

A P2P Job Assignment Protocol for Volunteer Computing Systems

Daniela Barbalace

barbalace@si.deis.unical.it

Università della Calabria, Rende (CS), Italy

Pasquale Cozza

pcozza@deis.unical.it

Università della Calabria, Rende (CS), Italy

Carlo Mastroianni

mastroianni@icar.cnr.it

ICAR-CNR, Rende (CS), Italy

Domenico Talia

talia@deis.unical.it

Università della Calabria, Rende (CS), Italy



CoreGRID Technical Report

Number TR-0117

December 13, 2007

Institute on Knowledge and Data Management
Institute on Architectural Issues: Scalability, Dependability,
Adaptability

CoreGRID - Network of Excellence

URL: <http://www.coregrid.net>

A P2P Job Assignment Protocol for Volunteer Computing Systems

Daniela Barbalace

barbalace@si.deis.unical.it
Università della Calabria, Rende (CS), Italy

Pasquale Cozza

pcozza@deis.unical.it
Università della Calabria, Rende (CS), Italy

Carlo Mastroianni

mastroianni@icar.cnr.it
ICAR-CNR, Rende (CS), Italy

Domenico Talia

talia@deis.unical.it
Università della Calabria, Rende (CS), Italy

CoreGRID TR-0117

December 13, 2007

Abstract

Complex applications often require the execution of a large number of jobs in a distributed environment. One highly successful and low cost mechanism for acquiring the necessary compute power is the “public resource computing” paradigm, which exploits the computational power of private computers. However, applications that are based on this paradigm currently rely upon centralized job assignment mechanisms that can hinder the achievement of performance requirements in terms of overall execution time, load balancing, fault-tolerance, reliability of execution results, scalability and so on. This paper extends a super-peer protocol, proposed earlier by this group, for the execution of jobs based upon the volunteer requests of workers. The paper introduces a distributed algorithm that aims to achieve a more efficient and fair distribution of jobs to workers. This is obtained by the definition of different roles that can be assumed by super-peers and ordinary nodes on the basis of their characteristics. A simulation study is carried out to analyze the performance of the super-peer protocol and demonstrate the advantage of distributing the job assignment process.

1 Introduction

The term “public resource computing” [1] is used for applications in which jobs are executed by privately-owned and often donated computers that use their idle CPU time to support a given (normally scientific) computing project. The pioneer project in this realm is SETI@HOME [3], which has attracted millions of participants wishing to contribute to the digital processing of radio telescope data in the search for extra-terrestrial intelligence. A number of similar

This research work is carried out under the FP6 Network of Excellence CoreGRID funded by the European Commission (Contract IST-2002-004265).

projects are supported today by the BOINC (Berkeley Open Infrastructure for Network Computing [2]) software infrastructure. The range of scientific objectives amongst these projects is very different, ranging from Climate@HOME's [5], which focuses on long-term climate prediction, to Einstein@HOME's [9], aiming at the detection of certain types of gravitational waves.

This paper enhances a P2P-based distributed model, firstly proposed by this group in [8], that supports applications requiring the distributed execution of a large number of jobs with similar properties to current public-resource computing systems like BOINC. In such systems, a "super-peer" node can act as a centralized resource for a limited number of regular nodes, in a fashion similar to a current Grid system. At the same time, super-peers can make interconnections with other super-peers to form a P2P overlay network at a higher level, thereby enabling distributed computing on much larger scales.

Unlike BOINC, the model presented here does not rely on any centralized mechanisms for job and data distribution, but exploits decentralized techniques which are enabled by the super-peer paradigm. The jobs to execute are assigned to workers by means of "job adverts" which are produced by a job manager. A job advert is an XML document that describes the properties of a job to execute.

In the enhanced version discussed here, a distributed approach is used not only for job execution and data caching but also for job assignment. Job adverts are disseminated to a number of "job assigners" that are available on the network, and then assigned by these to worker nodes. Assignment is made in two phases: (i) first the job manager searches the network to discover job assigners and distribute job adverts among them, then (ii) workers, which are available for job executions, issue query messages to find job assigners and retrieve job adverts.

The super-peers play two fundamental roles: they route messages in a peer-to-peer fashion and also act as rendezvous to *match* queries issued by job assigners and workers with compatible job adverts.

The objective of this work is to evaluate and point out the benefits that derive from this decentralized approach for job assignment. This approach is profitably combined with a decentralized data caching scheme, already described in [7], through which workers retrieve input data, needed for the execution of jobs, from "data centers", i.e., from nodes specialized for the storage and maintenance of such data.

Simulation analysis performed with an event-driven simulator shows that the simultaneous use of these decentralized mechanisms for job assignment and data download can actually improve performance of public computing. To this aim, a set of performance indices have been analyzed, specifically regarding the overall time needed to execute the jobs, the average utilization of data centers, the network load and the load balancing among workers.

The remainder of the paper is organized as follows. Section 2 discusses related work in the field and shows how the implemented architecture presented here goes beyond currently supported models. Section 3 presents the super-peer model and the related protocol. Performance is analyzed in Section 4, and conclusions and future work are discussed in Section 5.

2 Related Work

Volunteer computing systems have become extremely popular as a means to garnish many resources for a low cost in terms of both hardware and manpower. The most popular volunteer computing platform currently available, the BOINC infrastructure [2] is composed of a scheduling server and a number of clients installed on users' machines. The client software periodically contacts the scheduling server to report its hardware and availability, and then receives a given set of instructions for downloading and executing a job. After a client completes the given task, it then uploads resulting output files to the scheduling server and requests more work. The BOINC middleware is especially well suited for CPU-intensive applications but is somewhat inappropriate for data-intensive tasks due to its centralized approaches for job assignment and data distribution.

The job assignment/scheduling problem consists in the assignment of a set of n jobs to a set of m machines such that some overall measure of efficiency is maximized, e.g., the time required to complete all tasks. This is a NP-hard problem [4], which is generally solved with centralized or hierarchical approaches [11]. Recently, distributed algorithms have been proposed for adaptation to P2P and Grid environments, for example, in [14] and [15]. However, to the best of our knowledge, distributed algorithms have never been applied to public resource computing applications.

The P2P paradigm has proven to be effective also for distributed data caching. Popular super-peer based networks among these system are the Napster [13] and Kazaa projects [12]. Recently, BitTorrent [6] has become the most widely used and accepted protocol for P2P data distribution, relying on a centralized tracking mechanism to monitor and coordinate file sharing.

However, P2P protocols might not be appropriate to scientific volunteer computing platforms due to their “tit for tat” requirement that necessitates a ratio between upload and download bandwidth, thus requiring peers to share data if they are recipients of it on the network. Further, it is difficult to establish trust for nodes that act as job assigners or data providers in the network; that is, it is difficult to stop these nodes acting as rogue providers and serve false data across the network or disrupt the network in some way.

The approach proposed in [8] and [7], and enhanced in this paper, attempts to combine the strengths of both a volunteer distributed computing approach like BOINC with decentralized, yet secure and customizable, P2P data sharing practices. It differs from the centralized BOINC architecture, in that it seeks to integrate P2P networking directly into the system, as job descriptions and input data is provided to a P2P network instead of directly to the client.

The BOINC middleware is especially well suited for CPU-intensive applications but is somewhat inappropriate for data-intensive tasks due to its centralized nature that currently requires all data to be served by a set group of centrally maintained servers.

The approach proposed in [8], and enhanced in this paper, attempts to combine the strengths of both a volunteer distributed computing approach like BOINC with decentralized, yet secure and customizable, P2P data sharing practices. It differs from the centralized BOINC architecture, in that it seeks to integrate P2P networking directly into the system, as job descriptions and input data is provided to a P2P network instead of directly to the client.

3 A Super-Peer Protocol for Job Submission

A data-intensive Grid application can require the distributed execution of a large number of jobs with the goal to analyze a set of data files. One representative application scenario defined for the GridOneD project [10] shows how one might conduct a massively distributed search for gravitational waveforms produced by orbiting neutron stars. In this scenario, a data file of about 7.2 MB of data is produced every 15 minutes and it must be compared with a large number of templates (between 5,000 and 10,000) by performing fast correlation. It is estimated that such computations take approximately 500 seconds. Data can be analyzed in parallel by a number of Grid nodes to speed up computation and keep the pace with data production. A single job consists of the comparison of the input data file with a number of templates, and in general it must be executed multiple times in order to assure a given statistical accuracy or minimize the effect of malicious executions.

Currently, this kind of application is usually managed through a centralized framework, in which one server assigns jobs to workers, sends them input data, and then collects results; however this approach clearly limits scalability. Conversely, we propose a decentralized protocol that exploits the presence of super-peer overlays, which are more and more widely adopted to deploy interconnections among nodes of distributed systems and specifically of Grids.

In the super-peer overlay, the simple nodes, or *workers*, are responsible for the execution of jobs, whereas the *super-peers* constitute the backbone of the super-peer overlay. A worker or super-peer node can play different *roles*, as detailed in the following:

- *job manager*: a node that plays this role produces *job adverts*, i.e., files that describe the characteristics of the jobs that must be executed, and distributes these adverts to job assigners, which in turn assigns jobs directly to workers. The job manager is also responsible for the collection of output results.
- *job assigner*: it receives a number of job adverts from the job manager and is responsible for the assignment of the corresponding jobs to workers.
- *data source*: it receives data from an external sensor, and provides this data as input for the execution of jobs. Each data file is associated to a *data advert*, i.e. a metadata document which describes the characteristics of this file.
- *data cacher*: it has the ability to cache data (and associated data adverts) retrieved from a data source or another data cacher, and can directly provide such data to workers.

In the following, *data sources* and *data cachers* are collectively referred to as *data centers*, since both are able to provide data to workers, although at different phases of the process: data sources from the beginning, data cachers after retrieving data from data sources or other data cachers. When a worker is available for a job execution, it first issues a *job query* to obtain a job advert and then a *data query* to retrieve input data. A worker can disconnect at any time;

if this occurs during the downloading of a data file or the execution of a job, that one will not be completed. Super-peers play the role of both *routing* and *rendezvous* nodes, since they compare job and data description documents (*job* and *data adverts*) with *queries* issued to discover these documents, thereby acting as a meeting place for job or data providers and consumers.

3.1 Job Assignment and Data Download

The objective of the job assignment protocol is the distribution of jobs among workers. To decentralize the assignment of jobs, the protocol exploits the presence of multiple *job assigners* on the network. After producing a number of job adverts, the job manager distributes them to the job assigners, which then assign them to workers.

Figure 1 depicts the sequence of messages exchanged among the job manager, the job assigners and the workers. A sample topology is shown, with 6 super-peers, among which 2 assume also the role of job assigner.

After producing a number of job adverts, which describe the job to execute, the job manager *JM* issues an “assigner query” to search for available job assigners on the network. This query is replicated and forwarded by super-peers (step 1). As job assigners, in this case the nodes *JA*₁ and *JA*₂, receive an assigner query, they respond to the job manager by directly sending it an “assigner advert” (step 2). The job manager collects assigner adverts for a given interval of time, than it distributes the job adverts among the discovered job assigners (step 3).

Subsequently, Figure 1 describes the behavior of the protocol when a job query is issued by the workers *W*_A and *W*_B (step 4). A job query is expressed by an XML document and typically contains hardware and software features of the requesting node as well as CPU time and memory amount that the node offers. A job query matches a job advert when it is compatible with the information contained in the job advert, e.g., the parameters of the job and the characteristics of the platforms on which it must be executed. A job query is forwarded through the super-peer network until it is delivered to a job assigner. If the job assigner has not assigned all the matching job adverts received by the job manager, one of these job adverts is sent to the worker (step 5) that will execute the corresponding job.

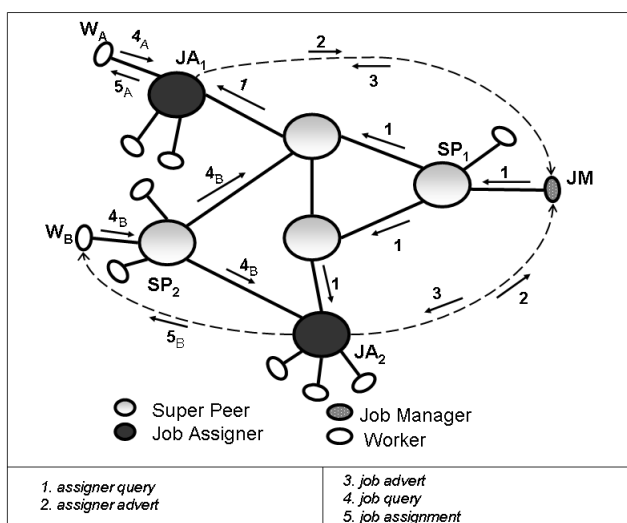


Figure 1: Super-peer job assignment protocol: sample network topology and sequence of exchanged messages to distribute job adverts among job assigners (messages 1, 2, 3) and assign them to workers (messages 4, 5).

The job advert also contains information about the data file which is required for the job execution and must be retrieved by the worker. Download of input data is performed in the *data-download* phase, which is not described in Figure 1 and is better detailed in [7]. In a similar fashion to the job assignment phase, the worker sends a *data query* message, which travels the super-peer network searching for a matching input data file stored by a data center. Since the same file can be maintained by different data centers, a data center that successfully matches a data query does not send data directly to the worker, in order to avoid multiple transmissions of the same file. Conversely, the data center sends only a small *data advert* to the worker. In general, a worker can receive many data adverts from different data

centers. Then it chooses a data center, according to policies that can rely on the distance of data centers, their available bandwidth etc. After making the choice, the worker initiates the download operation from the selected data center. When the super-peer connected to the worker acts also as a data cacher, data is first retrieved from the data cacher and then forwarded to the worker. This enables the dynamic caching functionality, which allows for the replication of data files on multiple data cachers and leads to well known advantages such as increased degree of *data availability* and improved *fault tolerance*. Dynamic caching also allows for a significant improvement of performances, as shown in Section 4.

Upon receiving the input data, the worker executes the job, reports the results to the job manager and possibly issues another *job query*, so restarting the protocol. Each job must be executed a specified number of times, as mentioned in Section 3. As a job manager receives the result of a job execution, it checks if the required number of executions has been reached for that job. In this case, the job manager informs the job assigners, that will no longer assign this specific job to workers.

4 Performance Evaluation

A simulation analysis was performed by means of event-based simulation, in order to evaluate the performance of the super-peer protocol described in the previous section. The simulation scenario, and the related network and protocol parameters, are set to assess the representative astronomy application mentioned in Section 3.

The number of workers is set to 1000 and it is assumed that an average of 10 workers are connected to a super-peer. Each super-peer is connected to at most 4 neighbor super-peers. Workers can disconnect and reconnect to the network at any time: average connection and disconnection time intervals are set, respectively, to 4 hours and 1 hour. A data download or job execution fails upon the disconnection of the corresponding worker. The number of jobs N_{job} varies from 50 to 500. In our application scenario, each job corresponds to the analysis of a portion of the gravitational waveforms received from the detector. The parameter N_{exec} is defined as the minimum number of executions that must be performed for each job, either to enhance statistical accuracy or minimize the effect of malicious executions. To achieve this objective, redundant job assignment is exploited: each job advert can be matched and assigned to workers up to a number of times equal to the parameter MTL, or *Matches To Live*, whose value must be not lower than N_{exec} . A job is assigned to workers until either the MTL parameter is decremented to 0 or the job manager receives the results for at least N_{exec} executions of this job. A proper choice of MTL can compensate for possible disconnections of workers and consequent job failures.

It is assumed that local connections (i.e., between a super-peer and a local simple node) have a larger bandwidth and a shorter latency than remote connections. Specifically, the bandwidth values of local and remote connections are set to 10 Mbps and 1 Mbps, respectively, whereas transmission delays are set to 10 ms and 100 ms. To compute the download time with a proper accuracy, each data file of 7.2 MB is split in 1 MB segments, and for each segment the download time is calculated assuming that the downstream bandwidth available at a data center is equally shared among all the download connections that are simultaneously active from this data center to different workers.

A TTL parameter is used to limit the traffic load: this corresponds to the maximum number of hops that can be performed by a job query issued by a worker. In our test, this parameter is set to 3.

Simulations have been performed to analyze the overall execution time, i.e. the time needed to execute all the jobs at least N_{exec} times. The overall execution time, T_{exec} , is crucial to determine the rate at which data files can be retrieved from the detector and sent to the network, so as to guarantee that the workers are able to keep the pace with data production. We also evaluated the balancing of jobs among workers, the traffic load and the average utilization of data centers.

4.1 Performance of Distributed Job Assignment

The first set of simulations was performed to verify the advantage of having multiple job assigners in a network. The number of data sources is set to 2, whereas the overall number of data centers is 50, which is half the number of super-peers. The values of T_{exec} and MTL are set, respectively, to 10 and 20. This value of MTL is sufficient to compensate for possible disconnections of workers, while larger values would be ineffective. This kind of analysis is discussed in [7].

Figure 2 shows the overall execution time vs. the number of job assigners, with different values of N_{job} . A significant reduction of T_{exec} is perceived as the number of job assigners increases. This can be explained as follows.

If only one or few job assigners are available, most jobs will be assigned to the workers which are close to job assigners, because their job queries will succeed with a higher probability and in a shorter time. Therefore a small percentage of workers will issue a high percentage of data queries, which will be likely served by the data centers close to such workers. Two main drawbacks derive from this scenario: (i) many jobs will be served by a few workers, and many download operations will be served by a small set of data centers. A larger overall execution time is the obvious consequence, whereas a wider availability of job assigners can limit both the mentioned drawbacks. Furthermore, the reduction of T_{exec} is more evident when the computational load, i.e., the number of jobs, is higher. It is noted, however, that as the number of job assigners increases, the execution time first decreases then tends to get stable: therefore the optimum number of job assigners can be set depending on the minimum incremental improvement (i.e., the improvement obtained by adding one more data center) that is considered acceptable.

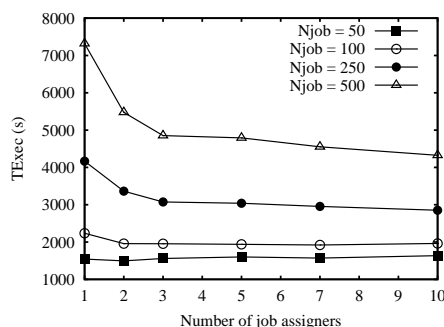


Figure 2: Overall execution time vs. the number of job managers, for different numbers of jobs.

Figure 3 reports the overall number of query messages that are forwarded on the network to complete the required number of job executions. As the number of job assigners increases, fewer hops are necessary to discover a job assigner, therefore the traffic load correspondingly decreases.

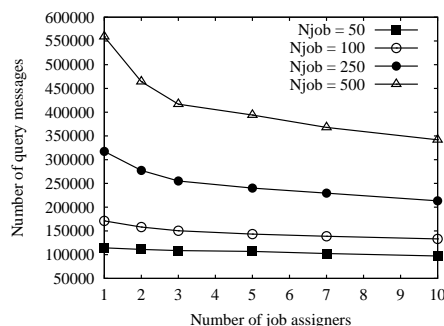


Figure 3: Network load vs. the number of job managers, for different numbers of jobs.

Figure 4 confirms that the work is better and more uniformly distributed with more Job Assigners. Indeed the maximum number of jobs assigned to a single worker remarkably decreases as the number of job assigners increases. This results not only in a shorter execution time, but also in a better load balancing among workers, which is another obvious objective of public scientific applications.

In conclusions, the protocol for distributed job assignment brings several significant benefits in terms of execution time, network load, load balancing. Of course the availability of a large number of job assigners requires more administrative workload on the super-peer nodes that are given this role. Moreover, strict security requirements must be guaranteed by job assigners, in order to prevent possible malicious actions that might hinder the correct progression of the scientific application.

4.2 Performance of Distributed Data Caching

A second set of experiments was performed to evaluate the effectiveness of the distributed caching approach, enabled by the availability of multiple data centers, when combined with distributed job assignment. The examined network is

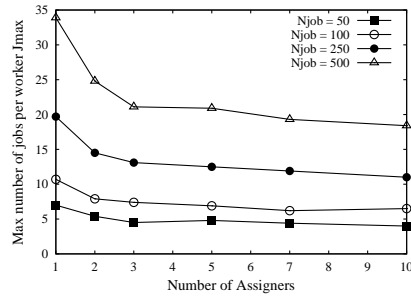


Figure 4: Maximum number of job per worker vs. the number of Job Managers, for different numbers of jobs.

analogous to that examined in Section 4.1, except that the number of jobs to execute is fixed to 500 and the number of available data centers is varied from 2 (which is the number of data sources) to 50, i.e., half the number of super-peers. Moreover, results are reported for a number of job assigners ranging from 1 to 5, since this is the interval for which a significant impact on performance indices can be perceived, as shown in Section 4.1.

Figure 5 shows the values of the overall execution time calculated for this scenario. The time decreases as more data centers are made available in the network, for two main reasons: (i) data centers are less heavily loaded and therefore data download time decreases, (ii) workers can exploit a higher parallelism both in the downloading phase and during the execution of jobs.

Figure 5 does not report results for some combinations of the number of data centers and the number of job assigners, because the disconnections of workers do not allow for the completion of all the required job executions. Specifically, if the number of job assigner is 2 or larger, the number of data centers should be at least 15, while, if only one job assigner is available, at least 10 data centers are needed. In fact, if only a few data centers are available, each of these is likely to be overloaded by a large number of workers' requests; as a consequence, the download time increases and the disconnection of a worker during the download phase becomes a more probable event.

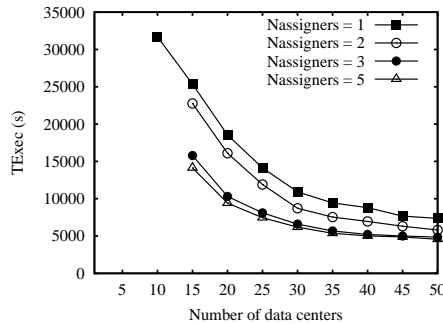


Figure 5: Overall execution time vs. the number of data centers, for different numbers of job assigners.

Figure 6 reports the maximum number of jobs executed by a single worker, and proves that the a wider availability of data centers improves load balancing among workers, as the maximum number of executed jobs decreases. In fact, with few data centers, the workers which are closer to them tend to execute more jobs, because the download phase is faster. However, if more data centers are installed, the differences among workers are attenuated, in particular when the overall computation load is high.

Figure 7 shows the average utilization of data centers for the same scenario. This is defined as the fraction of time that a data center is actually utilized, i.e., the fraction of time in which at least one download connection, from a worker or a data cacher, is active with this data center. The value of this index is averaged on all the data centers and is an important efficiency index that helps to evaluate the convenience of adding more data centers to the network.

The average utilization decreases as the number of data centers increases but, in contrast with the execution time, curves do not get to a relatively stable value. This is a useful indication for setting a proper number of data centers. To illustrate this, consider the results obtained with 3 job assigners. While the overall execution time can be decreased until the number of data centers is increased to about 40, the average utilization continues to decrease as more data centers are made available. As an example, if the number of data centers were increased from 40 to 50, there would be

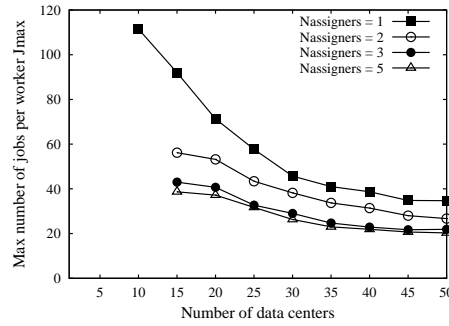


Figure 6: Maximum number of jobs per worker vs. the number of data centers, for different numbers of job assigners.

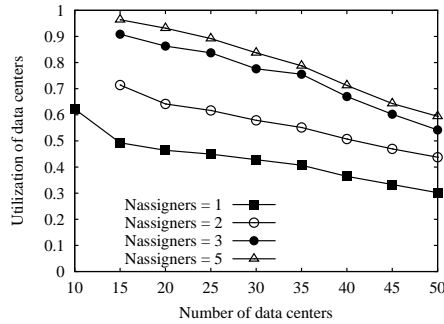


Figure 7: Average utilization of data centers vs. their number, for different numbers of job assigners.

a worse exploitation of data centers but no significant reduction in the execution time, from which it can be concluded that an appropriate number of data centers is indeed 40.

5 Conclusions

In this paper we have reported on the ongoing work and results of a super-peer architecture, and related protocols, for the execution of scientific applications according to the “public resource computing” paradigm, where resources are distributed and generally donated by network volunteers, and a large number of independent jobs are executed in parallel by dispersed worker nodes.

Use of super-peer overlays allowed us to define distributed protocols both for the assignment of jobs and for the retrieval of input data. Specifically, these two phases exploit the availability of super-peer nodes that, in addition to play the role of routing and rendezvous nodes, are also configured to act as job assigners and data centers, respectively.

Simulation results showed that the combined application of distributed job assignment and distributed data download leads to considerable performance improvements, in terms of the amount of time required to execute the jobs, the traffic load, the balancing of load among workers and the utilization of data centers.

By using a system described in this paper, BOINC-like applications are able to distribute the job assignment procedure, which is today mostly centralized, and replicate their current static data server functionality through a dynamic and decentralized data distribution system. These enhancements would enable projects to automatically scale their computational and data needs without additional administrative overhead as their user-base or problem size increases.

6 Acknowledgments

We would like to thank Ian Taylor and his colleagues at the Cardiff University for their help in defining the distributed architecture presented here.

References

- [1] David P. Anderson. Public computing: Reconnecting people to science. In *Proceedings of Conference on Shared Knowledge and the Web*, pages 17–19, Madrid, Spain, November 2003.
- [2] David P. Anderson. Boinc: A system for public-resource computing and storage. In *GRID '04: Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing (GRID'04)*, pages 4–10, 2004.
- [3] David P. Anderson, Jeff Cobb, Eric Korpela, Matt Lebofsky, and Dan Werthimer. Seti@home: an experiment in public-resource computing. *Communications of the ACM*, 45(11), 2002.
- [4] Peter Brucker. *Scheduling Algorithms*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2001.
- [5] Climateprediction.net. See web site at <http://climateprediction.net/>.
- [6] B. Cohen. Incentives Build Robustness in BitTorrent. In *Proceedings of the First Workshop on Economics of Peer-to-Peer Systems*, June 2003.
- [7] Pasquale Cozza, Ian Kelley, Carlo Mastroianni, Domenico Talia, and Ian Taylor. Cache-enabled super-peer overlays for multiple job submission on grids. In *Proceedings of the CoreGRID Workshop on Grid Middleware*, Dresden, Germany, June 2007.
- [8] Pasquale Cozza, Carlo Mastroianni, Domenico Talia, and Ian Taylor. A super-peer protocol for multiple job submission on a grid. In *Euro-Par 2006 Workshops*, volume 4375 of *Springer-Verlag LNCS*, pages 116–125, Dresden, Germany, 2007.
- [9] Einstein@home. See web site at <http://einstein.phys.uwm.edu/>.
- [10] GridOneD. See web site at <http://www.gridoned.org/>.
- [11] Klaus Krauter, Rajkumar Buyya, and Muthucumar Maheswaran. A taxonomy and survey of grid resource management systems for distributed computing. *Software-Practice and Experience*, 32(2).
- [12] Jian Liang, Rakesh Kumar, and Keith W. Ross. The KaZaA Overlay: A Measurement Study. In *Proceedings of the 19th IEEE Annual Computer Communications Workshop*, Bonita Springs, Florida, USA, 2004.
- [13] Napster. See web site at <http://www.napster.com/>.
- [14] Vijay Subramani, Rajkumar Kettimuthu, Srividya Srinivasan, and P. Sadayappan. Distributed job scheduling on computational grids using multiple simultaneous requests. In *Proceedings of the 11th IEEE Symposium on High Performance Distributed Computing*, Edinburgh, UK, 2002.
- [15] Norihiro Umeda, Hidemoto Nakada, and Satoshi Matsuoka. Peer-to-peer scheduling system with scalable information sharing protocol. In *Proceedings of the 2007 International Symposium on Applications and the Internet Workshops SAINT-W '07*, 2007.