. We focused more on the environment variables than BOINC-MR itself, since it may suffer significant changes to deal with NAT traversal, for example. This way, we were able to obtain results on a system that could be used immediately, without any constraints.

TABLE I
WORD COUNT MAKESPAN

Nodes	# Map WUs	# Red WUs	Map Time	Reduce Time	Total Time
10	10	2	484	337	1121
10	20	2	376	349	1133
15	15	3	747 [*396*]	604 [*312*]	1529 [*1011*]
15	30	3	983 [*364*]	322	1378 [*758*]
20	20	5	383	455 [*341*]	1111 [*997*]
20	40	5	649 [*360*]	700 [*391*]	1681 [*1083*]
30	30	7	716 [*373*]	345	1373 [*1030*]
30	40	5	368	399	1174
BOINC-MR					
20	**20**	**5**	**612**	**318**	**1216**

Execution times for word count application, with initial 1GB input file, with variable number of map and reduce work units. Reduce and map phase execution is considered to start once the first task is assigned to a client. The end of a phase is signaled by the report or upload of the last output file. Total time is the interval between the scheduling of the first map task and the return of the last reduce output. All times are in seconds.

Our BOINC-MR client did not improve the overall performance, although the reduce step was the fastest (due to the inter-client transfers). The map step took too much of a share of the whole job to really see larger differences. In future iterations, we expect to experiment with a wider range of applications, to evaluate which scenarios are the most suited. As we stand, we can see it can provide the same level of performance, even with the added complexity on the client-side.

We have two results per cell: the first, to the left is the value obtained after running the experiment, and average of the time taken for each step (interval between receiving task from scheduler to reporting it as done); the second value, in italic, was derived after studying the results, and is also the average of time each step would take.

The main difference to the second result is that we discarded the results of the slowest node of the experiment. In other words, by examining the results obtained, it was not unusual for a single node to hold up the entire computation. This happens because we are running a single job, with few work units, which means that clients will usually be sent away by the scheduler when there is no more work to distribute. To avoid server congestion, BOINC uses exponential backoff, which means that for several minutes, a client does not attempt to contact the server, not even to report a finished computation.

A little more in-depth search revealed that it was not uncommon for a node (or even two when using 30 nodes) to back off at the exact moment before he had the result ready to report. This creates a delay sometimes larger than the backoff interval (600 seconds).

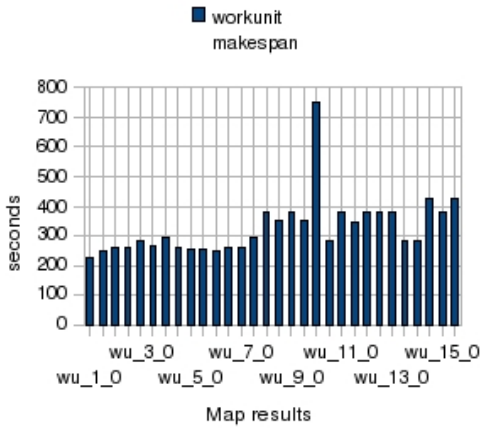

Fig. 4. Map application makespan for our experiment with 15 map work units (30 results). This is a good example of a recurring problem during our tests: one node did not report the completion of its tasks due to the backoff interval, and consequently delayed the beginning of the reduce step.

Fig. 4 is a perfect example from a scenario with 15 map workers. BOINC workers upload map outputs as soon as they are available. However, the task they belong to is only reported as completed in the next scheduler RPC. Although the server already has the files ready for validation, the task can only be processed after it has been reported. This delay is dependant on the frequency of client-server communication, which is specified by the user. Each user contacts the scheduler to request more work whenever it is below a threshold of remaining hours of work, or after a timeout (in case of error or no work being available from the server). In our case, since we only execute a single job, eventually there will be no more work units available, which causes the unwanted delay.

There is another delay on the passage between the map and reduce phase. Once the last remaining map task has been reported, the server has to validate it, create new reduce work units and insert them into the database. During this process, the server has no work, and clients are obligated to backoff exponentially from contacting the server. The backoff interval determines how soon a client will be able to request work again and obtain a reduce work unit.

C.Minimizing Impact of Slower Nodes

There are a few alternative solutions for this problem. First, this may be less noticeable when using a larger number of jobs at the same time. We wanted to evaluate the prototype in a simple environment, but having work constantly available at the scheduler should minimize the problem.

The fact that results are not immediately reported can be more of a nuisance in a MapReduce job for the map step. Delays in the first stage can have a major impact on the rest of the computation. Therefore, map work units should have priority in these cases and be reported as soon as their upload is completed, even if it meant increasing server congestion. Another possibility is to introduce intermediate data downloads. Instead of waiting for the computation to finish before downloading the output, clients should be able to start downloading as soon as files become available.

V.RELATED WORK

There are several research projects dealing with Cloud Computing, since it is a hot topic. We will limit the scope to work related to volunteer computing or with applicability to this system. Cloud@home [27] is a recent, interesting project that aims to combine concepts of Cloud Computing and Volunteer Computing, to overcome the weaknesses of the original computing paradigms. It is mainly focused in taking advantage of cloud resources by volunteer computing projects. One of its papers studied the cost and benefits of using clouds in either side of the "barricade": either substituting volunteers; or hosting projects [28]. Although it deals with the same research areas, its goals and application is transversal to our work. It would be possible to use cloud resources as an alternative to BOINC-MR, whenever harder deadlines were set or more resources were needed.

MOON (MapReduce On opportunistic eNvironments) [29] is the system that most closely resembles our prototype. MOON is an extension of Hadoop, with adaptive task and data scheduling mechanisms to account for node failure typical of an enterprise desktop grid. It was designed for cluster-like environment, such as a student lab, where nodes are controlled and trusted. According to the authors, it can deliver a three-fold performance improvement to Hadoop in a volatile environment. The target environment is completely unlike our own, which makes all the difference in terms of requirements and assumptions. Adapting Hadoop may work in a closed environment such as a cluster, but it would be of little help over the Internet, since there are too many obstacles when taking this path. We feel it is much easier to start on a system tailored for the Internet, and large-scale volunteer computing, such as BOINC, and then add MapReduce capabilities. Our prototype, although not optimized, is ready for deployment in any BOINC project, either by having users open ports to allow inter-client communication or by having data go through the central server. MOON, on the other hand, is restricted to clusters.

An interesting application of the MapReduce framework was presented in [30]. The authors state that using the reduce phase as a bloom filter enabled large scale. Results came back as 0 or 1, and the successful searches ("1") would then be re-run locally. This turned out to be faster than transferring the full result back to the master. In our case, we intend to send the hash instead of the full file, but for certain types of applications it would make sense to apply a Bloom filter.

VI. Conclusions

We have been witnessing rapid hardware and software evolution, which cannot be fully leveraged without having enough variety in the types of application that can be run on distributed systems. Furthermore, network bandwidth for end-users has increased considerably in the last few years, providing the environment for more complex application scenarios to become a reality.

MapReduce is a paradigm that has enjoyed tremendous success lately, and has been growing in popularity, with several different implementations and use cases. Its main advantage is that many applications can be broken down into sequences of MapReduce jobs (some with only map or just reduce sections). MapReduce applications are often used within Cloud Computing environments, since they are tailored for cluster execution and use large amounts of data stored in data centers. By adapting them to a volunteer environment we are taking advantage of the increasing computational power, while expanding the range of tasks that can be tackled by volunteers.

We created a BOINC prototype that can run MapReduce jobs (BOINC-MR), using a pull-model in which communication is always initiated by the client, instead of the traditional push-model used in clusters (by Apache's Hadoop for example). It uses TCP sockets to establish connections between clients, thus significantly reducing the network overhead on the central BOINC server.

We obtained preliminary results, running a simple MapReduce application, word count, as a proof of concept to demonstrate how a typical MapReduce application can be run on top of a modified volunteer computing middleware. We expect to run experiments on a more realistic setting such as Planetlab in the near future to more accurately assess the performance of our prototype.

There are many avenues of research opened for us, but we must first tackle the existing problems such as connectivity (NAT traversal) before we can harness the full computational power of a potentially very rewarding research area.

References

[1] F. Costa, I. Kelley, L. Silva, and G. Fedak, "Optimizing Data Distribution in Desktop Grid Platforms", *Parallel Processing Letters*, vol. 18, # 3, September 2008.

[2] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," in *Proceedings of OSDI '04: 6th Symposium on Operating System Design and Implementation*, San Francisco, 2004.

[3] Apache Hadoop.
See website at: http://hadoop.apache.org/

[4] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: Distributed data-parallel programs from sequential building blocks", in *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007 (EuroSys '07)*, March 2007.

[5] Folding@home Project.
See website at: http://folding.stanford.edu/

[6] Top 500.
See website at: http://www.top500.org/

[7] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer, "SETI@home: An experiment in public-resource computing", *Communications of the ACM*, Vol. 45 No. 11, pp. 56-61, Nov. 2002.

[8] David Anderson, "BOINC: A System for Public-Resource Computing and Storage", in *Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*, Pittsburgh, USA, November 2004.

[9] F. Cappello, S. Djilali, G. Fedak, T. Herault. F. Magniette, V. Neri, and O. Lodygensky, "Computing on large-scale distributed systems: XtremWeb architecture, programming models, security, tests and convergence with Grid", *FGCS Future Generation Computer Science*, 2004.

[10] MilkyWay@home.
See website at: http://milkyway.cs.rpi.edu/milkyway/

[11] ClimatePrediction.net
See website at: http://climateprediction.net/

[12] David Anderson, "BOINC: The Year in Review", presented at the 6th Annual Pangalactic BOINC Workshop, London 2010.

[13] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung, "The Google file system", in *19th Symposium on Operating Systems Principles*, pp. 29–43, Lake George, New York, 2003.

[14] J. Brenton, C. Caldas, J. Davies, S. Harris, and P. Maccallum, "CancerGrid: developing open standards for clinical cancer informatics", in *Proceedings of the UK e-science All Hands Meeting*, pp. 678–681, 2005.

[15] D. Borthakur, "The Hadoop Distributed File System: Architecture and Design".
http://hadoop.apache.org/hdfs/docs/r0.21.0/hdfs_design.html

[16] List of institutions using Hadoop:
http://wiki.apache.org/hadoop/PoweredB

[17] A. Venkataramani, R. Kokku, and M. Dahlin, "TCP Nice: A mechanism for background transfers", in *Proc. of OSDI*, Boston, MA, Dec. 2002.

[18] B. Ford, P. Srisuresh, and D. Kegel, "Peer-to-peer communication across network address translators", in *Proceedings of the 2005 USENIX Annual Technical Conference*, Anaheim, CA, Apr. 2005.

[19] J. Rosenberg, J. Weinberger, C. Huitema, and R. Mahy, "STUN: Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs)", RFC 3489, IETF, Mar. 2003.

[20] J. Rosenberg, R. Mahy, and C. Huitema, "Traversal Using Relay NAT (TURN)", IETF Internet Draft, February 2005.

[21] S. Guha, N. Daswani, and R. Jain, "An experimental study of the Skype peer-to-peer VoIP system," in *IPTPS*, 2006.

[22] M. Castro, M. Costa, and A. Rowstron, "A. Debunking some myths about structured and unstructured overlays", in *Proceedings of the NSDI '05*, Boston, MA, May 2005.

[23] J. Liang, R. Kumar, K. W. Ross, "The KaZaA Overlay: A Measurement Study", *Computer Networks Journal*, Oct. 2005.

[24] M. Ripeanu, I. Foster, and A. Iamnitchi, "Mapping the gnutella network: Properties of large-scale peer-to-peer systems and implications for system design," *IEEE Internet Computing Journal*, vol. 6, no. 1, 2002.

[25] µtorrent Micro Transport Protocol (µTP).
See website at: http://www.utorrent.com/documentation/utp

[26] University of Texas' Emulab.
See web site at: https://boss.cias.utexas.edu

[27] Vincenzo D. Cunsolo, Salvatore Distefano, Antonio Puliafito and Marco Scarpa, "Cloud@Home: Bridging the Gap between Volunteer and Cloud Computing", in *ICIC'09 Proceedings of the 5th international conference on Emerging intelligent computing technology and applications*, 2009.

[28] D. Kondo, B. Javadi, P. Malecot, F. Cappello, and D. Anderson, "Cost-benefit analysis of Cloud Computing versus desktop grids", in *Proceedings of the 2009 IEEE international Symposium on Parallel & Distributed Processing*, pp. 1-12, May 2009.

[29] Heshan Lin, Xiaosong Ma, Jeremy Archuleta, Wu-chun Feng, Mark Gardner, Zhe Zhang, "MOON: MapReduce On Opportunistic eNvironments", in *ACM International Symposium on High Performance Distributed Computing (HPDC '10)*, June 2010.

[30] P. Balaji, W. Feng, and H. Lin, "Semantics-Based Distributed I/O with the ParaMEDIC Framework", in *Proceedings of the IEEE/ACM International Conference on High Performance Distributed Computing (HPDC)*, Boston, MA, June 2008.