CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2011; **23**:458–476
Published online 5 November 2010 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/cpe.1657

Grid and P2P middleware for wide-area parallel processing

Fatos Xhafa[1, *, †], Sabri Pllana[2], Leonard Barolli[3] and Evjola Spaho[4]

[1]*Languages and Informatic Systems, Technical University of Catalonia, Campus Nord, Ed.Omega C/Jordi, Girona 1-3, Barcelona, Catalonia 08304, Spain*
[2]*Department of Scientific Computing, University of Vienna, Nordbergstrasse 15/C308, 1090 Vienna, Austria*
[3]*Department of Information and Communication Engineering, Fukuoka Institute of Technology, 3-30-1 Wajiro-higashi, Higashi-ku, Fukuoka 811-0295, Japan*
[4]*Graduate School of Engineering, Fukuoka Institute of Technology (FIT), 3-30-1 Wajiro-higashi, Higashi-ku, Fukuoka 811-0295, Japan*

SUMMARY

Grid computing emerged as a paradigm for high-performance computing and massive parallel processing. Currently, Grid systems have become an important paradigm for efficiently solving large-scale complex problems from many fields. On the other hand, P2P paradigm originated from file-sharing, but each time more is being used for the development of large-scale distributed platforms. Grid and P2P systems have thus followed different trajectories pushed by different motivations, needs and research communities. In fact, both paradigms are evolving in a way that each time they are sharing more common characteristics and are mutually benefiting from their best features. Among these characteristics, we could distinguish the cooperative model for solving complex problems by exploiting their large computing capacity. As such, Grid and P2P systems have achieved notable success, in particular, for e-Science applications, a family of complex applications arising in science and engineering that need considerable computation power. Despite important advances in the design and use of Grid and P2P systems, they remain still difficult to implement and apply to real-life problems. The main difficulties reside in the lack of easy-to-use middleware for Grid and P2P, in the complexities of setting up and in the tedious task of deploying real-world Grid/P2P platforms as well as in experimental studies which are often complex and not easy to repeat. In this paper we survey and analyze the advances in communication libraries and middleware for both Grid and P2P systems as well as their limitations when used in real Grid and P2P infrastructures. We also bring examples of real-life applications of massive data processing that can be efficiently handled through Grid and P2P approaches. Copyright © 2010 John Wiley & Sons, Ltd.

Received 6 July 2010; Accepted 9 July 2010

KEY WORDS: Grid computing; P2P computing; parallel processing; middleware; communications libraries; e-Science applications

1. INTRODUCTION

Computational Grids and *P2P systems* have emerged as new distributed computing paradigms for the development of large-scale distributed applications. With the fast developments in Internet technologies and with the continuous increase in the connected computational resources, Grid and P2P appeared as the disruptive technologies that can greatly impact not only on scientific and academic activity but also on business and enterprise productivity. The rationale is that such

*Correspondence to: Fatos Xhafa, Languages and Informatic Systems, Technical University of Catalonia, Campus Nord, Ed.Omega C/Jordi, Girona 1-3, Barcelona, Catalonia 08304, Spain.
†E-mail: fatos@lsi.upc.edu, fatos@dcs.bbk.ac.uk

Contract/grant sponsor: General Secretariat of Universities of the Ministry of Education, Spain

technologies are *inexpensive* (often built up and maintained in a contributory way), technologically *manageable* and easy to maintain and extend, thus reducing the needs for replacement of the existing systems or acquisition of new infrastructures.

The demand for more computing power has been a constant trend in many fields of science, engineering and business. Now more than ever, the need for more and more processing power is raising in the resolution of problems from life sciences, financial services, drug discovery, weather forecasting, massive data processing for e-Commerce and e-Government, etc. Grid and P2P computing are based on the premise to deliver greater computing power at less cost enabling thus the solution of complex problems from many fields of science, engineering and business.

Grid Computing. The term Grid computing was popularized in 1990s [1, 2] to express the 'computing power Grid' in analogy to an electric power Grid. Computational Grids were motivated by the need to develop computational frameworks to support large-scale applications that benefit from the computing power offered by such distributed infrastructures. As a matter of fact, the first successful stories of Grid-enabled applications are found from scientific computing domain. NetSolve/GridSolve [3] is among the first Grid middleware used for high-performance computing. Another example of middleware for Grid computing is that developed by the MetaNEOS[‡] project, essentially aimed for solvers running on Condor and Globus [4]. Interfaces based on Master–Worker meta-computing applications were also provided. However, such interfaces (MW, iMW, iNEOS, PBB and Modeling/Condor) are not integrated into one middleware and are intended for different families of optimization problems (meta-computing applications, nonlinear optimization, branch-and-bound algorithms, optimization problems [5] and stochastic programming [6]).

Owing to advances in the development of middleware for Grid computing, such as Globus, MPI-Grid2 and Condor-G as well as to the continuous improvement in hardware and networking communications, we are witnessing an explosion in research projects an initiative related to Grid computing. Nowadays, we can find several types of Grids such as *Compute Grids*, *Data Grids*, *Science Grids*, *Access Grids*, *Bio Grids*, *Sensor Grids*, *Cluster Gr1ids*, *Campus Grids*, *Tera Grids*, *and Commodity Grids*, among others, that support high-performance real-life applications. Such applications are showing the feasibility of Grid-enabled approaches for scientific and high-performance computations for a wide range of e-Science applications including parametric modelling [7], parameter sweep applications and Monte-Carlo simulations [8], computing-intensive applications in physics and life sciences [9, 10] and data-intensive computing applications [11]. A common feature of all these applications is that they are embarrassingly parallel, that is, they can be split into a set of independent or loosely coupled tasks, which are processed in parallel.

P2P systems. P2P systems [12, 13] appeared as a new paradigm after client–server and web-based computing. P2P systems became quite popular for file-sharing among Internet users through Napster, Gnutella, FreeNet, BitTorrent and other similar systems. Clearly, the motivation and the community of users and developers behind P2P systems have been different from that of Grid computing. Foster *et al.* stressed the difference between Grids and P2P systems by stating that '...the sharing that we are concerned with is not primarily file exchange but rather direct access to computers, software, data, and other resources, as is required by a range of collaborative problem-solving and resource brokerage strategies emerging in industry, science, and engineering'.

Differently from centralized or hierarchical models of Grid systems, in P2P systems nodes (peers) have equivalent capabilities and responsibilities and can be both servers and clients. Decentralization is thus an important feature of P2P systems as it is also their large scale as compared with small to moderate size of Grid systems. Although motivated by file-sharing, the improvements of P2P protocols are enabling the development of P2P applications other than file-sharing. For instance, Sun's JXTA library [14] and MPJ [15] are enabling the development of P2P large-scale distributed applications. One important initiative in this direction is that of SETI@home—the largest distributed computing effort with over 3 million users—which harnesses the power of networked PCs worldwide, through CPU scavenging and volunteer computing [16] of under-utilized desktop PCs [17]. The resulting P2P systems are used for solving CPU-intensive research problems

[‡]http://www-unix.mcs.anl.gov/metaneos/project/index.html.

such as protein folding. It should be noted, however, that in the P2P distributed project, the performance is usually 'best effort'-like when compared with the high-performance Grid. Nonetheless, P2P systems each time are more able to leverage high rates of throughput (currently processing at rates of few Terabytes), which is significant processing power for data-intensive applications.

Until now, there has been little work done to apply P2P systems to real-word applications, mainly due to the lack of easy-to-use middleware and difficulties in programming and the tedious task of deployment of large P2P systems.

Grid and P2P integrated approaches. Computational Grids and P2P systems still remain difficult for many users [18], mainly, because Grid/P2P middleware provide fundamental services but they are low level, making thus difficult their use in the development of Grid and P2P applications. Moreover, such middleware are not complete regarding the needs of different domain applications, requiring thus some *ad hoc* development. All in all, researchers on Grid and P2P systems are addressing issues that concern both systems, which will yield to new views of large-scale distributed application development in such systems.

In this paper, we survey and analyze the advances in communication libraries and middleware for both Grid and P2P systems as well as their limitations when used in real Grid and P2P infrastructures. We also exemplify the usefulness of P2P approaches for massive parallel processing of large size log data files generated in collaborative environments such as virtual campuses or IT enterprises.

The remainder of the paper is organized as follows. In Section 2, we overview the use of Grid paradigm for parallel processing. The P2P counterpart is presented in Section 3. The use of Grid paradigm for e-Science applications is tackled in Section 4. The common characteristics and lessons learned from both paradigms are presented in Section 5. In Section 6 we bring an application example of massive processing of large size log data files using a P2P approach. We end this paper in Section 7 with some conclusions.

2. GRID PARADIGM FOR PARALLEL PROCESSING

In a broader context of Grid systems, Bal *et al.* [18] motivated the need for research from conceptual, algorithmic and Grid application-level tools to facilitate the application development task. As claimed by authors, there is a need for tools, namely *Grid application-level tools*, that enable writing, deploying and running Grid applications. A set of requirements on Grid tools is commonly accepted: Grid tools should be built on the Grid software infrastructure, should isolate users from the dynamics of the Grid infrastructure, should reduce costs and complexities and should be generic and easy to use facilitating thus the application development also to users and practitioners non-familiar with the Grid technology. For instance, it is desirable for Grid users to be able to manage the entire application from a single interface. In particular, Grid systems should account for efficient resource allocation and scheduling, that is, accept user or application jobs, schedule jobs to execution hosts and provide statistical results upon completion of jobs.

At the application level, Grid paradigm can be exploited for different purposes:

High-throughput computing: the interest is to complete the largest number of applications per time unit. It should be noted that in such a case, we have a coarse-grained parallelism whereas fine-grained is not necessarily exploited. Quality-of-Service (QoS) system requirement could be an important indicator in this case.

High-performance computing: the interest is to reduce the execution time of applications. To achieve this speed-up goal, exploitation of fine-grained parallelism is necessary. User requirement on fast completion of applications could be important as well in this case.

QoS-aware computing: the objective is to develop Grid-enabled applications that will ensure QoS support for a series of parameters such as response time guarantees and cost. Such applications commonly arise in businesses aiming to minimize the cost of running business critical applications and support customer QoS demands.

How can these purposes be achieved? Kra [19] at IBM's Grid computing identified six levels of Grid-based applications, grouped according to batch vs service-based: (1) batch anywhere; (2) independent concurrent batch; (3) parallel batch; (4) service; (5) parallel service and (6) tightly coupled parallel program. As can be seen, the levels of Grid-enabled applications go from independent or loosely coupled applications to tightly coupled ones.

Much of the current research effort are devoted to bridging the gap between the existing Grid middleware and application-level needs. The development of Grid application software, such as GrADS [20], aims at facilitating and simplifying the development, execution, monitoring and tuning applications on the Grid.

2.1. Grid middleware and communication libraries

A major concern in Grid systems is that of dealing with the high degree of heterogeneity of resources. The unified virtual view of Grid systems and the efficient and transparent access to computational resources are achieved by Grid middleware. Grid middleware aims at integrating heterogenous resources, abstracting from low-level characteristics, efficient assignment of resources, job/application allocation and execution, monitoring, data access and transfer and secure access. Grid middleware can be seen as distributed software that enables the communication between applications and the underlying computational platforms and, as such, it should be as complete and generic as possible in terms of operations/services offered to match any needs of applications to be build on top of it. Owing to such vital functionalities, Grid middleware is considered as the backbone of Grid computing systems.

Considerable research efforts are currently devoted to the design and implementation of Grid middleware. Since Grid systems are natural continuation of parallel and distributed systems, the existing middleware and communication libraries used in parallel and distributed systems, such as Condor, MPI and PVM, were extended to support Grid-enabled applications. Certainly, the existing libraries show several limitations when faced to Grid infrastructures; therefore, new middleware and communication libraries are proposed by the Grid computing community to cope with new characteristics of such systems and thus match the needs of the development of large-scale Grid applications. Below we briefly consider some of them, actually widely used by the Grid computing community.

MPI-based middleware. Message Passing Interface (MPI) is established as the main communication libraries in the development of parallel scientific applications. Message passing paradigm is more appropriate for tightly coupled parallel applications rather than loosely coupled Grid model; however, new versions of the library appeared upon Grid computing attempting to improve the performance of the MPI collective communications for fault tolerant MPI implementations. MPICH-G2 [21] is a Grid-enabled implementation of the MPI, which allows to couple multiple heterogenous machines to run MPI applications. PArallel Computer eXtension to MPI (PACX-MPI) [22] enables an MPI application to run on a *meta computer* consisting of several, possibly heterogeneous machines, each of which may itself be massively parallel. MagPIe library [23] implements MPIs collective operations with optimizations for cluster wide-area systems. FT-MPI [24] is another implementation of MPI aiming to overcome MPI static process model and controlling failure modes by the applications.

Java RMI-based middleware. Owing to its platform independence feature, Java is expected to be of growing importance for the development of Grid middleware. MPJ (MPI Java) [15] is the MPI-like Message Passing for Java, attempting to establish a standard Java parallel programming APIs after MPI. Other efforts use Java's RMI. GMI [25] is a generalization of Java's RMI for parallel programming, whereas RepMI [26] is a compiler-based approach for object replication in Java. It implements a MagPIe-like broadcast operation for Grid environments.

NetSolve/GridSolve. NetSolve/GridSolve [3] are among the first Grid middleware used for high performance computing. NetSolve aims at using the best computational resources on a network. Fault-tolerance and load-balancing are also features of the NetSolve system to achieve high performance. It should be noted however that NetSolve is based on the client–server RPC model and has thus the limitations of a centralized system in contrast to full-featured Grid systems that are cheaper than a server farm approach. It specializes for coarse-grain task parallelism. Another limitation of NetSolve is the lack of universal interface; its interface protocol can only be applied to the clients of the NetSolve system.

Globus and Globus-based middleware. Globus Toolkit [4] is the most widely used middleware for Grid applications. It allows finding resources, running applications, dealing with heterogeneity and security issues. It is an integrated toolkit of Grid services offering resource allocation and process management, communication services, authentication and security services, system monitoring, remote data access, etc. Owing to its wide use, Globus has been integrated by other projects such as Condor-G, MPICH-G2 and NetSolve. Condor-G [27] is the combination of Condor and Globus projects. Security and resource access in multi-domain environments, management of computation and harnessing of resources as well as task brokerage are its distinguished features. Condor-G can be used as front-end to a computational Grid and provides job management and monitoring, logging, notification, among others. MPICH-G2 [21] and NetSolve [28] are other examples of integration with Globus using services from the Globus Toolkit.

gLite. gLite[§] is another middleware for Grid computing, which provides a framework for building Grid applications through an integrated set of components that support resource sharing. gLite is developed as part of the Enabling Grids for E-sciencE (EGEE) project. The gLite Grid services follow a Service-Oriented Architecture, aiming to facilitate compliance with Grid standards, such as Open Grid Service Architecture (OGSA). One distinguished feature of gLite is that users can implement services according to their particular needs without having to use the systems as a whole.

BOINC. The Berkeley Open Infrastructure for Network Computing (BOINC) is an open platform that tries to benefit distributed computing from infra-utilized computational resources (PCs that donate cycles). Although originally designed for volunteer computing, it is appropriate for Grid computing as well, especially for desktop Grids. It is based on client/server setting, where the server generates work units, that is, executable and data input files, send them to clients (PCs) and collects results from them. Owing to its volunteering nature, there are security issues and it is expected that volunteers will return correct computational results. In fact, BOINC can be seen as more appropriate for P2P systems and has become popular by Seti@Home projects.

SunGrid Engine. The Grid Engine [29] is a software project for distributed computing by Sun Microsystems and hosted by CollabNet. Its aim is to provide distributed resource management software for wide ranging requirements from compute farms to Grid computing. The Grid Engine software accepts jobs submitted by users and uses resource management policies to schedule them on to the Grid. This process is transparent to the user, namely, users are not concerned about where the jobs are allocated. The Grid Engine is mostly oriented to support *Campus Grids*, that is, Grids made up of computing resources of multiple projects or departments within an organization.

Concurrency and Coordination Runtime (*CCR*). CCR [30] is an asynchronous programming library based on *.Net* framework for robotics. In fact, it is not only limited to modelling robotic applications but can be also used as a general programming paradigm through efficient thread management. Recently, CCR has been investigated as a paradigm for integrating Grid computing and multicore systems [31].

Zenturio. Zenturio [32] is an middleware for experiment management of parallel and distributed applications on cluster and Grid architectures. It is based on Web services standards and the Globus toolkit. A directive-based language (in the Fortran HPF and OpenMP style) called ZEN is used to specify value ranges for arbitrary application parameters, including loop and array distributions, software libraries and algorithms, interconnection networks or execution machines.

Other Grid middleware and tools. Other Grid middleware and tools have been developed or are currently under development in the context of many research projects on Grid computing. Askalon [33], CometG [34] and PadicoTM [35] are other examples of middleware and environments for Grid application development.

[§] http://glite.web.cern.ch/glite/.

2.2. Programming languages and models for grid systems

Features of the conventional parallel programming models [36] are also desirable for Grid programming models. Yet, due to heterogeneous and dynamic nature of Grid systems, new programming tools should support specific characteristics of Grid applications and environments. As stated by Foster and Kesselman [1]: 'Grid environments will require a rethinking of existing programming models and, most likely, new thinking about novel models more suitable for specific characteristics of Grid applications and environments'. We briefly consider here recent development in languages and runtime environments for Grid computing. (The reader is referred to [37, 38], for surveys on Grid programming models.)

Programming, languages and runtime environments. One common observation is that up-to-date programming tools and languages are insufficient to support the effective development of Grid applications. In particular, the lack of a simple, *standard* programming interface that hides complexities of Grid systems makes difficult and tedious their programming.

Grid Execution Language (GEL) [39] is a scripting language for programming parallel applications for a Grid environment. The language is aimed to facilitate coping with high-latency communications and heterogeneity in Grid environments. GEL's semantics provides constructs for while loops, conditionals and explicitly parallel execution. In the *Grid-Occam* project [40], a parallel programming language to support programming in heterogeneous Grid environments is also developed. *GAP* [41] is a high-level programming toolkit for Grid-enabled applications developed by the GridLab project. It aims at bridging the gap between the existing Grid middleware and application-level needs through a unified simple programming interface to the Grid infrastructure. *Abacus* [42] is a service-oriented programming language for Grid applications. Abacus offers a service abstraction at language level and hides the user from low-level details of service deployment supported by the compiler and the runtime system. *Lightweight Java task-spaces framework* [43] is suited for applications that require inter-task communication. The framework is characterized by decentralization, direct communication between tasks through tuple space distributed over the worker hosts. *Assist* [44] is a prototype of parallel programming environment in which Grid applications are built as compositions of components. It is based on Web Services and the CORBA Component Model. *Grid/ML* [45] is a high-level ML-like language (for functional parallelism) developed as part of the ConCert Grid framework. The language is especially suited for handling/overcoming security concerns in the dissemination of Grid programs.

Parallel and distributed models for Grids. Parallel and distributed models employed for Grid applications are essentially those from traditional parallel and distributed computing such as Shared Memory Model, Threads/OpenMP, Message Passing Model, Object and Service Oriented Models as well as their hybridizations. Yet, due to the intrinsics characteristics of Grid infrastructures, those models need adaption in order to efficiently exploit the parallel nature of Grid systems. It should be noted that heterogenous Grid systems are most suited for embarrassingly parallel applications. In most cases, the explicit degree of parallelism to exploit in a particular application is upon the Grid programmer who will provide task decomposition for that application.

Classical parallel models, such as Master–Worker, Task Farming, Parametric Computation and Divide-and-Conquer, are commonly used in Grid applications. On the other hand, adaption of workflow models has been proposed for Grid applications having dependencies among their tasks in order to exploit coarse-grain, dataflow-like parallelism. Finally, some models are also proposed to address needs of specific classes of applications. For instance, the IBM's Compute Grid programming includes two programming models, namely, transactional batch and compute-intensive, within the WebSphere Software. In this case, a compute Grid application is a J2EE application that conforms to the aforementioned Grid programming models.

Task, data parallelism and their combination. Task and data parallelism are two traditional models for parallel applications. The former exploits functional parallelism features and is achieved by distributing processes on different nodes of the system. The later exploits parallelism by distributing data across the nodes of the system. With the advent of Grid computing, the exploitation of both modes of parallelism can enable the solution of multidisciplinary applications in which a large amount of data has to be accessed. Current Grid middleware do not fully benefit features of both parallel models; unified frameworks and languages that support both forms of parallelism are lacking in Grid computing, although there are some proposals [46, 47] in this direction. One additional challenge in the Grid context is the need to access data sources and repositories having different formats and structures requiring the integration of semantic datagrid features. Finally, the Grid middleware should also offload the query processing to databases (scheduling and resource allocation) to support parallelism.

3. P2P PARADIGM FOR PARALLEL PROCESSING

While Grid computing was originated in the scientific community and was as at the very beginning conceived as a way for harnessing more processing power for the development of large-scale distributed applications, P2P computing only recently has started to consider features of using P2P technologies for large-scale distributed systems. Early studies [13] addressed issues for P2P systems, given the important differences in sharing files vs sharing computation and computational resources. Special emphasis is put on the need for scalability of P2P applications that would require keeping knowledge of a small fraction of global state in each peer, the need for load-balancing, its separation from the P2P application, deploying internet services by overlaying, etc.

P2P technologies can contribute in several ways to large-scale parallel and distributed systems and applications; however, the most important one is collaborative and contributory computing. Harnessing the power of networked PCs, through CPU scavenging and volunteer computing [16], P2P systems are showing their great potential in solving complex problems. It should be noted however that in P2P distributed projects, the performance is usually 'best effort'-like as compared with high-performance Grid. Nonetheless, P2P systems each time more are able to leverage high rates of throughput (currently processing at rates of few Terabytes), which is significant processing power for data-intensive applications.

Despite recent advances, there are still many issues that prevent P2P from being widely used as paradigm for parallel processing, which on the other hand shed light on why Grid computing is to date more successful than P2P computing for parallel processing.

Scalability: Unlike current Grid systems, which tend to be moderate to medium size, P2P systems can actually be very large, virtually joining millions of peers. This is a great feature of P2P systems regarding the computing capacity but at the same time poses a big challenge: *how to make a huge P2P system reliable for distributing computing out of unreliable P2P clients*?

Standardization: Creating standard interfaces is a problem for both Grid and P2P systems. Still, Grid computing community is devoting significant efforts to create a standard Grid interface for the development of Grid-enabled applications that would offer authentication, resource discovery and allocation, efficient scheduling. Globus Toolkit and OGSA implementations are efforts toward this standardization. In the P2P side, up to date, there are no general-purpose P2P systems in use; rather, P2P systems have been developed to support specific services. Also, one can find many independent clients running on the same peer machine, with the peer belonging to different P2P networks (each one using different protocols). Moreover, in general there is no data sharing among different P2P applications.

Fault-tolerance: Grid systems can be considered as more problematic regarding fault-tolerance due to their centralized nature, whereas P2P systems are in principle more robust. However, in practice it is difficult to build robust P2P applications, especially, in the presence of task dependencies. Popular P2P applications, such as Seti@Home using BOINC are in fact client/server approaches. Decentralization and dynamism are thus not taken full advantages in the current P2P systems.

Efficiency: The efficiency of the P2P systems depends much on the contributed bandwidth and other peer resources (storage, processing power, etc.). Incentive mechanisms, such as using bandwidth as currency, are very important in this context to increase the efficiency of the system. Some P2P networks implement the '*give-to-get algorithm*' to solve shortcomings in P2P file-sharing systems. Thus, implementing give-to-get algorithms so that any peer is both a contributor and a beneficiary of the shared resources would increase the systems' efficiency also for the case of parallel applications.

Security: P2P systems have all security issues of volunteer computing. Peers might wish to hide personal identifying information; hence, maintaining anonymity is each time more important to contributing peers. Detecting malicious peers, such as those who can falsify and return incorrect results or distributing malicious executable programs to peers, etc. are important problems in P2P systems. Such problems are, for instance, found with the BOINC system, although it implements some mechanisms for preventing them.

These issues, although not altogether, are addressed in the P2P middleware, considered in the next section.

3.1. P2P middleware and programming models

As in the case of Grid computing, middleware and communication libraries are vital for the development of P2P-enabled applications. Compared with the Grid systems, there have been proposed fewer middleware and communication libraries for P2P systems. They are based on three architectures: client/server, pure P2P (fully decentralized) and hybrid (super-peer) architectures. Current research focus is on general-purpose P2P middleware to support P2P applications. We briefly review some of them next.

JXTA Library. JXTA [14] is a library[¶] consisting of a set of generalized P2P protocols. The set of protocols can be used as a basis for the implementation and deployment of P2P networks. JXTA protocols in fact aim to standardize the way peers discover other peers in the network, peer communication, organization of peers in peergroups, the announcement and discovery of services in the network, etc. Moreover, security services are offered in using JXTA protocols (confidentiality, authentication, data integrity, etc.).

A JXTA-based P2P network consists of a set of interconnected peers, which can be self-organized in groups and offer common services to the rest of the network. Services are announced using XML documents, the so-called, advertisements. Advertisement is an important piece in a JXTA network; through advertisements peers can join other peers, be aware and use available services in the network. Among others, there are peer advertisements for peergroup, pipes, rendezvous and peer information. Messages are XML documents encapsulating routing information, credential information (binary data can be also encoded and included) and are sent through pipes.

JXTA-based networks can have four types of peers: *minimal edge peer*, which can send and receive messages but cannot send or save in its local cache advertisements; *full edge peer*, which can send and receive messages and also save advertisements in its local cache but cannot handle advertisements for resource discovery; *rendezvous peer*, which besides the aforementioned properties is able to handle advertisements for resource discovery. It should be noted that each peergroup must have at least a rendezvous peer. When a peer wants to join a peergroup, it searches for a rendezvous peer. Moreover, rendezvous peers keep lists of other known rendezvous and peers connected to them. The rendezvous peer is also known as super-peer (*aka* ultra-peer) and is the kind of peer that plays a crucial role in a JXTA network. Finally, *relay peers* keep information about routing of other peers and routing of messages. In particular, relay peers help sending messages of peers to other peers they do not have access to.

Peers use several protocols for communication and accomplish their functionalities. Essentially there are six protocols in JXTA; it should be noted that a peer need not implement all these protocols but only those required for its specification. These protocols are: *peer discovery protocol*, which serves for announcing resources (peers, groups, pipes, services, etc.) and discovers other peers' resources; *peer information protocol*, which serves for getting state information (time, state, traffic, ...); *peer resolver protocol*, which serves for sending generic requests to one or more peers, usually for information exchange; *pipe binding protocol*, which serves for establishing virtual

[¶]Open source community project https://jxta.dev.java.net/.

communication channels (pipes); *endpoint routing protocol*, which serves for finding routing to other peers through relay peers; and, *rendezvous protocol*, which enables peers for subscribing to service propagation.

As can bee seen from the above description, JXTA features, such as stable API, NAT and firewall traversal, decentralized architecture and scalability, make it appropriate for developing higher-level middleware.

JXTA-based middleware. Several middleware based on JXTA have been reported in the literature. These include JXTA-Overlay, P3, JXTPIA, Jalapeno, JNGI, JXTA-Grid, OurGrid, Triana and Xeerkat. We briefly describe them next.

JXTA-Overlay[‖] [48, 49] project is an effort to use the JXTA technology for building an overlay on top of JXTA offering a set of basic primitives that are most commonly needed in P2P applications. It comprises primitives for peer discovery, peer resources discovery, resource allocation, task submission and execution, File/data sharing, discovery and transmission, instant communication, peer group functionalities (rooms, etc.) and monitoring of peers, groups, tasks, etc. This set of basic functionalities is intended to be as complete as possible regarding the needs of P2P applications, which can be built on top of the overlay. One of the characteristics of the primitives offered by the JXTA-Overlay is their genericity and independence from the applications that will be using them, yet allowing to keep the intrinsic decentralized nature of Grid/P2P systems. Regarding the architecture, JXTA-Overlay is based on a broker–client model, implemented as a broker and a client layer, respectively. Brokers are the governors of the JXTA P2P network as they are in charge of achieving efficient resource allocation functionalities, resource monitoring, management of executable tasks, etc. Note that broker peers extend rendezvous peer and do not interact with final user applications; therefore, they represent just one layer. The client layer is in charge of receiving and managing all events generated at the application level due to calls to the primitives. In this way, we achieve the set of primitives to be completely decoupled of any application as the client layer will allow (final) user applications to transparently communicate with the overlay. Several economic-like models have been implemented for selecting peers in job executions [48]. JXTA-Overlay has been tested through two different types of applications. The first is a distributed application that processes large log data files in regularly sequenced data (RSD) format. The second one is to create a group-ware environment for students of a virtual university. Both applications showed the feasibility of the overlay approach.

Personal Power Plant (*P3*) [50] is another middleware using JXTA library aimed at supporting common requirements of P2P software. P3 consists of a job management subsystem, a job monitor and parallel programming libraries. P3 creates a JXTA-supported peer group called a job group for each job that is submitted by a parallel application. The experimental evaluation of this library [50] showed an overhead in terms of communication performance although a cluster on Gigabit Ethernet LAN was used as infrastructure.

JXTPIA [51] is a JXTA-based P2P network interface and architecture comprising home-work distribution module, trigger distribution module and data sharing module. JXTPIA aims at supporting Grid computing through resources allocation, task scheduling and task assignments.

Jalapeno [52] is intended for supporting task execution in a desktop Grid. It consists of manager, worker and task submitter hosts. Jalapeno is appropriate as a framework for solving embarrassingly parallel applications in which the problem is split into smaller independent sub-problems. *JNGI* [53] is another JXTA-based framework for job executions using computational peers in the P2P network. Jobs are split and distributed among several peers. It is claimed that by providing redundancy within peer groups, it is ensured that failures do not affect job completion. *OurGrid* [54] addresses the use of an economic model for resource sharing in Grids. It uses JXTA protocols for basic P2P-functionalities such as peer discovery and message broadcasting. *Triana* [55] is an open-source problem-solving environment especially as a workflow manager for Grid applications. *Xeerkat*[**]

[‖]https://jxta-overlay.dev.java.net/.
[**]Xeerkat http://code.google.com/p/xeerkat/.

is a JXTA-based framework that uses a worker/hiring analogy to establish computational Grids. It uses an agent computing model where an agent runs a number of available services.

XtremWeb. XtremWeb [56] is a Java-based software for the development and deployment of light-weight Grids, namely Grids based on desktop PCs. It is mainly addressed for developing embarrassingly parallel applications (parametric model). The computing resources are provided on a volunteer-basis. XtremWeb architecture is made up of Client, Coordinator and Workers and uses a pull model in which Workers initiate communications to request jobs from Server. Worker's results are sent to the Coordinator (result collector). In a P2P scenario, XtremWeb can be used to build centralized P2P systems in which a Worker behaves as a Client. Jobs submitted by the Client are registered on the Server and scheduled on Workers. Communications between different parts of XtremWeb include remote procedure call messages and data transfers.

P2P-MPI. P2P-MPI [57] is a middleware for developing embarrassingly parallel applications by grouping computing resources of desktop Grids. Typically, a user can request P2P-MPI to transparently find a given number of processors for running a Java application, using the P2P-MPI message passing library, which in turn conforms to MPJ.

Other middleware include *Omnix* [58], *BestPeer* [59], *IRIS* [60] and *PeerBus* [61], whose descriptions are omitted here.

Shared data programming model. One of the most important features of P2P systems is data sharing among peers in the network. In fact, data sharing and data parallelism are not yet fully exploited in P2P systems. Mechanisms for data sharing and transfer, data publishing and replication, data discovery and access are common to e-Science applications. JUXMEM [62] is a JXTA-based interface to shared data programming on the Grid.

Efficient query-based programming model. Efficiently querying the network is a key issue in P2P programming. Despite recent advances, there are still many shortcomings in this regard. Teaq [63] addresses the efficient distribution of processes in the network according to peers' processing capabilities and efficient querying of programming-level objects in the network. Other XML-based queries, such as XQuery and XPath, are also being addressed in the P2P programming to efficiently query collections of XML data.

4. GRID AND P2P-ENABLED E-SCIENCE APPLICATIONS

Grid computing has shown its usefulness for a family of applications arising in science and engineering. The Grid infrastructure used for this purpose is identified as e-Science, defined as a '*set of advanced technologies for collaboration and sharing resources across the Internet, so-called Grid technologies, and technologies integrated with them*'. e-Science stands for a new research field that focuses on collaboration for solving complex problems from science and engineering using new computational paradigms and infrastructures. From an application perspective, e-Science embraces a large family of applications, although at the beginning Grid technology consisted mainly of purely scientific applications. Simulation-driven or experiment-driven applications, such as parametric modelling, parameter sweep applications, Monte-Carlo simulations, are examples of successful e-Science applications.

Presently, e-Science applications span domains such as chemistry and physics, medical and life sciences, engineering and design, mathematics, economics, business and finance, environmental science, earth sciences and astronomy, etc. Nowadays, the number of applications with e-Science keeps growing in new areas of science and engineering. In [64], the authors provide a classification of different approaches for e-Science Grid infrastructures in solving complex problems. The multi-disciplinary nature of e-Science applications as well as the cross-domain collaboration of researchers and Grid infra-structures and virtual laboratories make Grid technologies promising collaborative technologies.

P2P paradigm is increasingly contributing to e-Science. P2P-enabled e-Science applications became popular by the SETI@Home project of massively distributing computing for protein folding, detecting intelligent life outside earth, etc. P2P paradigm has also started to be considered in combinatorial optimization (e.g. for branch-and-bound algorithms [65, 66] and evolutionary algorithms [67]) and data mining [68].

5. INTEGRATED APPROACHES FOR GRID AND P2P SYSTEMS

As can be seen from the exposition in previous sections, Grid and P2P systems have followed different trajectories motivated by different needs and users' interest and supported by rather different developing and research communities. Fortunately, both paradigms are evolving in a way that each time they share more common characteristics. Among these characteristics, we distinguish the cooperative model for solving complex problems. Grid and P2P systems have common intrinsic features of large distributed systems and both of them are concerned with harnessing and sharing of computationally resources. In fact, they can benefit from each other characteristics. Grid systems can benefit from P2P techniques of resource discovery, decentralization, replication and scalability, which are extensively addressed in the P2P literature. In this sense, P2P systems can provide more flexibility to Grid approaches. For instance, resource and service discovery approach in Grid systems is centric or hierarchic-based, whereas P2P systems use advanced presence mechanisms of peers. By implementing P2P-like presence and discovery mechanisms, Grid systems will be able to use in a decentralized and efficient way smaller groups of resources within the whole system. On the other side, P2P systems can benefit from resource allocation, scheduling, load-balancing techniques, security techniques, etc. developed for Grid systems.

Currently there are no proposals for P2PGrid middleware that would support best features of both paradigms; however, there are attempts to develop such integrated approaches. OGSA model is one such example, in which Grid, P2P and Web Service concepts are merged. Also, efforts are being done at the infrastructure level, such as mixing clusters and desktop machines for P2P-based computational Grids (e.g. [69–71]) or the PlanetLab platform [72] as a testbed for Grid-P2P applications.

6. CASE STUDY: PARALLEL PROCESSING USING JXTA-OVERLAY MIDDLEWARE

In this section we present an application, developed on top of JXTA-Overlay middleware, for parallel processing of log data files of a virtual campus.

6.1. Log data files of virtual campuses

Virtual campuses, virtual organizations and emerging virtual institutions [2] are new ways of organizing community-based activities by using IT technologies. These new paradigms of organization are sustained by the common interest of the members of the community to achieve their goals. Thus, in the case of virtual campuses, which is one of the most widely used of virtual organizations in the today's teaching and learning activity, the members of the campus pursue academic goals. In such context, log data are important for the description and studying students' behavior and navigation patterns when interacting with the virtual campus.

As a real example, the Open University of Catalonia (http://www.uoc.edu) offers distance education through the Internet in different languages to about 40 000 students, lecturers and tutors. Students and other users (lecturers, tutors, administrative staff, etc.) continuously browse virtual individual or community areas. All users' requests are chiefly processed by a collection of Apache web servers as well as database servers and other secondary applications, all of which are providing service to the whole community. For load balance purposes, all HTTP traffic is smartly distributed among the different Apache web servers available and each web server stores in a log file each user request received and the information generated from processing it. Once a day, all web servers in a daily rotation merge their logs producing a single, very large log file containing the whole user interaction with the campus performed in the last 24 h. A daily log file size may be up to 10 GB.

Algorithm 1 Master processing: Phase I

```
 1: Read the configuration file.
 2: Compute the chunk size, nb_lines, according to the number of chunks specified in configu-
    ration file.
 3: repeat
 4:    Read nb_lines lines from the log data file and create a chunk with them.
 5:    if (transmission mode is FTP) then
 6:       Upload the chunk at FTP server
 7:    else
 8:       Store chunk at local cache.
 9:    end if
10: until (the log data file has been completely scanned)
11: for all (chunk files) do
12:    Create a job request;
13:    Submit job request to JXTA-Overlay platform.
14: end for
```

Algorithm 2 Master processing: Phase II

```
1: repeat
2:    Receive partial results from peers.
3:    Append, in the correct order, the result to the final file.
4:    Delete chunk files from FTP or from local storage.
5: until (there are messages from peers to receive)
```

This huge amount of information needs pre-processing to remove non-relevant information and then can be processed for extracting useful information.

6.2. *Parallel processing using JXTA-overlay*

The log data files of the virtual campus follow an RSD pattern. In such format, log data are textual, event or record oriented and the boundaries between events/records are easily identifiable. The RSD format of log data is an important feature for its parallel processing. Indeed, since the boundaries between events/records are easily identifiable, processing log data files falls into the family of embarrassingly parallel applications.

The parallel implementation is based on a Master–Worker paradigm. The log data file is split off into several parts (the number of such parts depending on how peer worker nodes will participate in the processing as well as the chunk size). The chunk files are sent to peers for processing. At the end, the master node collects partial solutions from peer nodes and composes the final result. The main steps followed by the master, brokers and peers for log file processing in the JXTA-Overlay platform are shown in Algs. 1, 2 and 3.

Both FTP and JXTA direct transfer were implemented for file transfer from the Master to worker peer nodes and were experimentally studied to identify the impact of data transmission on the overall performance.

6.3. *P2P infrastructure and experimental setup*

The P2P network using the JXTA-Overlay was deployed in the nodes of nozomi cluster (at `nozomi.lsi.upc.edu`) and nodes of the PlanetLab infrastructure. The former were used to deploy broker nodes, whereas the latter to deploy client peers without graphical user interface (called `SimpleClient` peers). Additionally, two PCs and a laptop of standard configurations were used to deploy client peers with graphical user interface (called `GUIClient` peers). In the experimental study, one of the GUIClient peers was used as *master* node, which runs the main flow of processing.

Algorithm 3 JXTA-Overlay processing

```
1: while (there are job requests) do
2:    Job request is received by a broker;
3:    The broker selects a worker peer according to peer selection models specified at configuration
      file;
4:    The broker allocates the request to the selected worker peer;
5:    The worker peer, upon receiving and accepting the request, notifies it to the broker;
6:    if (transmission mode is FTP) then
7:       The worker peer downloads the chunks from FTP server;
8:    else
9:       The worker peer requests chunk file to the broker;
10:   end if
11:   The worker peer sequentially processes the chunk file;
12:   The worker peer sends to the Master the result;
13: end while
```

PlanetLab nodes used as SimpleClients (abbreviated SCx below) comprised the following nodes:

- ait05.us.es (SC1),
- planetlab1.hiit.fi (SC2),
- planetlab01.cs.tcd.ie (SC3),
- planetlab1.csg.unizh.ch (SC4),
- edi.tkn.tu-berlin.de (SC5),
- lsirextpc01.epfl.ch (SC6),
- planetlab1.itwm.fhg.de (SC7),
- planetlab-1.ssvl.kth.se (SC8).

A log file of 100 MB (roughly 300 000 lines) was processed. Besides processing it as a whole, it was also split into 4 and 16 parts ('chunks') of the same size. Each experiment was run for five independent times to avoid biased results and average results are reported.

The evaluated scenarios. We give below the scenarios for evaluating the performance of the parallel processing at the JXTA-Overlay. They consisted of the following:

1. *Pre-processing*: the master node splits the file in as many parts as indicated in the configuration. If the transfer mode is the FTP, then the master uploads the chunks to the FTP server; if the transfer mode is JXTA direct transfer, then the master stores the chunks locally and sends them to the selected worker nodes afterwards upon request. The graphical representation of the pre-processing time is shown in Figure 1.
2. *Processing at broker peer site*: Here we measure the time needed by broker peer to process and allocate the job execution requests. For each request, the broker has to select among all available peers, the ones that will process the job execution request (in case there are more parts than peers, as it is the case of 16 chunks, some peers receive more than one part; in this latter case, the broker is in charge of achieving load-balancing among peers and re-submitting the job execution request to another peer in case of failure). The graphical representation of the processing at broker site is shown in Figure 2.
3. *Receiving the file or chunk files for processing*: In this step, the whole file or chunk files are received by the worker peers for processing either from an FTP server or by direct transfer from the master node. The time for this step thus depends on the chunk size as well as the transmission mode. The graphical representation of the file reception time (for the case of JXTA direct transfer) is shown in Figure 3.

   As can be seen from Figure 3, the time for transmitting the whole file is considerably larger than sending the file by chunks. For instance, when the file is split into 16 parts (each of about 6.25 MB) the direct transfer time takes in average 1.47 min.
4. *Processing the chunk file*: This step consists of running the sequential program at peers' sites for processing the chunk files. The processing time is graphically shown in Figure 4. In the

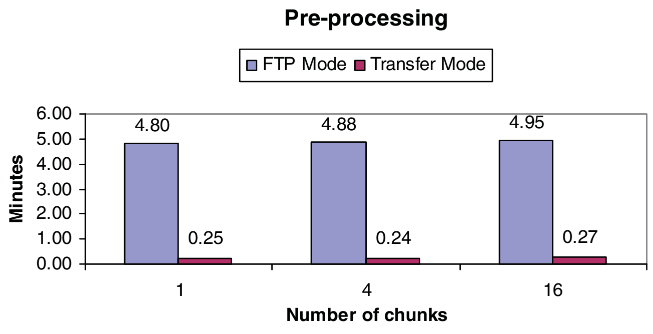

Figure 1. Master's pre-processing time.

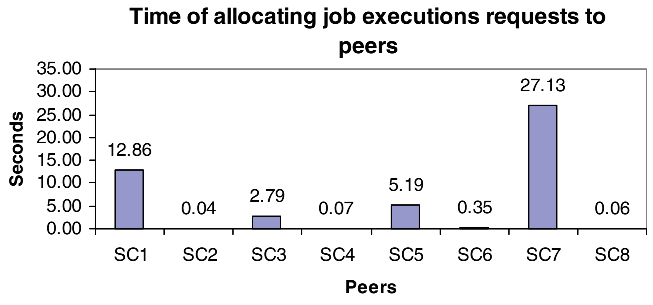

Figure 2. Broker's processing time for allocating job executions requests to peers.

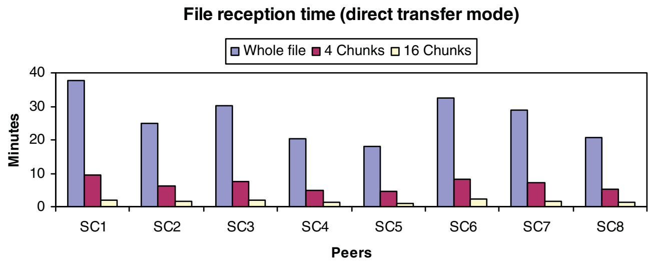

Figure 3. File transfer time from master node to peers.

figure, we also show the sole processing time vs downloading the file from the FTP server and processing.

We also show the processing time (see Figure 5) for just one chunk file of 25 MB (that would result from splitting the 100 MB into four chunks) and that of 6.25 MB (that would result from splitting the 100 MB into 16 chunks).

5. *Sending the result to Master*: Upon finishing, peers send the result of processing the chunk file to the Master. The time required for completing this step again depends on peers' networking connections. As can be seen from Figure 6, one of the peers (SC7) appeared to be as a bottleneck in communicating the results.

Finally, the total processing time is graphically shown in Figure 7 by using different peer selection models (see line 4 in Alg. 3). In the figure, the peer selection methods are abbreviated as follows: `EcFTP`: economic based selection model using FTP; `EcTr`: economic-based selection using JXTA transfer; `PriceFTP_s`: price-based model using FTP (peers having the same priority); `PriceTr_s`: price-based model using JXTA transfer (peers having the same priority); `PriceFTP_q`: price-based model using FTP (quickest peer); `PriceTr_q`: price-based model with quick peer

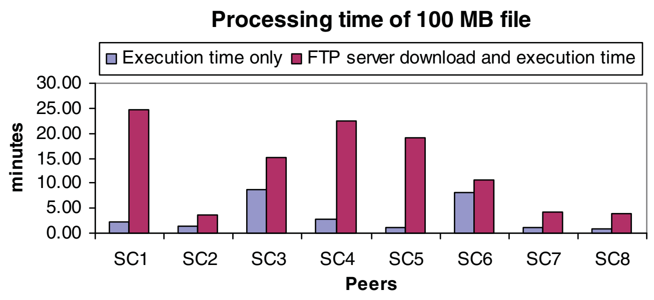

Figure 4. Execution time at peers site.

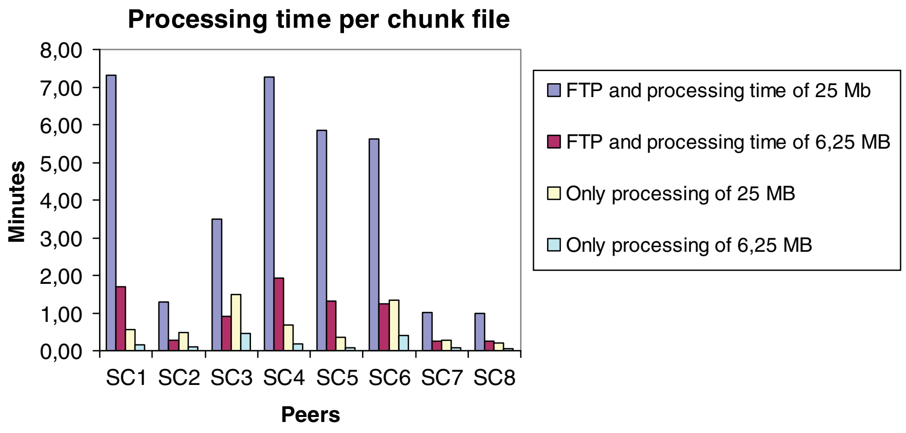

Figure 5. Processing time per chunk files at peers site.

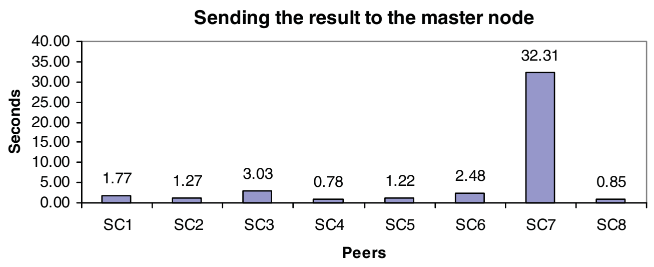

Figure 6. Time in sending the result to master node.

using JXTA transfer (quickest peer). (The reader is referred to [48] for details of peer selection models.)

As can be seen from Figure 7, a considerable reduction was achieved in the overall processing time by partitioning the original log data file into 16 chunks that are processed in the JXTA-Overlay platform. The use of different peer selection methods showed to be useful in reducing the processing time, especially when the JXTA direct transfer is used.

The experimental analysis also pointed out the need to achieve different degrees of granularity (chunk size) instead of using the same size for all chunks. Indeed, as can be seen from the experimental results (see Figures 2 and 6) the overall processing time is conditioned by the computational characteristics and networking connections of peer nodes.

(The reader is referred to [73] for a similar approach, but implemented using Grid services [74] and Globus Toolkit.)

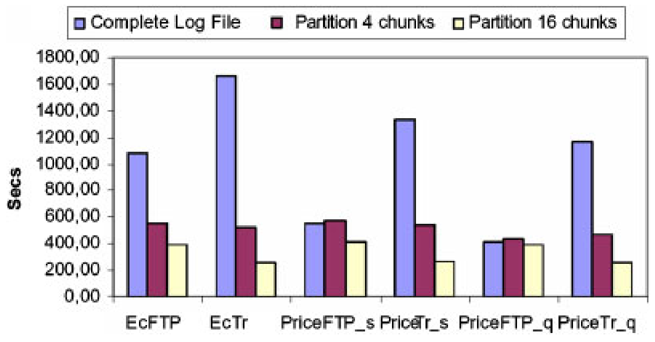

Figure 7. Total processing time of the 100 MB log data file in the JXTA-Overlay platform.

7. CONCLUSIONS

Grid computing originated in the scientific community in response to the need for harnessing more processing power for the development of large-scale distributed applications. P2P computing only recently has started to be used for large-scale distributed systems based on collaborative and contributory computing models. Harnessing the power of networked PCs, through CPU scavenging and volunteer computing, is enabling P2P systems to leverage high rates of throughput.

The analysis of the existing Grid middleware, programming tools and languages showed that they do not fully support the effective programming and development of Grid applications. In particular, the lack of simple, *standard* programming interfaces that hide complexities of Grid systems makes difficult and tedious the programming task in Grid systems. Nonetheless, Grid computing has shown to be more successful for parallel and distributing computing than P2P systems. Despite recent advances, there are still many issues, such as scalability, standardization, efficiency and security, that prevent P2P from being fully exploited and widely used as paradigm for parallel processing.

Although Grid and P2P systems have followed different trajectories and are motivated by different needs and users' interest and supported by rather different developing and research communities, both paradigms are evolving in a way that each time they share more common characteristics. Among these characteristics, we distinguish the cooperative model for solving complex problems by exploiting the large computing capacity by harnessing and sharing of computationally resources. As such, Grid and P2P systems can benefit from each other features to improve the overall system's performance.

Finally, there is a clear need for integrated frameworks for Grid and P2P, namely middleware that would support best features of both systems. Models, such as OGSA, which merge Grid, P2P and Web Service concepts, are showing their usefulness in this regard.

ACKNOWLEDGEMENTS

Fatos Xhafa's research work is supported by a grant from the General Secretariat of Universities of the Ministry of Education, Spain.

REFERENCES

1. Foster I, Kesselman C. *The Grid—Blueprint for a New Computing Infrastructure*. Morgan: Los Altos, CA, 1998.
2. Foster I, Kesselman C, Tuecke S. The anatomy of the grid. *International Journal of Supercomputer Applications* 2001; **15**(3):200–222.
3. Arnold D, Agrawal S, Blackford S, Dongarra J, Miller M, Seymour K, Sagi K, Shi Z, Vadhiyar S. Users' Guide to NetSolve V1.4.1. *Technical Report*, *ICL-UT-02-05*, University of Tennessee, 2002.
4. Available at: http://www.globus.org/ [August 2010].
5. Wright SJ. Solving optimization problems on computational grids. *Optima* 2001; **65**:8–13.
6. Linderoth L, Wright SJ. Decomposition algorithms for stochastic programming on a computational grid. *Computational Optimization and Applications* 2003; **24**:207–250.

7. Abramson D, Giddy J, Kotler L. High performance parametric modeling with Nimrod/G: killerapplication for the global grid? *Proceedings of 14th International Parallel and Distributed Processing Symposium* (*IPDPS 2000*). IEEE CS: Silver Spring, MD, 2000; 520–528.
8. Casanova H, Zagorodnov D, Berman F, Legrand A. Heuristics for scheduling parameter sweep applications in Grid environments. *Proceedings of the Ninth Heterogeneous Computing Workshop*. IEEE CS: Silver Spring, MD, 2000; 349.
9. Stockinger H, Pagni M, Cerutti L, Falquet L. Grid approach to embarrassingly parallel CPU-intensive bioinformatics problems. *Second IEEE International Conference on e-Science and Grid Computing* (*e-Science 2006*), Amsterdam, Netherlands, 4–6 December 2006; 58.
10. Zheng C, Katz MJ, Papadopoulos PM, Abramson D, Ayyub S, Enticott C, Garic S, Goscinski W, Arzberger P, Lee BS, Phatanapherom S, Sriprayoonsakul S, Uthayopas P, Tanaka Y, Tanimura Y, Tatebe O. Lessons learned through driving science applications in the PRAGMA grid. *International Journal of Web and Grid Services* 2007; **3**(3):287–312.
11. Chervenak A, Foster I, Kesselman C, Salisbury Ch, Tuecke S. The data grid: towards an architecture for the distributed management and analysis of large scientific data sets. *Journal of Network and Computer Applications* 2001; **23**:187–200.
12. Oram A. *Peer-to-Peer*: *Harnessing the Power of Disruptive Technologies*. O'Reilly, 2001. Available at: http://oreilly.com/.
13. Crowcroft J, Moreton T, Pratt I, Twigg A. Peer-to-peer technologies. *The Grid*: *Blueprint for a New Computing Infrastructure*, ch. 29, Foster I, Kesselman C (eds.). Morgan: Los Altos, CA, 2003; 593–622.
14. Oaks S, Traversat B, Gong L. *JXTA in a Nutshell*. O'Reilly, 2003. Available at: http://oreilly.com/.
15. Baker M, Carpenter B, Shafi A. MPJ Express: Towards thread safe Java HPC. The *IEEE International Conference on Cluster Computing*, Barcelona, Spain, 25–28 September 2006; 1–10.
16. Anderson DP, Fedak G. The computational and storage potential of volunteer computing. *Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid*. IEEE Computer Society: Silver Spring, MD, 2006; 73–80.
17. Tanaka K, Uehara M, Mori H. The performance evaluation of a grid using windows PCs. *International Journal of Web and Grid Services* 2008; **4**(4):395–417.
18. Bal H, Casanova H, Dongarra J, Matsuoka S. Application-level tools. *The Grid*: *Blueprint for a New Computing Infrastructure*, ch. 24, Foster I, Kesselman C (eds.). Morgan: Los Altos, CA, 2003; 463–489.
19. Kra D. Six levels of grid application enablement. IBM, 2004.
20. Berman F, Chien A, Cooper K, Dongarra J, Foster I, Gannon D, Johnsson L, Kennedy K, Kesselman C, Mellor-Crummey J, Reed D, Torczon L, Wolski R. The GrADS project: Software support for high-level grid application development. *International Journal of High Performance Computing Applications* 2001; **15**:327–344.
21. Karonis N, Toonen B, Foster I. MPICH-G2: A grid-enabled implementation of the message passing interface. *Journal of Parallel and Distributed Computing* 2003; **63**(5):551–563.
22. Brune MA, Fagg GE, Resch M. Message-passing environments for meta-computing. *Future Generation Computer Systems* 1999; **15**(5–6):699–712.
23. Kielmann Th, Bal HE, Gorlatch S, Verstoep K, Hofman R. Network performance-aware collective communication for clustered wide area systems. *Parallel Computing* 2001; **27**(11):1431–1456.
24. Fagg GE, Dongarra JJ. FT-MPI: Fault Tolerant MPI, supporting dynamic applications in a dynamic world. *Euro PVM/MPI User's Group Meeting* (*Lecture Notes in Computer Science*, vol. 1908). Springer: Berlin, 2000; 346–353.
25. Maassen J, Kielmann Th, Bal HE. GMI: Flexible and efficient group method invocation for parallel programming. *Sixth Workshop on Languages*, *Compilers*, *and Run-time Systems for Scalable Computers*, Washington, DC, U.S.A., 22–23 March 2002; 1–6.
26. Maassen J, Kielmann Th, Bal HE. Parallel application experience with replicated method invocation. *Concurrency and Computation*: *Practice and Experience* 2001; **13**(8–9):681–712.
27. Frey J, Tannenbaum T, Foster I, Livny M, Tuecke S. Condor-G: A computation management agent for multi-institutional grids. *Cluster Computing* 2002; **5**(3):237–246.
28. Brzezniak M, Makiela T, Meyer N. Integration of NetSolve with Globus-based grids. *Proceedings of Fifth IEEE/ACM International Workshop on Grid Computing*, Pittsburgh, U.S.A., 8 November 2004; 449–455.
29. Available at: http://gridengine.sunsource.net/ [August 2010].
30. Chrysanthakopoulos G, Singh S. An asynchronous messaging library for C#, synchronization and concurrency. *Object-oriented Languages* (*SCOOL*) *at OOPSLA 2005 Workshop*, San Diego, CA, 2005.
31. Qiu X, Fox G, Yuan H, Bae S, Chrysanthakopoulos G, Nielsen HF. High performance multi-paradigm messaging runtime integrating Grids and multicore systems. *Third IEEE International Conference on e-Science and Grid Computing*, Bangalore, India, 10–13 December 2007; 407–414.
32. Prodan R, Fahringer Th. ZENTURIO: A grid middleware-based tool for experiment management of parallel and distributed applications. *Journal of Parallel and Distributed Computing* 2004; **64**(6):693–707.
33. Fahringer Th, Jugravu A, Pllana S, Prodan R, Seragiotto C Jr, Truong HL. ASKALON: A tool set for cluster grid computing. *Concurrency and Computation*: *Practice and Experience* 2005; **17**:1–27.
34. Li Z, Parashar M. A computational infrastructure for grid-based asynchronous parallel applications. *Proceedings of the 16th International Symposium on High Performance Distributed Computing*, 2007; 229–230.

35. Denis A, Prez Ch, Priol Th. PadicoTM: An open integration framework for communication middleware and runtimes. *Future Generation Computer Systems* 2003; **19**(4):575–585.
36. Skillicorn D, Talia D. Models and languages for parallel computation. *ACM Computing Surveys* 1998; **30**(2):123–169.
37. Lee C, Matsuoka S, Talia D, Sussman A, Karonis N, Allen G, Saltz J. A Grid programming primer. *Global Grid Forum*, Advanced Programming Models Working Group, 2001.
38. Laforenza D. Grid programming: Some indications where we are headed. *Parallel Computing* 2002; **28**(12):1733–1752.
39. Ching Lian Ch, Tang F, Issac P, Krishnan A. GEL: Grid execution language. *Journal of Parallel and Distributed Computing* 2005; **65**(7):857–869.
40. Tröger P, von Löwis M, Polze A. The Grid-Occam project. *Grid Services Engineering and Management* (*Lecture Notes in Computer Science*, vol. 3726). Springer: Berlin, 2004; 151–164.
41. Allen G, Kelly D, Tom G, Andrei H, Hartmut K, Thilo K, M. Andr, Rob VN, Alexander R, Florian S, Thorsten S, Ed S, Brygg U. The grid application toolkit: Toward generic and easy application programming interfaces for the grid. *Proceedings of the IEEE* 2005; **93**(3):534–550.
42. Wang X, Xiao L, Li W, Yu H, Xu Zh. Abacus: A service-oriented programming language for grid applications. *IEEE International Conference on Services Computing*, Orlando, FL, U.S.A., 11–15 July 2005; 225–232.
43. De Sterck H, Markel RS, Phol T, Rde U. A lightweight Java taskspaces framework for scientific computing on computational grids. *Proceedings of the 2003 ACM Symposium on Applied Computing*, Melbourne, FL, U.S.A., 9–12 March 2003; 1024–1030.
44. Aldinucci M, Campa S, Danelutto M, Laforenza D, Scarponi L, Vanneschi M, Zoccolo C. Components for high performance Grid programming in Grid.it. *Proceedings of the Workshop on Component Models and Systems for Grid Applications* (*CoreGRID Series*). Springer: Berlin, 2005; 19–38.
45. Murphy T. ML grid programming with ConCert. *Proceedings of the 2006 Workshop on ML*, Portland, OR, U.S.A., 16 September 2006; 2–11.
46. Ben Hassen S, Bal HE, Jacobs CJ. A task- and data-parallel programming language based on shared objects. *ACM Transactions on Programming Languages and Systems* 1998; **20**(6):1131–1170.
47. Tudor D, Cretu V. Experiences on Grid shared data programming. *Proceedings of International Conference on Complex*, *Intelligent and Software Intensive Systems* (*CISIS 2008*), Barcelona, Spain, 4–7 March 2008; 387–393.
48. Xhafa F, Barolli L, Daradoumis Th, Fernandez R, Caballé S. JXTA-Overlay: An interface for efficient peer selection in P2P JXTA-based systems. *Computer Standards and Interfaces* 2009; **31**(5):886–893.
49. Xhafa F, Barolli L, Daradoumis T, Fernandez R, Caball S. Extension and evaluation of JXTA protocols for supporting reliable P2P distributed computing. *International Journal of Web Information Systems* 2008; **4**(1):121–135.
50. Shudo K, Tanaka Y, Sekiguchi S. P3: P2P-based middleware enabling transfer and aggregation of computational resources. *Fifth IEEE International Symposium on Cluster Computing and the Grid* (*CCGrid'05)*, Cardiff, U.K., 9–12 May 2005; 259–266.
51. Saitoh Y, Sumitomo K, Izaiku T, Oono T, Yagyu K, Wang H, Guo M. JXTPIA: A JXTABased P2P network interface and architecture for Grid computing. *Proceedings of 2005 International Conference on High Performance Computing and Networks* (*HPCC 2005*) (*Lecture Notes in Computer Science*, vol. 3726). Springer: Berlin, 2005; 409–418.
52. Therning N, Bengtsson L. Jalapeno: Decentralized grid computing using peer-to-peer technology. *Proceedings of the Second Conference on Computing Frontiers*, Ischia, Italy, 4–6 May 2005; 59–65.
53. Verbeke J, Nadgir N, Ruetsch G, Sharapov I. *Framework for Peer-to-peer Distributed Computing in a Heterogeneous*, *Decentralized Environment* (*Lecture Notes in Computer Science*, vol. 2536). Springer: Berlin, 2002; 1–12.
54. Andrade N, Cirne W, Brasileiro FV, Roisenberg P. *Ourgrid*: *An Approach to Easily Assemble Grids with Equitable Resource Sharing*, (*Lecture Notes in Computer Science*, vol. 2862), Feitelson DG, Rudolph L, Schwiegelshohn U (eds.), JSSPP. Springer: Berlin, 2003; 61–86.
55. Taylor I, Shields M, Wang I. Resource management for the Triana peer-to-peer services. *Grid Resource Management*: *State of the Art and Future Trends*, Nabrzyski J, Schopf JM, Weglarz J (eds.). Kluwer: Dordrecht, 2004; 451–462.
56. Cappello F, Djilali S, Fedak G, Herault T, Magniette F, Neri V, Lodygensky O. Computing on large scale distributed systems: XtremWeb architecture, programming models, security, tests and convergence with grid. *Future Generation Computer Systems* 2005; **21**(3):417–437.
57. Genaud S, Rattanapoka Ch. P2P-MPI: A peer-to-peer framework for robust execution of message passing parallel programs on grids. *Journal of Grid Computing* 2007; **5**(1):27–42.
58. Kurmanowytsch R, Kirda E, Kerer C, Dustdar S. OMNIX: A topology-independent P2P middleware. *CAiSE Workshops*, *Information Systems for a Connected Society*, Klagenfurt/Velden, Austria, vol. 75, 16–20 June 2003; 47–57.
59. Ng WS, Ooi BC, Tan KL. BestPeer: A self-configurable peer-to-peer system. *Proceedings of the 18th International Conference on Data Engineering*, IEEE Computer Society: San Jose, CA, 26 February–1 March 2002; 272.
60. Dabek F, Zhao B, Druschel P, Kubiatowicz J, Stoica I. Towards a common API for structured peer-to-peer overlays. *Proceedings of IPTPS'03* (*Lecture Notes in Computer Science*, vol. 2735). Springer: Berlin, 2003; 33–44.

61. Li M, Sun X-H, Deng Q, Ni J (eds.). *Grid and Cooperative Computing*, *Second International Workshop*, *GCC 2003*, *part I*, Shanghai, China (*Lecture Notes in Computer Science*, vol. 3032). Springer, 2004; 277–284.
62. Antoniu G, Boug L, Jan M. JuxMem: An adaptive supportive platform for data sharing on the grid. *Scalable Computing*: *Practice and Experience* 2005; **6**(3):45–55.
63. Evans H, Dickman P. Peer-to-peer programming with Teaq. *Web Engineering and Peer-to-peer Computing* (*Lecture Notes in Computer Science*, vol. 2376). Springer: Berlin, 2008; 289–294.
64. Riedel M, Streit A, Wolf F, Lippert Th, Kranzlmller D. Classification of different approaches for e-Science applications in next generation computing infrastructures. *IEEE Fourth International Conference on e-Science*, Indianapolis, IN, U.S.A., 7–12 December 2008; 198–205.
65. Talbi E-G, Bendjoudi A, Melab N. A parallel peer to peer branch and bound algorithm for computational Grids. *Proceedings of the International Workshop on Peer to Peer*, *Parallel*, *Grid and Internet Computing* (*3PGIC-2007*), Vienna, Austria, 10–12 April 2007; 271–278.
66. Tagashira S, Mito M, Fujita S. Towards generic solver of combinatorial optimization problems with autonomous agents in P2P networks. *Transactions on Information and Systems E89-D* 2005; **6**:1940–1947.
67. Laredo JL, Eiben AE, Steen M, Castillo PA, Mora AM, Merelo JJ. P2P Evolutionary algorithms: A suitable approach for tackling large Instances in hard optimization problems. *Proceedings of the 14th Euro-Par Conference on Parallel Processing* (*Lecture Notes in Computer Science*, vol. 5168). Springer: Berlin, 2008; 622–631.
68. Guan H. A study of parallel data mining in a peer-to-peer network. *Concurrent Engineering* 2007; **15**(3):281–289.
69. Fox G, Pallickara Sh, Rao X. Towards enabling peer to peer grids. *Journal of Concurrency and Computation*: *Practice and Experience* 2008; **17**(7–8):1109–1131.
70. Reddy MV, Srinivas AV, Gopinath T, Janakiram D. Vishwa: A reconfigurable P2P middleware for Grid computations. *Thirty-fifth International Conference on Parallel Processing* (*ICPP-2006*), Columbus, OH, U.S.A., 14–18 August 2006; 381–390.
71. Chazapis A, Zissimos A. A peer-to-peer replica management service for high-throughput Grids. *Proceedings of the 2005 International Conference on Parallel Processing*. IEEE CS: Silver Spring, MD, 2005; 443–451.
72. Planet lab. Available at: http://www.planet-lab.org/ [August 2010].
73. Xhafa F, Paniagua C, Barolli L, Caballé S. A parallel grid-based implementation for real time processing of event log data in collaborative applications. *International Journal of Web and Grid Services* 2010; **6**(2):124–140.
74. Comito C, Talia D, Trunfio P. Grid services: Principles, implementations and use. *International Journal of Web and Grid Services* 2005; **1**(1):48–68.