Use of P2P Overlays for Distributed Data Caching in Public Scientific Computing

Pasquale Cozza, Domenico Talia
`{cozza|talia}@si.deis.unical.it`
Università della Calabria, Rende (CS), Italy

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

Ian Kelley, Ian Taylor
`{I.R.Kelley|Ian.J.Taylor}@cs.cardiff.ac.uk`
Computer School, Cardiff University, United Kingdom

Use of P2P Overlays for Distributed Data Caching in Public Scientific Computing

Pasquale Cozza, Domenico Talia
{cozza|talia}@si.deis.unical.it
Università della Calabria, Rende (CS), Italy

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

Ian Kelley, Ian Taylor
{I.R.Kelley|Ian.J.Taylor}@cs.cardiff.ac.uk
Computer School, Cardiff University, United Kingdom

Abstract

Many types of distributed scientific and commercial applications require the submission of a large number of independent jobs. 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. Recently decentralized peer-to-peer and super-peer technologies have been proposed for adaptation in these systems. A super-peer protocol, proposed earlier by this group, is used for the execution of jobs based upon the volunteer requests of workers, and a super-peer overlay is used to perform two kinds of matching operations: the assignment of jobs to workers and providing workers the input data needed for job execution. This paper extends this super-peer protocol to account for a more dynamic and general scenario, in which: (i) workers can leave the network at any time; (ii) each job is executed multiple times, either to obtain better statistical accuracy or to perform parameter sweep analysis; and, (iii) input data is replicated and distributed to multiple data caches on-the-fly, in an effort to improve performance in terms of data availability, fault tolerance and execution time. A simulation study was performed to analyze the latest iteration of the super-peer protocol and specifically evaluate the new features.

1 Introduction

Distributed computing has in recent years become the next technological evolution in the high-performance and consumer computing fields. Grid computing and Peer-to-Peer (P2P) networking are two sets of such technologies that have partly addressed issues in that area and even though they have evolved from different communities, it has started to become desirable in the academic and industrial arenas to explore possible areas of convergence [12]. Super-peer systems have been proposed [9] [13] to achieve a balance between the inherent efficiency of centralized networks, and the autonomy, load balancing and fault-tolerant features offered by P2P networks. In such systems, a "super-peer" node can act as a centralized resource for a limited number of regular "peer" 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.

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 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 [4], which focuses on long-term climate prediction, to Einstein@HOME's [5], aiming at the detection of certain types of gravitational waves.

This paper reports on a super-peer 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. Unlike BOINC, the data distribution scheme outlined here does not heavily rely on any centralized mechanisms for job and data distribution. To adapt to a P2P environment, the super-peer job submission protocol requires that job execution is preceded by two *matching* phases, the first for job assignment and the second for downloading of input data from *data centers*, which are super-peers having data storage facilities.

In the work here, we extend and enhance the data distribution scheme defined in [8] and refine its analysis to account for a more dynamic and general scenario, in which: (i) workers can leave the network at any time; (ii) each job is executed multiple times, either to obtain better statistical accuracy or to perform parameter sweep analysis; and, (iii) input data is replicated and distributed to multiple data centers on-the-fly, in an effort to improve protocol performance in terms of data availability, fault tolerance and execution time. To demonstrate these concepts, a set of simulation runs have been performed to evaluate the impact of the replication and caching mechanisms on performance indices, specifically regarding the overall time needed to execute the chosen jobs and the average utilization of data centers.

The remainder of the paper is organized as follows. Section 2 discusses related work in the field and shows how the proposed architecture 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

Desktop Grids, in the form of 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 nature that currently requires all data to be served by a set group of centrally maintained servers. BOINC allows a project to configure a fixed and static set of data servers that are maintained for a particular project and made available for data distribution. Although this scheme enables a number of servers to help load balance the network and scales well for the current applications utilizing BOINC, the topology is static and has a number of problems scaling if more data-intensive applications are introduced. For example, under the current system, an administrator must dedicate time to configure and maintain these data serving machines, which are generally independent for each BOINC project. Such machines are costly to purchase and maintain, additionally they are centrally administered; therefore cannot generally be used by other BOINC projects. The real cost, however, generally lies with the expenditure required to maintain the needed network bandwidth to support a project, especially given the extremely large scale of some public resource computing projects.

Peer-to-Peer (P2P) data sharing networks have proven to be effective in distributing both small and large files across public computing platforms in a relatively efficient manner that utilizes both participants' upload and download bandwidth. Popular super-peer based networks among these system are the Napster [10] and Kazaa projects [11]. Recently, BitTorrent [7] 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. Although this approach has proved quite scalable and efficient, it might not be appropriate to scientific volunteer computing platforms due to its "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. Such stringent requirements are likely to prove problematic for volunteer computing platforms. For example, there are security implications of opening additional ports for traffic since every client in the network becomes a server. Further, it is difficult to establish trust for data providers in the network; that is, it is difficult to stop people acting as rogue providers and serve false data across the network or disrupt the network in some way.

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, employing a job manager that sends data to a P2P network instead of directly to the client. Once data enters the P2P network, it is automatically propagated across the data nodes as required through simple caching schemes. Such a system helps to distribute data load dynamically in a decentralized fashion, both in topology and administratively, making it far more suitable to the Grid domain than static centralized systems. For example, inherent in BOINC-style networks is the requirement to send a needed data file to several workers multiple times to provide reliability and fault-tolerance. This replication imposes an extra and unneeded expenditure of server bandwidth, which can be avoided through a P2P caching mechanism that replicates the data across the network when it is first transferred. By replicating the data in such a way, there is an immediate decrease on the required central server bandwidth and also more advanced data distribution mechanisms can be supported, such as placing the data in locations where it is most needed on the network. Further, a number of projects require many nodes to process the same data, with different parameters, a situation that can also exploit the overlay described here. A gravitational-wave scenario presented in this paper is an example of such an algorithm.

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 [6] 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.

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.

The super-peer protocol relies on the definitions of different *roles* that can be assumed by Grid nodes (i.e., by super-peers or by simple nodes), as detailed in the following:

- the *data sources* are nodes that receive data from an external sensor, for example a gravitational wave detector in the GridOneD scenario, and provide this data to nodes 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.

- a *job manager* produces *job adverts*, i.e., files that describe the characteristics of the jobs that must be executed, and is also responsible for the collection of output results.

- the *workers* are Grid nodes that are available for job execution. A worker first issues a job query to obtain a job to be executed and then a data query to retrieve the input data file. A worker can disconnect at any time; if this occurs during the execution of a job, that one will not be completed.

- the *super-peers* constitute the backbone of the super-peer overlay. Super-peers are connected to workers through a centralized topology and to each other through a high level P2P network. In the proposed protocol, super-peers play the role of *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.

- *data cachers* are super-peers which have the additional ability to cache data (and associated data adverts) retrieved from a data source, and can directly provide such data to workers. Super peers and data cachers can be distributed on separate nodes if desired but in this experiment we hosted the data cachers on super peers for simplicity.

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 very beginning, data cachers after retrieving data from data sources or other data cachers. Notice that the distinction between data sources and data cachers has been introduced in this work, since here evaluation focuses on dynamic caching mechanisms. Conversely, in [8] it was assumed that all data centers possess the data files before starting the job submission process: in other words the replication and caching phase was separated from the job submission phase.

We assumed that only super-peers can assume the role of data centers, but the protocol can be easily extended to the case in which even simple peers can cache and provide data. We envisage that the same user-driven process is used to configure a peer; that is, each user can decide if a node will be a super peer and/or data center, as well as a worker. In the BOINC scenario, the existing dedicated machines would form the obvious data-center backbone and other peers (with high storage and network capacity) would also make themselves available in this mode.

3.1 Job Assignment and Data Download

Figure 1 depicts the sequence of messages exchanged among workers, super-peers and data centers for the execution of the job submission protocol in a sample topology with 5 super-peers, of which one is a data source and two others are data cachers. This example describes the behavior of the protocol when a job query is issued by the worker W_A. In this case dynamic caching is not exploited because: (i) input data is only available on the data source DS_0, i.e., no data cachers have yet downloaded data; (ii) data cannot be stored by the super-peer connected to W_A, since this is not a data cacher. The behavior of the protocol with dynamic caching is explained later.

The protocol requires that job execution is preceded by two matching phases: the *job-assignment* phase and the *data-download* phase. In the *job-assignment* phase the *job manager*(the node JM in the figure) generates a number of *job adverts*, which are XML documents describing the properties of the jobs to be executed (job parameters, characteristics of the platforms on which they must be executed, information about required input data files, etc.), and sends them to the local rendezvous super-peer, which stores the adverts. This corresponds to step 1 in the figure. Each worker, when ready to offer a fraction of its CPU time (in this case, worker W_A), sends a *job query* that travels the Grid through the super-peer interconnections (step 2), until the message time-to-live parameter is decremented to 0 or the job query finds a matching job advert. 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 the job query parameters are compatible with the information contained in the job advert. Whenever the job query gets to a rendezvous super-peer that maintains a matching job advert, such a rendezvous assigns the related job to the requesting worker by directly sending it a *job assignment* message (step 3).

In the *data-download* phase, the worker that has been assigned a job inspects the job advert, which contains information about the job and the required input data file, e.g., size and type of data. In a similar fashion to the job assignment phase, the worker sends a *data query* message (step 4), 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 (in this example the data source DS_0) sends only a small *data advert* to the worker (step 5). The worker chooses a data center, and initiates the download operation (steps 6 and 7). After receiving the input data, the worker executes the job, reports the results to the job manager (step 8) and possibly issues another *job query*.

It can be noticed that in the job assignment phase the protocol works in a way similar to the BOINC software, except that job queries are not sent directly to the job manager, as in BOINC, but travel the super-peer network hop by hop. Conversely, the data download phase differs from BOINC in that it exploits the presence of multiple data centers in order to replicate input data files across the network.

3.2 Dynamic Caching

Dynamic caching allows for the replication of input data files on multiple data cachers. This leads to well known advantages such as increased degree of *data availability* and improved *fault tolerance*. Moreover, dynamic caching
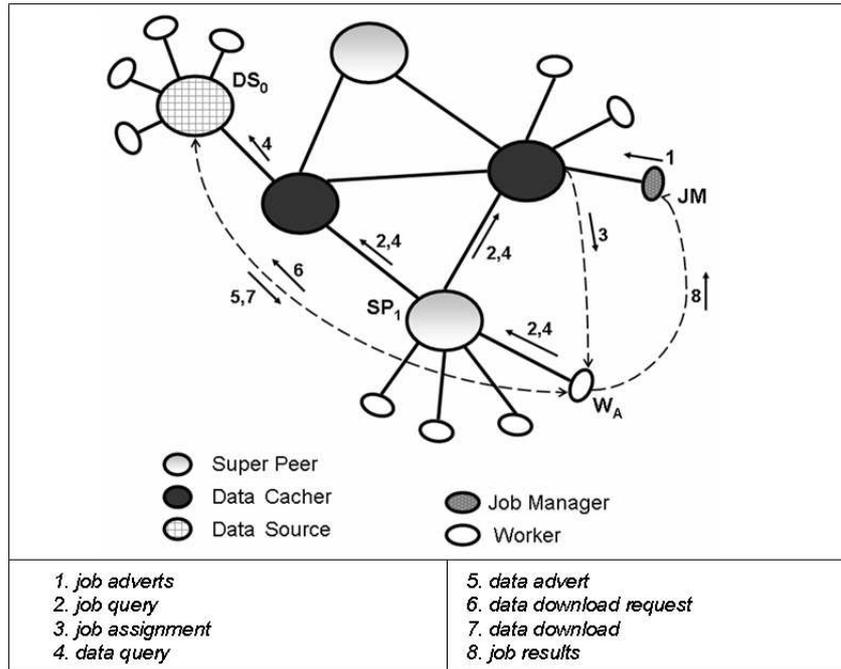
Figure 1: Super-peer job submission protocol: sample network topology and sequence of exchanged messages to execute one job at the worker W_A. Dynamic caching is not used because it is assumed that data cachers have no yet stored data.

allows for a significant saving of time in the data download phase, because data queries have a greater chance to find an available data center, and most workers can download data from a neighbor data cacher instead of a remote data source. The remaining part of this section illustrates the dynamic caching mechanism, while the performance evaluation is discussed in Section 4.

Figure 2 shows how the protocol handles dynamic caching, both in the *replication* phase (which occurs when data is downloaded from a data source and stored by a data cacher) and in the *retrieval* phase (which occurs when data is retrieved from a data cacher by a worker). These two mechanisms are described in Figure 2 by displaying the messages exchanged when two workers W_B and W_C, connected to the same data cacher DC_2, issue two job queries at different times, first W_B then W_C. For simplicity, only messages related to the *download phase* are shown, and they are distinguished by subscripts B and C, corresponding to the two workers. The data query issued by W_B finds a matching in the data source DS_0. As opposed to the case described in Figure 1, this time the super-peer connected to W_B is a data cacher, DC_2. To let this data cacher store the data file, the data advert is not sent directly to the worker W_B, but first to DC_2 and then from DC_2 to the worker. Analogously, the data file is downloaded by DC_2, which replicates and caches it, and then passes it to the worker. Subsequently, DC_2 will act like a data source for the period of time in which it maintains the data file in its cache. In this example, the data query issued by W_C will be served directly by the cacher DC_2 instead of the data source DS_0.

To increase performance, a file splitting approach is adopted: data files are not downloaded as a whole but split in ordered fragments (of 1 Mbytes size in our case). For example, if the data cacher DC_2 when receiving a data query does not hold the entire data file but has already received a part of it from DS_0, it will not forward the data query, because it will soon receive the remaining fragments from DS_0. As soon as it receives these fragments, DC_2 will pass them to the requesting worker.

A further improvement could be obtained by enabling the parallel download of data segments from two or more data centers. The benefits and drawbacks of this enhancement are currently under investigation.
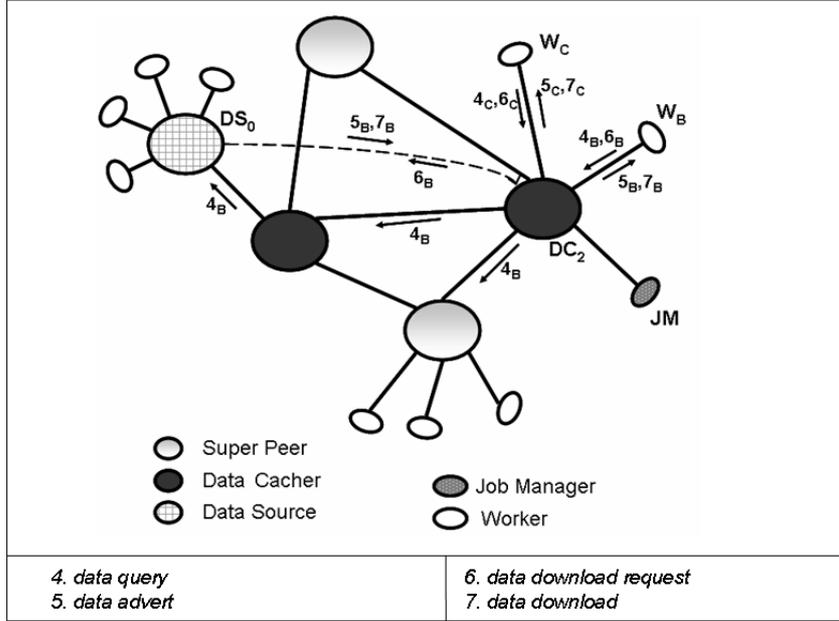
Figure 2: Download phase of the super-peer job submission protocol, with dynamic caching. After the request of worker W_B, the data cacher DC_2 retrieves the data file from the data source DS_0, replicates and caches the file, and delivers it to W_B. Subsequently, the request of worker W_C is directly server by the data cacher DC_2.

4 Performance Evaluation

A simulation analysis was performed by means of an ad hoc event-based simulator, written in C++, to evaluate the performance of the cache-enabled super-peer protocol.

The simulation scenario is described in Table 1. The parameters of the representative astronomy scenario mentioned in Section 3 are used for the test case (file size, job execution time, etc.). It is assumed that all the jobs have similar characteristics and can be executed by any worker.

Workers can disconnect and reconnect to the network at any time. This implies that a job execution fails upon the disconnection of the corresponding worker. This is a new feature with respect to the basic protocol presented in [8]. Table 1 specifies the assumed average connection and disconnection times of workers.

To achieve multiple execution of every single job (which can be useful to enhance statistical accuracy or perform parameter sweep analysis) two parameters have been added: N_{exec} and MTL. Specifically, each job must be executed at least a given number of times, N_{exec}, which is set to 10. To guarantee this, a strategy based on 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*, which must be not lower than N_{exec}. 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. To compute the download time with a proper accuracy, a data file 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 simultaneous active from this data center to different workers.

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 data source in order to guarantee that the workers are able to keep the pace with data. We also computed the average utilization index of data centers, U, which 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 U is averaged on all the data centers and can be seen as an efficiency index.

Table 1: Simulation scenario.

Scenario feature	Value
Number of super-peers, N_{speer}	25 to 100
Maximum number of neighbors for a super-peer	4
Average number of workers connected to a super-peer	10
Average connection time of workers	4 h
Average disconnection time of workers	1 h
Number of data centers (data sources + data cachers)	about 50% of N_{speer}
Size of input data files	7.2 Mbytes
Latency between two adjacent super-peers	
(*or between two remote peers in a direct connection*)	100 ms
Latency between a super-peer and a local worker	10 ms
Bandwidth between two adjacent super-peers	
(*or between two remote peers in a direct connection*)	1 Mbps
Bandwidth between a super-peer and a local worker	10 Mbps
Mean job execution time	500 s
Number of jobs, N_{job}	50 to 500
Number of executions requested for each job, N_{exec}	10
Matches to live, MTL	10 to 30

4.1 Redundant Submission of Jobs

A first set of simulation was performed for a network with 25 super-peers, 1 data source and 12 data cachers. The purpose was to investigate the possible benefits of the redundant submission of jobs, in other words the impact of the Matches To Live (MTL) parameter on performance indices. Values of MTL were set to values ranging from 10 to 30, while N_{exec} was fixed to 10.

Figure 3 shows that the overall execution time increases with the number of jobs N_{job} and, more interestingly, tends to decrease as the value of MTL increases. The reason of the latter phenomenon is that a larger MTL allows to better compensate for the possible failure of jobs due to peer disconnections. However, this effect is not more evident when the MTL exceeds a threshold, in fact the execution time becomes approximately constant. Actually very large values of MTL are not even exploited since the job assignment phase is terminated as soon as the Job Manager receives, for each job, the results related to N_{exec} executions. Finally, notice that results related to MTL=10, and MTL=12 for N_{job} equal to 500, are not reported because with no or low degree of redundancy it was proved not possible to complete all the required jobs.
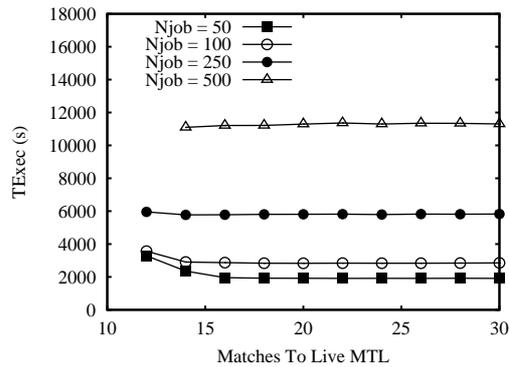

Figure 3: Overall execution time vs. the value of MTL, for different numbers of jobs.

Figure 4 shows that the average utilization of data centers, and hence the efficiency of the protocol, increases with the amount of computation assigned to workers, i.e., both with the number of jobs and with the MTL value. To understand this, it must be considered that data cachers are not very utilized in the first phase of the process, because they have not yet retrieved data from data sources, whereas they are fully exploited only after they have retrieved such data. Therefore, the utilization of data centers is high only when the number of required job executions is large

enough to make the caching of data convenient. On the other hand, when the amount of computation is low, the time interval required by data cachers to retrieve data files is relevant with respect to the overall execution time, therefore data cachers are not exploited for a large fraction of time, which explains the low values of the utilization index.
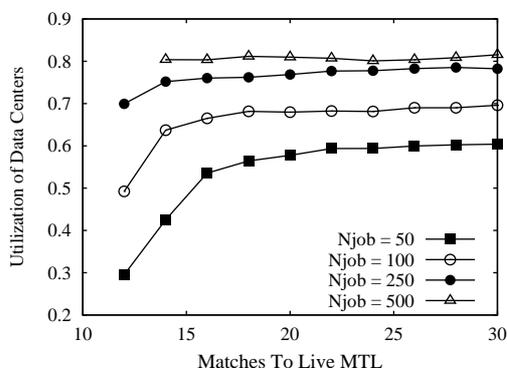

Figure 4: Average utilization of data centers vs. the value of MTL, for different numbers of jobs.

4.2 Scalability Analysis

An additional set of simulations were performed to evaluate the behavior of the protocol in variable-sized networks. The number of super-peers was set to 25, 50, and 100, which corresponds to about 250, 500 and 1000 workers. The number of data centers was set to about half the number of super-peers, specifically to 13, 25 and 50, respectively. Two different scenarios were tested: when only one data source is available, regardless of the network size; and when the number of data sources is proportional to the network size. In this second case, we increase the number of data sources linearly with the increased network size by doubling the data sources at each stage i.e. 1, 2 and 4, respectively. This essentially compares how our approach affects a BOINC network if the BOINC administrator provides more data servers or data sources into the network. The number of jobs N_{job} was set to 250, for both scenarios.

Results are reported in Figures 5 and 6. It is interesting to note that the overall execution time may be decreased by using a larger number of workers. However, in the analyzed scenarios, this improvement is only noticed when the number of super-peers is increased from 25 to 50, while the use of larger networks is not beneficial. Furthermore, it can be noticed that the reduction of the execution time is obtained only if the MTL is larger than a threshold. Indeed, if MTL is low, it is likely that a considerable percentage of jobs are assigned to workers that are distant from the data source(s); the larger is the network, the longer are download times, and therefore the overall execution time. Conversely, with a large MTL, it is more probable that the at least N_{exec} jobs are assigned to workers directly connected to data centers, which assures lower download times; in this case, a larger number of workers actually decreases the overall execution time, because more jobs can be executed in parallel.
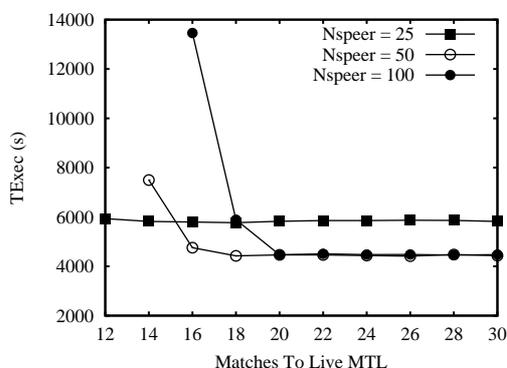

Figure 5: Overall execution time vs. the MTL value, for different network sizes: 25, 50 and 100 super-peers. Results are obtained with 1 data source.
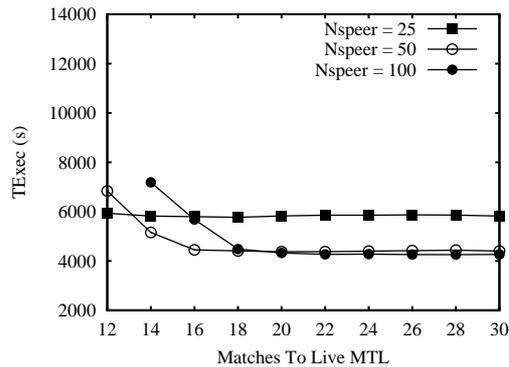
Figure 6: Overall execution time vs. the MTL value, for different network sizes: 25, 50 and 100 super-peers. Results are obtained with a number of data sources proportional to the network size, i.e., equal to 1, 2 and 4.

Moreover, the comparison of Figures 5 and 6 shows that the execution time decreases if the number of data sources is increased from 1 to a number proportional to the network size, but again this only occurs with low values of MTL and for large networks (num. of workers $>= 500$). With large values of MTL, the execution time hardly depends on the number of data sources: one data source suffices to propagate data files to data cachers and to workers.

5   Conclusions

In this paper we have reported on the ongoing work and results of our research into a decentralized architecture for data-intensive scientific computing. This research has been undertaken according to the "public-resource computing" paradigm, where resources are distributed and generally donated by network volunteers. To take full advantage of the full spectrum of client-side capabilities in these types of networks, where participants generally have not only idle CPU cycles, but also substantial network bandwidth, we have presented a super-peer data distribution scheme that attempts to leverage the available resource capabilities for the submission of a very large number of jobs. In the scenario presented here, a small group of nodes maintains and advertises job description files and a large number of dispersed worker nodes execute the required tasks. Job assignment is performed by this group of rendezvous peers, which form a super-peer overlay network and match job descriptions with job queries when they are issued by available worker nodes.

To provide support for this scheme, a number of simulations have been performed to evaluate the impact of application (the number of jobs and the number of times that each of them is assigned to workers for statistical analysis) and network parameters (the number of workers and data centers) on performance indices such as the overall time to execute a given set of jobs and the utilization of data centers. Results for the test-case we identified show that the availability of several data centers and the use of dynamic caching bring benefits to applications. During this process, we have also observed that there is a balance between a larger number of data servers and the effective utilization of a single data center. Given the network and data parameters, the optimal number of data centers for a given problem space can be identified, thereby helping maximize the return of investment related to the deployment of new data centers. By using a system described in this paper, BOINC-like applications are able to replicate their current static data server functionality through a dynamic and decentralized data distribution system that enables projects to automatically scale their data needs without additional administrative overhead as their user-base or problem size increases.

Future work in this area will investigate a number of interesting research avenues, such as: (i) the evaluation of the pros and cons of parallel downloading of data segments from two or more data centers; and, (ii) the performance evaluation of using a super-peer schema for scenarios where input data is progressively being fed into the network from an external source as a data stream.

References

[1] David P. Anderson. Public computing: Reconnecting people to science. In *Proceedings of Conference on Shared Knowledge and the Web*, pages 17–19, 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, Washington, DC, USA, 2004. IEEE Computer Society.

[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] BOINC. Climate@home, http://climateprediction.net.

[5] BOINC. Einstein@home, http://einstein.phys.uwm.edu.

[6] Jodrell Bank Observatory Cardiff University, Manchester University. Gridoned, http://www.gridoned.org.

[7] B. Cohen. Incentives build robustness in bittorrent. In *Proceedings of the First Workshop on Economics of Peer-to-Peer Systems*, June 2003.

[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 *LNCS*, pages 116–125, Dresden, Germany, 2007. Springer-Verlag.

[9] Carlo Mastroianni, Domenico Talia, and Oreste Verta. A super-peer model for resource discovery services in large-scale grids. *Future Generation Computer Systems*, 21(8):1235–1248, 2005.

[10] Free Napster. Napster, http://www.napster.com.

[11] Sharman Networks. Kazaa, http://www.kazaa.com.

[12] Domenico Talia and Paolo Trunfio. Toward a synergy between p2p and grids. *IEEE Internet Computing*, 7(4):96–95, 2003.

[13] Beverly Yang and Hector Garcia-Molina. Designing a super-peer network. In *19th International Conference on Data Engineering ICDE*, 2003.