A Scalable Super-Peer Approach for Public Scientific Computation

Carlo Mastroianni [a], Pasquale Cozza [b], Domenico Talia [b], Ian Kelley [c], Ian Taylor [c]

[a] *ICAR-CNR*
87036 Rende (CS), Italy
mastroianni@icar.cnr.it
phone +39-0984-831725
fax +39-0984-839054

[b] *DEIS University of Calabria*
87036 Rende (CS), Italy
{*talia,pcozza*}*@deis.unical.it*
phone +39-0984-494726
fax +39-0984-494713

[c] *School of Computer Science, Cardiff University*
CF10 3XQ Cardiff, UK
{*I.R.Kelley,Ian.J.Taylor*}*@cs.cardiff.ac.uk*
phone +29-208-70109

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. We designed a super-peer protocol for the execution of jobs based upon the volunteer requests of workers, and a super-peer overlay for performing two kinds of matching operations: the assignment of jobs to workers and the download of input data needed for job execution. This paper analyzes a 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. A simulation study was performed to analyze the super-peer protocol and specifically evaluate performance in terms of execution time, utilization of data centers, load balancing, and ability to efficiently scale with the number of jobs and the network size.

Keywords

data caching, Grid computing, job execution, peer-to-peer, public resource computing, super-peer networks

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 [31].

The super-peer model has been recently proposed [25][34] to build the information system of Grids. This model is naturally appropriate for Grids, as a large-scale Grid can be viewed as a network composed of small-scale, proprietary systems, also referred to as Grid Organizations. Within each Organization, the nodes that have the largest capabilities act as super-peers, while the other nodes use super-peers to access the Grid. Super-peers are interconnected with each other, thus forming a P2P network at a higher layer.

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 [4], 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 software infrastructure that evolved out of that project, the Berkeley Open Infrastructure for Network Computing, or BOINC [2][3][7]. The range of scientific objectives amongst these projects is very different, ranging from Climateprediction.net [11] focus on long-term climate prediction to Einstein@HOME's [17] aim of detecting gravitational waves. The potential impact of these public resource computing networks is quite large, for example, in September 2007, the Folding@HOME project [19], which simulates protein folding, received the Guinness World Records as the most powerful distributed computing network in the world. This is a major accomplishment, with its distributed network reaching the petaflop mark for compute power, and representing 200,000 PCs and over 670,000 PS3 gaming consoles [5].

This paper extensively describes the extended version of a super-peer based distributed model, originally proposed by this research group in [15], 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 rely heavily 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 [15][14] 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, execution time and load balancing among workers. 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, the average utilization of data centers, the network load and the average and maximum load of workers.

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, and performance evaluation is given in Section 4. Section 5 describes three application areas that could benefit from the decentralized and flexible data sharing techniques presented in this paper. Conclusions and future work are discussed in Section 6.

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. Two of the popular volunteer computing platforms available today are BOINC and XtremWeb.

BOINC [2] is by far the most popular volunteer computing platform available today, and to date, over 5 million participants have joined various BOINC projects. Although the projects that utilize BOINC are diverse in their scientific nature, each one has something in common with the others: they have work units that can be easily distributed to run autonomously in a highly distributed and volatile environment. To achieve this task, each project must not only prepare its data and executable code to work with the BOINC libraries and client/server infrastructure, but they must also setup and maintain their own individual servers and databases to manage the projects data distribution and result aggregation. The core BOINC infrastructure is composed of a scheduling server and a number of clients installed on users' machines. To participate in the network and perform calculations, the client software periodically contacts a centralized scheduling server to receive 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 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 can be costly to purchase and maintain, additionally they are centrally administered and 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.

XtremWeb [8][18] is another Desktop Grid project that, like BOINC, works well with "embarrassingly parallel" applications that can be broken into many independent and autonomous tasks. XtremWeb follows a centralized architecture and uses a three-tier design consisting of a Worker, a Coordinator, and a Client. In contrast to BOINC projects, which are generally tied to a specific application executable and therefore a particular scientific problem, the XtremWeb software allows multiple Clients to submit task requests to the system. When these requests are dispensed to Workers for execution, the Workers will retrieve both the necessarily data and executable to perform the analysis. The role of the third tier, called the Coordinator, is to decouple Clients from Workers and to coordinate tasks execution on Workers. To ease the deployment phase regarding the connection issues raised by firewall and NAT configuration, Clients and Workers initiate all communications toward the Coordinator node. Like BOINC, XtremWeb does not utilize a data caching network or sophisticated P2P algorithm for data propagation.

There are many other desktop grid computing platforms besides BOINC and XtremWeb that operate on the same basic principles of utilizing volunteers' donated computer power. Others worth mentioning are Aneka [10] and Entropia [9]. Aneka, the successor to Alchemi [24], is a modular and hierarchical web-services based desktop grid system, built using Microsoft's .NET framework, that has the advantage of letting users decouple the message passing and the application management logic to more easily facilitate a modular and customizable system. Additionally, Aneka strives to provide users with the ability to support message passing (MPI), which has generally not been pursued by other desktop grid architectures due to the large latencies involved and volatile nature of participant nodes. Entropia is another desktop grid environment, however, it differs from the previously mentioned architectures in that it provides a virtual machine in which the different client applications can run. This creates a sandbox that can support different programming languages and codes, and is achieved by modifying job binaries and to link to Entropia's custom DLL libraries, thereby overriding certain calls made to the host environment.

Peer-to-Peer 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. P2P information systems are usually classified into *unstructured* and *structured*. In *unstructured* systems, resources are published by peers with no global planning, so that the network continuously grows and changes as peers join and leave the system. These systems generally feature the power-law and scale-free properties analyzed by Albert and Barabasi [6], which incorporate the preferential attachment behavior (the more connected a node is, the more likely it is to receive new links) that was proved to exist widely in real networks. Conversely, in *structured* systems, resources are assigned to hosts with a well-specified strategy, for example through a Distributed Hash Table (DHT) [30], by which the correct location of a resource on the network is obtained as the result of a hash function. The structured approach generally speeds up discovery operations but also presents important drawbacks [29], such as: (i) limited fault tolerance and scalability, since the disconnection of a peer requires an immediate reorganization of resources and of peer index tables; (ii) poor load balancing, because peers that are assigned the most popular resources can be easily overloaded.

Overall, structured approaches are usually efficient in file sharing P2P networks, but structure management can be cumbersome and poorly scalable in large and dynamic Grids. Moreover, structured algorithms may hinder the discovery of resources when they are not specified with a name or a code (for example the name of a file) but are defined with looser constraints, for example when the CPU speed of a computer or the response time of a Web service are required to be within a given interval. The latter case is much more frequent in Grids than in file sharing P2P networks. In particular, in public computing applications the matching of job queries and job adverts, as described in Section 3, is performed through the comparison of several hardware and software features, which is the reason why unstructured algorithms are preferred to structured ones in our study.

Super-peer systems have been proposed 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 nodes, in a fashion similar to a current Grid system, whereas the super-peer overlay network enables distributed computing on much larger scales. Popular super-peer based networks are the Napster [27] and Kazaa projects [22]. In [35] performance of super-peer networks is evaluated, and rules of thumb are given for an efficient design of such networks: the objective is to enhance the performance of search operations and at the same time to limit bandwidth and processing load. In [26] a general mechanism for the construction and the maintenance of a super-peer network is proposed and evaluated. In this work, a gossip paradigm is used to exchange information among peers and dynamically decide how many and which peers can efficiently act as super-peers.

Recently, BitTorrent [12] has become the most widely used and accepted protocol for P2P data distribution, gaining commercial support to distribute media content such as movies, MP3s, and TV shows. Unlike the super-peer architectures mentioned above, BitTorrent relies on a flat network and in its default implementation uses a centralized tracking mechanism to monitor and coordinate file sharing. In [21] authors show that the BitTorrent protocol features the following properties: reliability of file transfers even in the context of high volatility and node churn; scalability even when nodes show low to medium bandwidth, e.g., as in the Internet; and, ability to distribute large files even if the node originally serving the files has low communication capabilities. The possibilities of using BitTorrent with BOINC for data distribution are explored in [13]. However, it is noted that although this approach has proven to be quite scalable and efficient for larger files, without modification is not efficient for small files [33], which might be encountered in existing Desktop Grid environments. Additionally, BitTorrent 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. Although this requirement is quite effective in enforcing a "fair" sharing policy between network participants to prevent parasitic behavior [23], it might prove problematic for volunteer computing platforms where it might be acceptable to have a worker node's download disproportionate to its upload. Reasons for worker nodes not equality participating as upload partners are many, 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 [15], 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. This approach 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. 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 bandwidth of data servers 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, albeit with different parameters, a situation that can also exploit the overlay described here. The 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 [20] 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 increasing being widely adopted to deploy interconnections among nodes in distributed systems and specifically Grid infrastructures.

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:

- *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 job execution. Each data file is associated with a *data advert*, i.e., a metadata document that describes the characteristics of this file.
- a *job manager* produces *job adverts*, i.e., files that describe the characteristics of jobs that must be executed, and it is also responsible for the collection of resulting output.
- *workers* are 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 downloading of a data file or the execution of a job, that task will not be completed.
- *super-peers* constitute the backbone of the super-peer overlay. Super-peers are connected to workers through a centralized topology and connect with one another through a high level P2P network. In the protocol proposed here, 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 the associated data adverts. These data caching nodes retrieve data from the data source or other data caching nodes, and then can directly provide the data to worker nodes.

In the following, *data sources* and *data cachers* are collectively referred to as *data centers*, since both are able to provide data to workers. The difference between them being which phase of the process they are involved in, with data sources serving data from the very beginning and data cachers serving data only after retrieving it from another cacher or a data source.

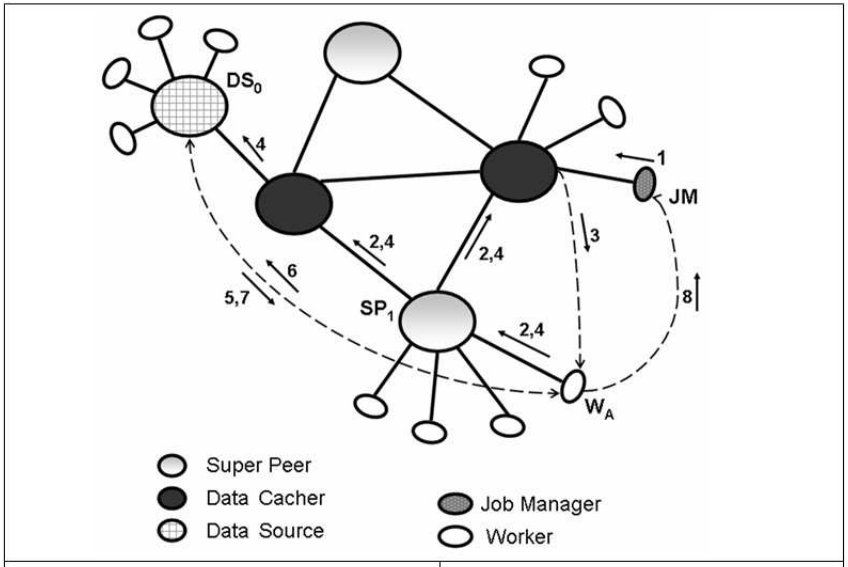
For the purposes of this experiment, we have assumed that only super-peers can act as data centers, however, the protocol can easily be extended to include the case in which even simple peers can store and provide data. We envisage that the same user-driven process is used to configure a peer; that is, each user can decide if a node will be a super peer or data center, as well as a worker. In the BOINC scenario, the existing dedicated machines would form the obvious core data-center backbone and other peers, preferably those with high storage and network capacities, would elect to also make themselves available to act in this mode.

3.1 Job Assignment and Data Download

Figure 1 depicts the messages sequence between workers, super-peers and data centers for the execution of the job submission protocol on a sample topology with five 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, SP_1, since it is not a data cacher. The behavior of the protocol with dynamic caching is explained in Section 3.2.

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 labeled 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 and possibly propagates them to other super-peers. 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 network through the super-peer interconnections (step 2). This continues 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 the 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, the 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



1. *job adverts*	5. *data advert*
2. *job query*	6. *data download request*
3. *job assignment*	7. *data download*
4. *data query*	8. *job results*

Fig. 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 here that data cachers have not yet stored data.

same file. Rather, the data center (in this example the data source DS_0) sends only a small *data advert* to the worker (step 5). In general, a worker can receive many data adverts from different data centers. The worker then chooses a data center, according to policies that could rely on the distance of data centers, their available bandwidth, or some other criteria. After making the choice, the worker initiates the download operation (step 6). Upon receiving the input data (step 7), the worker executes the job, reports the results to the job manager (step 8) and possibly issues another *job query*.

In both the *job-assignment* and *data-download* phases, the unstructured topology of the super-peer network can cause the unnecessary duplication of messages, for example two replica of the same query can be delivered to the same super-peer. To overcome this problem, and in general to limit the traffic load, a number of techniques are adopted. (i) The number of hops is limited by a Time-To-Live (TTL) parameter. For the application discussed in this paper, a TTL value equal to 5 proved to be sufficient. (ii) Each query message contains a field used to annotate the nodes that the query traverses along its path. A peer does not forward a query to a neighbor peer that has already received it. (iii) Each peer maintains a cache memory where it annotates the IDs of the last received query messages. A peer discards the queries that it has already received. Techniques (ii) and (iii) are used to avoid the formation of cycles in the query path, and are complementary, since technique (ii) can *prevent* cycles in particular cases (i.e. when a query, forwarded by a peer, is subsequently delivered to the same peer), whereas technique (iii) can *remove* cycles in other cases (e.g., when two copies of a query, sent by a peer A to two distinct neighbor peers B and C, are both subsequently delivered to the remote peer D).

3.2 Dynamic Caching

One of the main features of our super-peer protocol is the dynamic caching functionality, which allows for the replication of data input files on multiple data cachers. This leads to well known advantages such as an increased degree of *data availability* and improved *fault tolerance*. Dynamic caching also allows for a significant amount of time savings in the data download phase. This is because data queries have a greater chance to find an available data center, and most workers are able to download data from a neighbor data cacher instead of a remote data source. Moreover, load balancing among workers is improved, because the presence of multiple data cachers limits the possible overloading of workers that are located in the proximity of a data center.

The remaining part of this section illustrates the dynamic caching mechanism, while 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, labeled by 4_B, finds a match in the data source DS_0. As opposed to the case described in Figure 1, in this scenario 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 sent by DS_0 not directly to the worker W_B, but rather to DC_2. DC_2 then replicates and caches the file, passing it to the worker. Subsequently, DC_2 will act as a data source for the time period in which it maintains the data file in its cache. In this example, the data query issued by W_C, labeled by 4_C, can be served directly by the cacher DC_2 instead of the original data source DS_0.

To increase performance, a file splitting approach is adopted: data files are not downloaded as whole units, but are rather split into ordered fragments: 1 Mbytes size in this case. For example, if a query is received by data cacher DC_2 and it does not hold the entire data file, but has already received a part

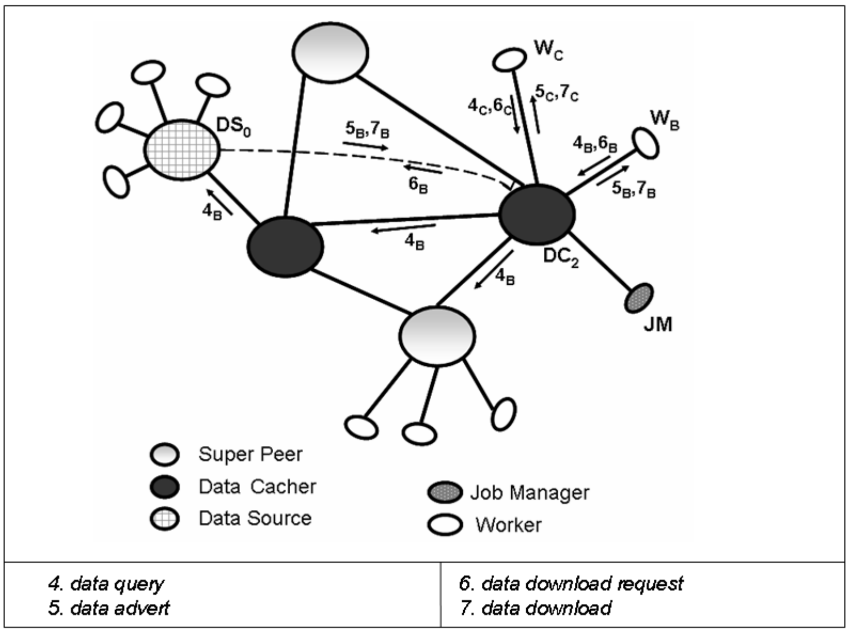


Fig. 2. Download phase of the super-peer job submission protocol, with dynamic caching. After the request of worker W_B, the data cacher DC_2 retrieves the data file from the data source DS_0, replicates and caches the file, and delivers it to W_B. Subsequently, the request of worker W_C is directly server by the data cacher DC_2.

of it from DS_0, it will not forward the data query, but rather will retain it, because it will soon receive the remaining fragments from DS_0. As it receives these fragments, DC_2 passes them to the requesting worker W_C.

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.

4 Performance Evaluation

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

The simulation scenario is described by the parameters described in Table 1. The parameter values of the representative astronomy scenario mentioned in Section 3 are used for the test case (for example, file size, job execution time, etc.).

The number of workers is set to 250, 500 and 1000 in different simulation tests. It is assumed that an average of 10 workers are connected to a super-peer, so the number of super-peers varies from 25 to 100. Data is initially provided by one data source and can be cached by several data cachers. The overall number of data centers (the data source plus the data cachers) varies from 1 to half the number of super-peers.

Workers can disconnect and reconnect to the network at any time. This implies that a data download or job execution fails upon the disconnection of the corresponding worker. Table 1 specifies the assumed average connection and disconnection times of workers.

The number of jobs varies from 100 to 1000. In our application scenario, each job corresponds to the analysis of a portion of the gravitational waveforms received from the detector. Scientific analysis can require multiple execution of every single job, either to enhance statistical accuracy or minimize the effect of malicious executions. The parameter N_{exec} is defined as the minimum number of executions that must be performed for each job and is set to 10 in this analysis. 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}. The job manager assigns a job 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. To compute the download time with a proper accuracy, a data file is split in 1 MB segments, as mentioned in Section 3.2, 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.

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 and return related results to the job manager. 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 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 is an important efficiency index that helps to evaluate the convenience of adding more data centers to the network.

Table 1
Simulation scenario

Scenario feature	**Value**
Number of workers, N_{peer}	250 to 1000
Average number of workers connected to the one super-peer	10
Average number of neighbors of a super-peer (power law topology)	4
Average connection time of workers	4 h
Average disconnection time of workers	1
Percentage of super-peers that are also data centers	up to 50%
Size of input data files	7.2 Mbytes
Latency between two adjacent super-peers	100 ms
Latency between a super-peer and a local worker	10 ms
Bandwidth between two adjacent super-peers	1 Mbps
Bandwidth between a super-peer and a local worker	10 Mbps
Number of jobs, N_{job}	100 to 1000
Number of executions requested for each job, N_{exec}	10
Matches to live, MTL	10 to 25
Mean job execution time	500 s

Another important figure is the network load caused by the exchange of messages among workers, super-peers and data centers. This was computed as the average number of messages transmitted on the network per time unit, and it is denoted as L_{net}.

Finally, load balancing among workers was analyzed, since an efficient and fair management of a scientific project should assign comparable computation loads to the different workers.

4.1 Redundant Submission of Jobs

A first set of simulation was performed for a network with 1000 workers and 100 super-peers, among which one is a data source and 49 are data cachers. The purpose is to investigate the effectiveness of the redundant submission of jobs, in other words the impact of the Matches To Live (MTL) parameter on performance indices. MTL is set to values ranging from 10 to 25, while N_{exec} is fixed to 10.

Figure 3 shows the overall execution time vs. the MTL value, with the number of jobs N_{job} ranging from 100 to 1000. The execution time tends to decrease as the value of MTL increases, then it gets stabilized. The reason of this is that a larger MTL allows to better compensate for the possible failures of jobs due to peer disconnections. This effect is not more evident when the MTL exceeds a threshold: in fact very large values of MTL are not exploited because the job manager stops the assignments of job adverts when output data related to N_{exec} executions have been received, as explained in the previous section.

Note also that the execution time could not be computed for values of MTL lower than 14 (or lower than 13 if N_{job} is 100). In such cases, worker disconnections do not allow to perform at least N_{exec} executions for each job. Therefore, the redundant approach is not only useful to decrease the execution time, but it is even necessary to complete the required job executions. An alternative approach could be based on the resubmission of jobs after the elapsing of a timer set by the job manager. However this approach would waste much more time than the approach examined here, which is based on a redundant job assignment made in advance by the job manager.

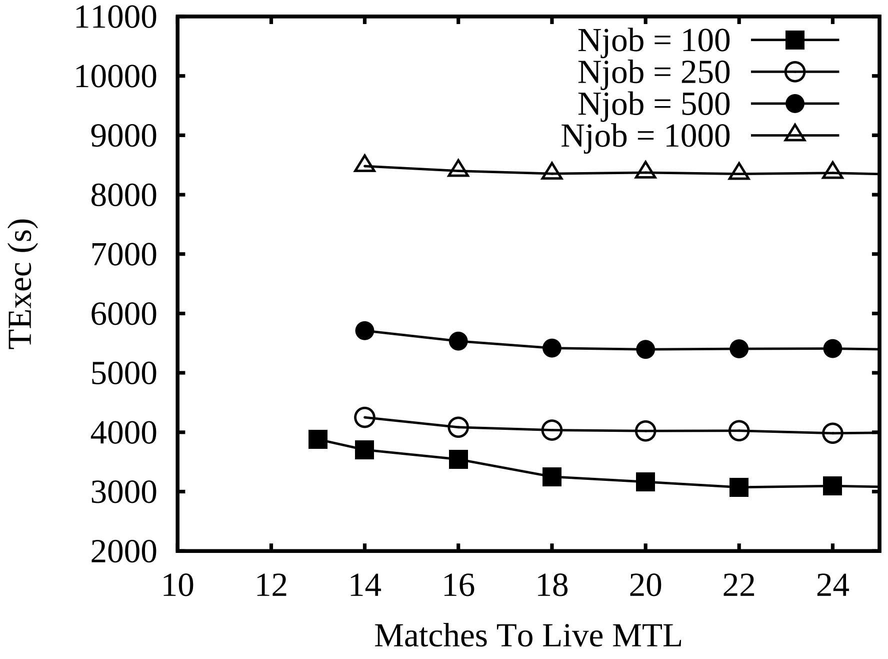


Fig. 3. Overall execution time vs. the value of MTL, for different numbers of jobs.

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., with the number of jobs and, more slightly, with the MTL value. To understand this, it must be considered that data cachers are not heavily utilized in the first phase of the process, because they have not yet retrieved data from the data source, whereas they are fully exploited only after they have retrieved such data. Therefore, the utilization of data centers

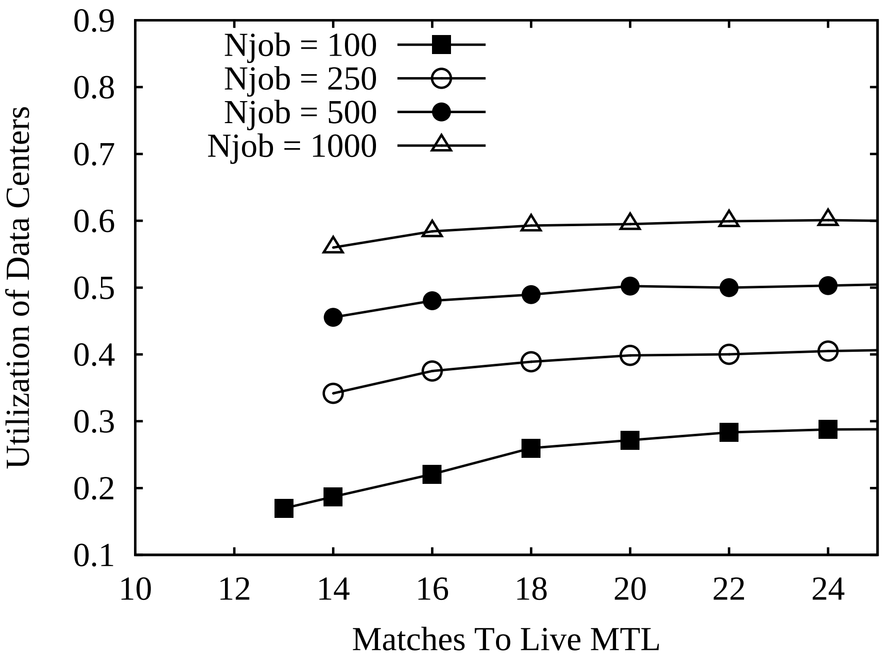


Fig. 4. Average utilization of data centers vs. the value of MTL, for different numbers of jobs.

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.

After this preliminary set of experiments, it was decided to set MTL to a constant value, in order to better evaluate the effect of other parameters and configuration options. The value of MTL was set to the lowest value for which the overall execution time is at most 10% higher that the "steady" execution time, for every tested value of N_{job}. This steady value was set as the execution time obtained with MTL equal to 50, beyond which no further variations of T_{exec} can be perceived. According to this strategy, the MTL was then set to 20. Furthermore, this value allows for the successful execution of jobs in most of the considered scenarios, as will also be discussed in the next section.

4.2 Performance of Data Caching

A second set of experiments aimed at evaluating the effectiveness of the dynamic caching mechanism described in Section 3.2. Simulations were performed for a network analogous to that examined in Section 4.1, except that the number of available data centers, N_{dc}, is varied from 1 to half the number of super-peers: one of these data centers is the data source, the others are data cachers. Furthermore, the MTL value was fixed at 20. This analysis essentially compares how our approach may affect a BOINC-like network if the administrator provides more data cachers into the network.

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.

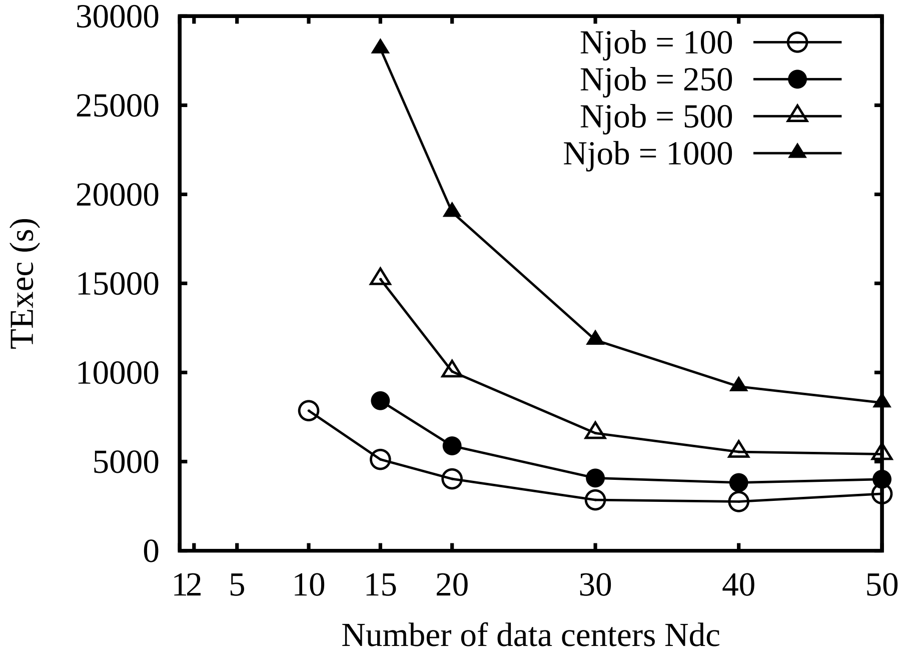


Fig. 5. Overall execution time vs. the number of data cachers, for different numbers of jobs.

Depending on the number of jobs to be executed, it is possible to determine a suitable number of data centers, beyond which the insertion of a further data center produces a very low decrease of execution time, or even a small increase of it. For example, if the number of jobs is 1000, a significant reduction of T_{exec} is perceived as the number of data centers is increased up to a value of 50, whereas if the number of jobs is 100 or 250, 30 data centers are sufficient to achieve a good performance level, and adding more data centers is not effective.

Figure 5 does not report results for some combinations of the number of data centers and the number of jobs, because the disconnections of workers do not allow for the completion of all the required job executions. Specifically, if the number of jobs is 250 or larger, the number of data centers should be at least 15, while, if the number of jobs is 100, 10 data centers are sufficient. 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.

It is very hard to deduce the overall execution time in an analytical way, due to the large number of network parameters (e.g., the number of workers and super-peers, the bandwidth and latency between nodes and so on, see Table 1) and the complexity of the super-peer protocol described in Section 3. However, we made several tests with different application scenarios and used Matlab tools to obtain a mathematical expression that is able to approximate simulation results as much as possible and at the same time is coherent with the dynamics of the protocol. We derived the expression shown in formula (1), that relates the overall execution time to the number of jobs to execute, N_{job}, and to the number of available data centers, N_{dc}.

$$T_{exec} = C_1 log(N_{dc}) + C_2 \frac{N_{job}}{N_{dc}} + C_3 \frac{N_{job}}{(N_{dc})^2} \tag{1}$$

Very interestingly, we found that this expression is valid for all the performed tests, regardless of the values of the other network parameters. Of course, the impact of these parameters is encompassed by the values of the coefficients that appear in formula (1).

The expression in the formula is composed of three terms, each of which can be associated to a basic characteristic of the protocol. In particular, the first term relates to the dissemination of input data to the network data centers. This term is logarithmic with respect to N_{dc}, because each data cacher, after retrieving data from a data center, is able to provide this data to a number of other data cachers. Due to the log-type relation, this term increases slightly with the number of data centers. The second term takes into account the time needed by workers to download data files from a single data center and execute the corresponding jobs (indeed we can consider one data center, since operations are made in parallel on different data centers). Specifically, this term is proportional to the average number of jobs which require a download operation from a single data center, N_{job}/N_{dc}. Finally, the third term gives an estimation of the additional amount of time that is required by worker disconnections. This "extra" time corresponds for the most part to the time taken by download operations that have to be re-executed because they failed during their first try. In fact, the third term comes out as the product of the time that would be taken if all the download operations failed (which is proportional to N_{job}/N_{dc}) and the probability that a single download operation actually fails. It was found that this probability is inversely proportional to the number of data centers, since download operations are longer and more at risk of failure as the number of data centers decreases. The third term takes into account only the possibility of repeating a download operation just once. The impact of multiple repetitions of download operations is actually negligible if the failure probability is much lower than 1. In conclusion, the overall execution time is the sum of three terms, of which the first increases with the number of data centers while the other two decrease. However, the effect of the first term is relatively low, except for the cases in which the number of data centers is large and the number of jobs is small, as can be seen in Figure 5. In fact, in such cases the overall execution time slightly increases with the number of data centers.

Figure 6 shows the average utilization of data centers for the same scenario. This index decreases as the number of data centers increases and, in contrast with the execution time, curves do not get to a relatively stable value. This is another useful indication for setting a proper number of data centers. For example, consider the submission of 500 jobs. While the overall execution time can be decreased until the number of data centers is increased to about 40, the utilization index continue to decrease as more data centers are made available. With 50 data centers there would be a worse exploitation of data centers and no significative reduction in the execution time, from which it can be concluded that an appropriate number of data centers is indeed 40.

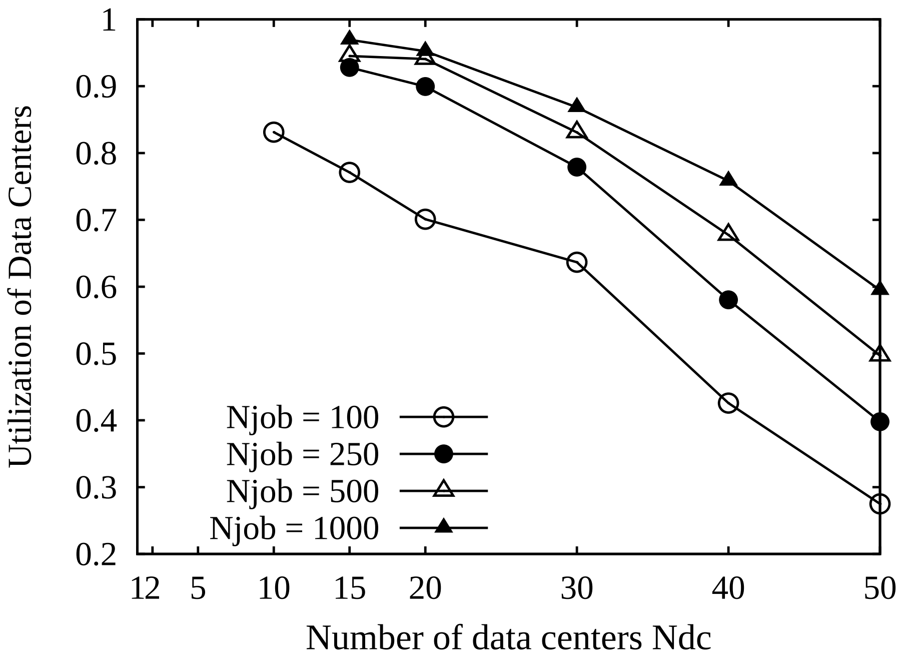


Fig. 6. Utilization of data centers vs. the number of data cachers, for different numbers of jobs.

Figure 7 shows the number of messages per second that circulate on the network. It can be noticed that the network load gets larger with the number of jobs, but the relative increase is much less than proportional. For example, if the number of data centers is 40, and the number of jobs is doubled from 500 to 1000, the network load only increases from 504 to 533 messages per second. This is a first hint about the good scalability characteristics of the protocol, that will be further examined in Section 4.3.

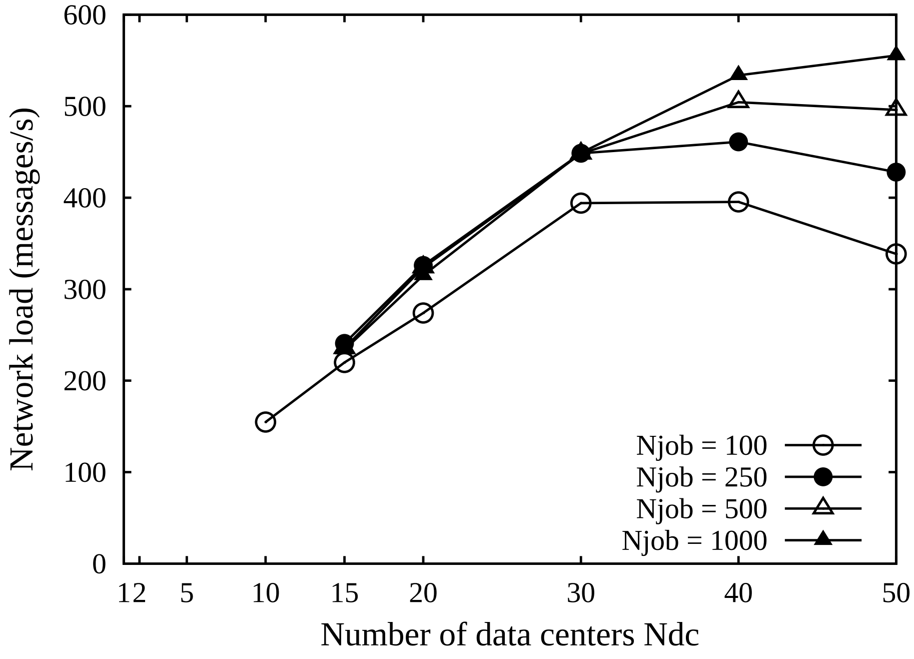


Fig. 7. Network load vs. the number of data cachers, for different numbers of jobs.

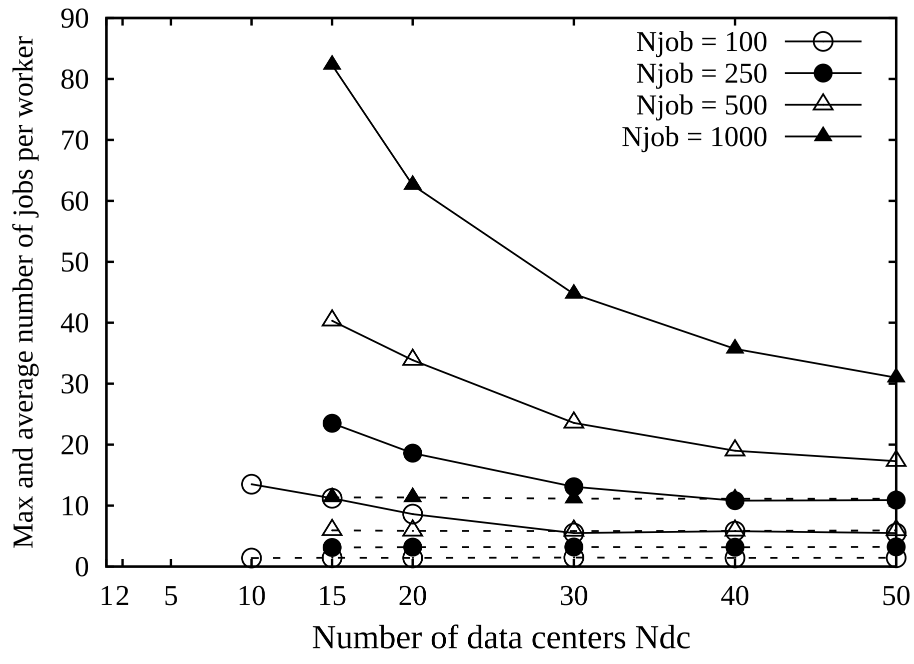


Fig. 8. Maximum and average number of jobs performed by a worker vs. the number of data cachers, for different numbers of jobs. The two indices are represented, respectively, with continuous and dashed lines.

Figure 8 helps to examine the load balancing features of the protocol. Figure

8 reports the maximum number of jobs executed by a single worker (i.e., the number of jobs performed by the most active worker), and the average figure, calculated as the overall number of job executions divided by the number of workers. It is interesting to note that the a wider availability of data centers improves load balancing among workers, as the maximum number of jobs executed on a worker decreases and approaches the average number. 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 available, the differences among workers are attenuated, in particular when the overall computation load is high.

Two more issues are strictly correlated with the use of data caching algorithms: the choice of the data replacement policy and the frequency of miss/hit events. In the application example examined in this paper, as well as in many other scientific applications, data is "read only" and is never modified by workers. Therefore, a sophisticated data replacement policy is not actually required. It is only necessary to provide the data source, and subsequently the data cachers, with the gravitational waveforms data as long as it is retrieved by the detectors. Of course, as old data has been thoroughly processed by workers, it can be discarded by the data source and the data cachers, to make more room for new data. This coarse-grained replacement process is controlled by the Job Manager node.

A *cache miss* event occurs whenever a data cacher contacted by a worker has not yet retrieved data, and must download it from another data cacher or from the data source. Conversely, if the data cacher has the data, a *cache hit* event occurs. As data is disseminated among the data cachers, the frequency of hit events becomes increasingly high. This is confirmed by Figure 9, which shows the percentage of cache hits in the first 500 seconds of execution of the applications. Of course, this percentage gets to 100% as soon as data has been retrieved by all the data cachers. it is also interesting to notice that in the transient phase the percentage of hit events decreases as more data cachers are made available on the network, since a longer time interval is necessary to distribute data to the cachers. Therefore, the presence of a large number of data cachers can slow down the execution of jobs in the very first phase, during the distribution of data to the cachers, but obviously this slowdown is immediately compensated in the steady phase, when workers can find the required data on any cacher of the network.

4.3 Scalability Analysis

An additional set of simulations were performed to evaluate the behavior of the protocol in variable-sized networks, to specifically examine its scalability.

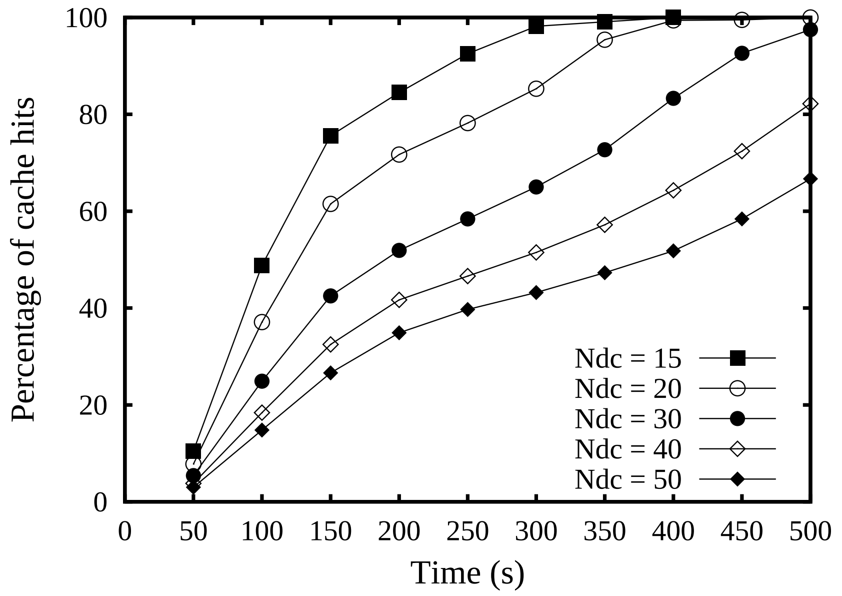


Fig. 9. Frequency of cache hit events vs. time, for different numbers of data cachers, with N_{job} equal to 1000.

We analyzed Grids having 250, 500 and 1000 workers, that is with 25, 50 and 100 super-peers, respectively. As in the previous simulations, one data source is available, while the overall number of data centers, including this data source and the data cachers, is varied from 1 to half the number of super-peers (the maximum value is approximated to 13 in the case of 25 super-peers). The required number of executions of each job, N_{exec}, is set to 10, while the maximum number of assignments per job, MTL, is set to 20.

Two different scenarios are taken into consideration. In the first, the number of jobs is constant, and set to 500, while the number of workers is increased: this is useful to verify if the availability of more workers can actually improve performance. In the second scenario, the number of jobs and the number of workers are increased with the same pace: this analysis is particularly useful to verify if the protocol is able to sustain an increase in the problem size in the case that the average computational load of a worker is maintained constant.

Figure 10 shows results related to the first scenario. Because of the high range of the values obtained, a logarithmic scale is adopted for the y axis. This figure shows that, with a constant problem size, that is, a fixed number of jobs, the availability of a larger number of workers actually improves performance, on condition that a proportional number of data centers are installed. For example, if half the number of super-peers are set up as data centers, the execution time is reduced by about 50% (from 11023 seconds to 5421 seconds) if the number of workers is increased from 250 to 1000. The performance gain is therefore relevant.

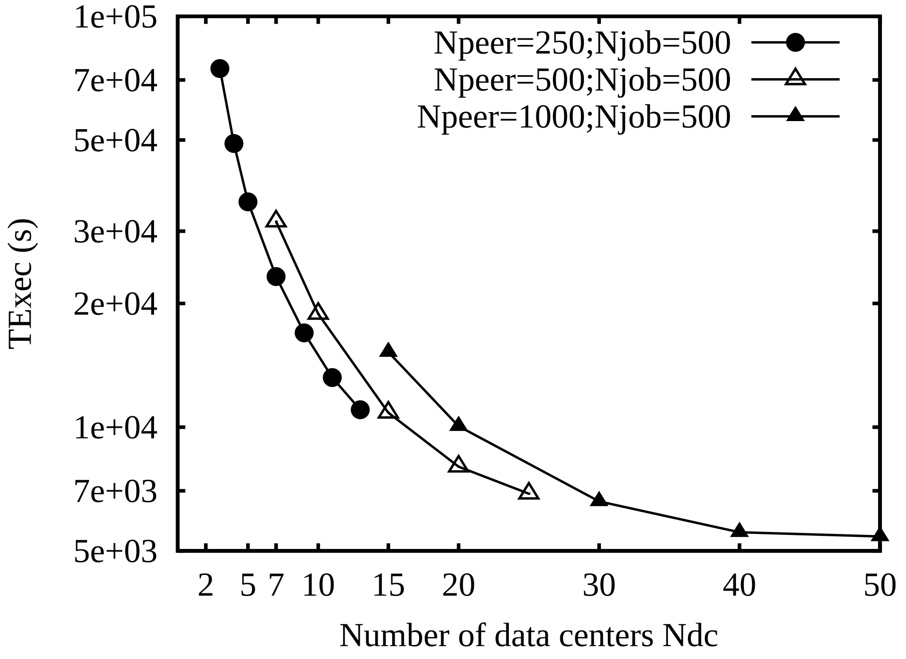


Fig. 10. Overall execution time vs. the number of data centers. The number of jobs is set to 500, and different network sizes are tested.

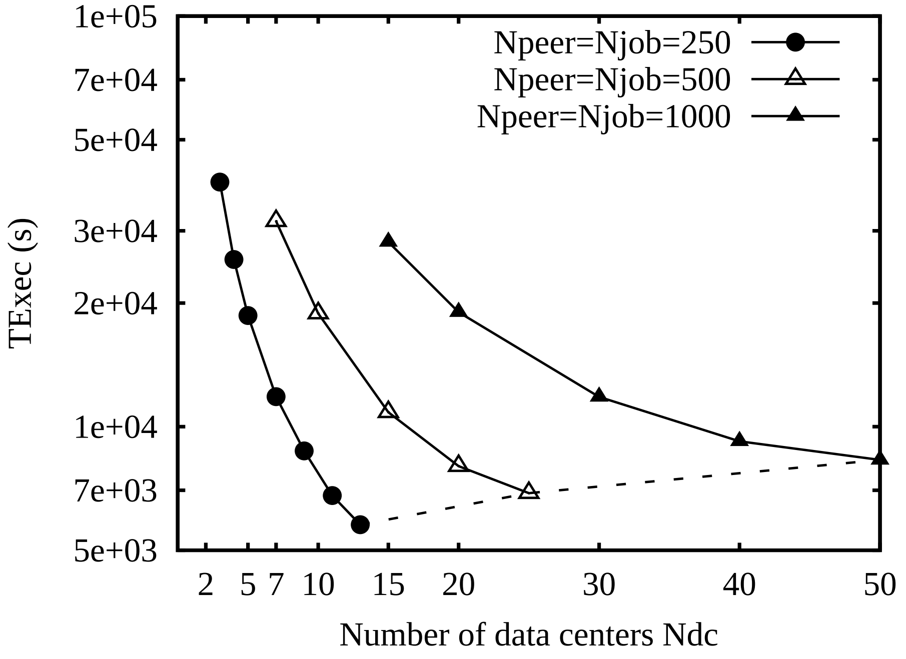


Fig. 11. Overall execution time vs. the number of data centers, for different numbers of jobs. The network size is proportional to the amount of computation.

Figure 11 reports the overall execution time for the second scalability test, namely the test with variable problem size, which aims to verify if it is effective to increase the number of workers proportionally to the amount of computation load. Here the number of workers is maintained equal to the number of jobs, each of which - it is useful to recall - must be executed at least 10 times. Again, the results obtained when using a fixed percentage of data centers are to be compared. The dashed line in figure 11 connects points obtained when the mentioned percentage is set to 50% and shows that the proposed approach is satisfactory scalable. In fact, the execution time is equal to 5770 seconds when the number of jobs (and the number of workers) is 250, while it increases to 6886 seconds with 500 jobs, and to 8301 with 1000 jobs. Therefore, if the amount of computation is doubled, the execution time increases by only 19%, while if it is quadruplicated, the execution time increases by only 43% per cent. Analogous results are obtained for different percentages of data centers in the network. This proves the good scalability of the approach presented in this paper, as the pace at which the execution time increases is much lower than the corresponding increase in the problem size.

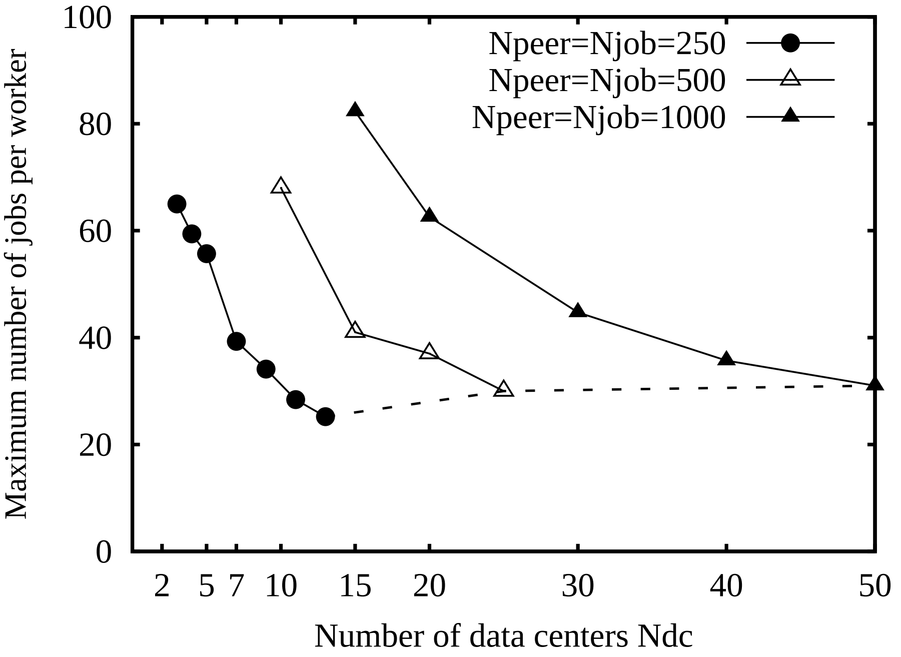


Fig. 12. Maximum number of jobs performed by a worker vs. the number of data centers, for different numbers of jobs. The network size is proportional to the amount of computation.

Scalability actually derives from the ability of the protocol to fairly distribute the computational load to workers, even when the problem and the network size increase. This can be observed in Figure 12: the dashed line highlights the values of the maximum number of jobs executed by a worker, obtained when the number of data centers is equal to 50% of super-peers. The maximum number of jobs is 25.2 in a network with 250 workers, and increases very slightly with the computational load: specifically, it increases to 30.0 and to 31.1 as the number of jobs increases to 500 and 1000, respectively.

5 Applied Decentralized Data Caching for Applications

There are many different applications and communities that would benefit from the decentralized and flexible data sharing techniques described in this paper. With new data management technologies, scientific users would be able to use Desktop Grids to explore new types of data-intensive application scenarios that were overly prohibitive before, given their large data transfer needs. In this section we will briefly describe three application areas that we are currently working with to enable data access in this manner. In the process of providing these case-studies, we will attempt to briefly show the reader the current limitations of these systems, and how P2P data sharing would further enable these application areas.

5.1 Einstein@HOME

The Einstein@HOME [17] is a scientific data analysis project employing the BOINC infrastructure to analyze gravitational wave data. Currently the project is employing the traditional HTTP server mirroring solution used by other BOINC projects to distribute its data sets. The system is able to cope with user demand with a small number of data mirrors, however, if the data input is significantly increased through new detectors, or larger data sets are introduced to perform higher resolution analysis, these data mirroring solutions can quickly become prohibitively costly. By applying a P2P approach for this project, it would allow the system to scale proportionately as the load or number of users increases, thus make the system completely dynamic and self-organising. One key advantage of such an approach is that maintenance and administrative overhead is virtually non-existent and it therefore becomes extremely inexpensive to run and to introduce increased data or explore new algorithms that might require a more robust data intensive solution.

5.2 Distributed Audio Retrieval

The Distributed Audio Retrieval using Triana (DART) [16] project is a joint collaboration between Cardiff University and the Laboratory for Creative arts and Technologies (LCAT) at Louisiana State University (LSU). DART is working to build a decentralized overlay for processing audio information through its Music Information Retrieval (MIR) mechanisms. To do this, the system makes use of peer-to-peer technologies to create a network, similar to BOINC projects. In DART, users provide CPU cycles for analysis of their local audio files, which provides audio metadata to the network that is then used to enable the system to make music recommendations based on collaborative filtering and music information retrieval techniques. The local processing on the network participants is achieved though the use of Triana workflows that can be uploaded to the members in the network.

The DART music recommendation system therefore is a process-intensive application, where potentially thousands of audio files (typically in MP3 format) can be analyzed on users' machines. The algorithms for statistical and audio-based analysis are Triana workflows. These workflows are bundled into a semi-self contained package and propagated onto the network for execution. One large component to this system is the fanning out of the workflow descriptions and associated code to all of the peers in the network for analysis. Since such code might comprise of dozens of complex spectral and temporal analysis tools, such workflow packages can be tens of megabytes in size. Therefore, to support the scale of DART's intended audience (potentially tens of thousands of participants), there is a need to avoid any centralized structure for content distribution. Rather, a decentralized, yet secure data center approach like the one described in this paper would sufficiently solve the project's data needs.

5.3 Enabling Desktop Grids for e-Science (EDGeS)

EDGeS is a new FP7 funded infrastructures project to build bridges between Desktop Grid and Service Grid systems and enable their interoperability. The EDGeS project started in January 2008, and is working to develop tools to bridge the technologies on computational Grids with desktop Grids. The Aneka framework [10] relates to some degree to EDGeS in that it provides a framework for creating services that can support multiple overlays, each supporting a desktop Grid. However, EDGeS differs by providing a translational toolkit for deploying applications written for a Grid infrastructure on Desktop Grid and vice verse. Specifically, the consortium will interconnect the largest European Service Grid infrastructure (EGEE) with existing desktop Grid systems, such as BOINC and XtremWeb. At the core of the EDGeS toolkit is a bridge mechanism that will enable users to transparently execute applications on any arbitrary platform involved in the EDGeS infrastructure.

With respect to the EDGeS data management, one of the hurdles to this integration is the ability to transfer traditional jobs that would run on Service Grids to Desktop Grid environments due to the bandwidth and space requirements that would be imposed upon the Desktop Grid nodes. Unlike a traditional Desktop Grid project, which has a centralized server providing data to worker nodes, in the EDGeS project, jobs would be sent to a Desktop Grid from a Service Grid through a bridge mechanism that translates the job to enable it to be enacted on a Desktop Grid environment. In this scenario, a new network requirement would be placed upon Service Grid systems and the techniques described in this paper provide useful input to the ongoing development of the core P2P data sharing infrastructure [32][28] that is being used as the basis for the data center layer of EDGeS.

6 Conclusions

In this paper we presented a decentralized architecture for data-intensive scientific computing and evaluated it. 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 entire spectrum of client-side capabilities in these types of networks, where participants generally not only have 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 large number of dispersed worker nodes execute the required tasks, while a smaller group of nodes are configured to store and provide input data files to workers. Job assignment and data distribution is performed through the rendezvous role of super-peers, which match job and data queries, issued by workers, to job and data adverts that describe the characteristics of tasks and data files.

To provide support for this scheme, a number of simulations have been performed to evaluate the impact of application and network parameters on performance indices such as the overall time to execute and network load. Results for our test-case show availability of several data centers and the use of dynamic caching brings benefits to applications, including scalability. 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, helping to maximize the return on investment when deploying new data centers. By using a system like the one 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. This 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.

7 Acknowledgements

This research work is carried out under the FP6 CoreGRID Network of Excellence which is funded by the European Commission (Contract IST-2002-004265). We would also like to thank Andrew Harrison for his 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. Volunteer computing: Planting the flag. In *Proceedings of the PCGrid 2007 Workshop*, Long Beach, California, USA, March 2007.

[4] 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.

[5] Grid Today article. Folding@home recognized by guinness world records. October 31st, 2007. See Web site at http://www.gridtoday.com/grid/1866374.html.

[6] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, October 1999.

[7] BOINC. Berkeley open infrastructure for network computing. See Web site at http://boinc.berkeley.edu/.

[8] Franck Cappello, Samir Djilali, Gilles Fedak, Thomas Herault, Frederic Magniette, Vincent Neri, and Oleg Lodygensky. Computing on large-scale distributed systems: Xtrem web architecture, programming models, security, tests and convergence with grid. *Future Generation Computer Systems*, 21(3):417–437, 2005.

[9] Andrew Chien, Brad Calder, Stephen Elbert, and Karan Bhatia. Entropia: architecture and performance of an enterprise desktop grid system. *J. Parallel Distrib. Comput.*, 63(5):597–610, 2003.

[10] Xingchen Chu, Krishna Nadiminti, Chao Jin, Srikumar Venugopal, and Rajkumar Buyya. Aneka: Next-generation enterprise grid platform for e-science and e-business applications. In *E-SCIENCE '07: Proceedings of the Third IEEE International Conference on e-Science and Grid Computing*, pages 151–159, Washington, DC, USA, 2007. IEEE Computer Society.

[11] Climateprediction.net. See web site at http://climateprediction.net/.

[12] B. Cohen. Incentives Build Robustness in BitTorrent. In *Proceedings of the First Workshop on Economics of Peer-to-Peer Systems*, June 2003.

[13] Fernando Costa, Luis Silva, Gilles Fedak, and Ian Kelley. Optimizing the Data Distribution Layer of BOINC with BitTorrent. Submitted to the 2nd Workshop on Desktop Grids and Volunteer Computing Systems PCGrid, 2008.

[14] 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.

[15] 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.

[16] DART. The DART music recommendation system. See Web site at http://www.mrsdart.com/.

[17] Einstein@home. See web site at http://einstein.phys.uwm.edu/.

[18] Gilles Fedak, Cecile Germain, Vincent Neri, and Franck Cappello. Xtremweb: A generic global computing system. In *Proceedings of the IEEE Int. Symp. on Cluster Computing and the Grid*, Brisbane, Australia, May 2001.

[19] Folding@HOME. See web site at http://folding.stanford.edu/.

[20] GridOneD. See web site at http://www.gridoned.org/.

[21] Mikel Izal, Guillaume Urvoy-Keller, Ernst W Biersack, Pascal A Felber, Anwar A Hamra, and Luis Garces-Erice. Dissecting BitTorrent: Five months in a torrent's lifetime. In *Proceedings of the 5th Passive and Active Measurement Workshop*, 2004.

[22] KaZaA. See web site at http://www.kazaa.com/.

[23] Thomas Locher, Patrick Moor, Stefan Schmid, and Roger Wattenhofer. Free Riding in BitTorrent is Cheap. In *Fifth Workshop on Hot Topics in Networks (HotNets-V)*, Irvine, CA, US, November 2006.

[24] Akshay Luther, Rajkumar Buyya, Rajiv Ranjan, and Srikumar Venugopal. Peer-to-peer grid computing and a.net-based alchemi framework. In *High Performance Computing: Paradigm and Infrastructure, Laurence Yang and Minyi Guo (eds)*, pages 403–429. Wiley Press, New Jersey, 2005.

[25] 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.

[26] Alberto Montresor. A robust protocol for building superpeer overlay topologies. In *Proceedings of the 4th International Conference on Peer-to-Peer Computing P2P'04*, pages 202–209, 2004.

[27] Napster. See web site at http://www.napster.com/.

[28] P2P-ADICS. Peer-to-peer architecture for data-intensive cycle sharing. See Web site at http://www.p2p-adics.org/.

[29] German Sakaryan, Markus Wulff, and Herwig Unger. Design and implementation tradeoffs for wide-area resource discovery. In *Proc. of the Innovative Internet Computing Conference, IICS '04, Springer LNCS, Vol. 3473*, Guadalajara, Mexico, June 2004.

[30] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications SIGCOMM'01*, pages 149–160, 2001.

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

[32] Ian Wang. P2PS (Peer-to-Peer Simplified). In *Proceedings of the 13th Annual Mardi Gras Conference - Frontiers of Grid Applications and Technologies*, February 2005.

[33] Baohua Wei, Gilles Fedak, and Franck Cappello. Scheduling independent tasks sharing large data distributed with bittorrent. In *Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing*, pages 219–226, 2005.

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

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