Journal of Grid Computing manuscript No.
(will be inserted by the editor)

The ShareGrid Peer-to-Peer Desktop Grid: Infrastructure, Applications, and Performance Evaluation

Cosimo Anglano · **Massimo Canonico** · **Marco Guazzone**

the date of receipt and acceptance should be inserted later

Abstract *Peer-to-Peer* (*P2P*) *Desktop Grids* are computing infrastructures that aggregate a set of desktop-class machines in which all the participating entities have the same roles, responsibilities, and rights. In this paper, we present *ShareGrid*, a P2P Desktop Grid infrastructure based on the *OurGrid* middleware, that federates the resources provided by a set of small research laboratories to easily share and use their computing resources. We discuss the techniques and tools we employed to ensure scalability, efficiency, and usability, and describe the various applications used on it. We also demonstrate the ability of ShareGrid of providing good performance and scalability by reporting the results of experimental evaluations carried out by running various applications with different resource requirements. Our experience with ShareGrid indicates that P2P Desktop Grids can represent an effective answer to the computing needs of small research laboratories, as long as they provide both ease of management and use, and good scalability and performance.

Keywords Desktop Grids · Volunteer computing · Bag-of-Tasks applications · Peer-to-Peer systems

1 Introduction

In many scientific areas, the use of computers to carry out research has become essential. The technological advances in scientific instrumentation have indeed led to the generation of increasing amounts of data whose processing requires larger and larger amounts of storage and computing power. The availability of computing infrastructures able to provide such amounts of resources has thus become fundamental for the achievement of scientific outcomes in these research areas.

Grid computing technologies [1] and infrastructures [2–5] have been shown to be able to provide very large amounts of storage and computing power. Typical Grid platforms (also referred to as *service Grids*) usually include relatively few and very powerful resources. Therefore, small research laboratories that cannot afford neither such powerful resources, nor the manpower costs required for their management, cannot exploit Grid computing to satisfy their computing needs.

Desktop Grids [6] (also referred to as *opportunistic Grids*), computing infrastructures that aggregate a (potentially very large) set of desktop-class machines owned by many independent individuals, have been used in the past both in the enterprise [7] and in the academia [8–11] as a low-cost alternative to traditional Grids. In academic settings a specific subclass of Desktop Grids, known as *Volunteer Computing (VC) systems* [12], has been typically adopted. VC systems exploit the unused capacity of non-dedicated computers voluntarily provided by a set of independent owners. Generally speaking, such systems use a *master server* that coordinates

This work has been partially supported by TOP-IX and the Piemonte Region Agency under the Innovation Development Program.

DOI: 10.1007/s10723-010-9162-z
The original publication is available at http://www.springerlink.com.

Cosimo Anglano · Massimo Canonico · Marco Guazzone
Dipartimento di Informatica, Università del Piemonte Orientale,
Viale Teresa Michel 11, 15121 Alessandria (Italy)

Cosimo Anglano
Tel.: +39 0131 360188
Fax: +39 0131 360198
E-mail: cosimo.anglano@unipmn.it

Massimo Canonico
E-mail: massimo.canonico@unipmn.it

Marco Guazzone
E-mail: marco.guazzone@unipmn.it

a set of *clients* by assigning them some work to do, collects the corresponding results and ultimately assembles them. To obtain significant benefits, however, VC systems must grow large enough to provide sufficient computing power and redundancy to tolerate the heterogeneity and failures of the involved computers. Furthermore, as discussed in [13], non-negligible efforts may be required to set up and maintain the master server. Consequently, small research laboratories are in general unable to exploit VC solutions, because of their lack of sufficient visibility to aggregate a large community of volunteers, and of enough manpower to set up and operate a master server. This is confirmed by the fact that the major part of successful Volunteer Computing projects is usually carried out in prestigious and famous institutions that typically have more resources to invest in project advertisement and infrastructure management.

Peer-to-Peer (P2P) Desktop Grids [14] have been recently developed to address the above issues; they are based on the principle that all the participating entities have the same roles, responsibilities, and rights. More specifically, participants to a P2P Desktop Grid share and use a set of resources on a reciprocity basis. That is, each participant lets other participants to use his/her resources when (s)he does not need them, provided that they do the same. In this way, small institutions that do not have enough computing power to satisfy their own needs, can federate with other similar institutions to obtain a possibly large Desktop Grid that can be easily shared by all participants. In contrast, in VC systems there is a clear difference in roles and rights between the resource donors (i.e., the clients) and the resource consumer (i.e., the master server); indeed, the former ones can only run the application(s) provided by the master server, while the latter is the only entity entitled to exploit the resources provided by the clients.

Although P2P Desktop Grids are considered very promising, converting this promise into reality is nontrivial, as it requires to properly address various issues affecting the performance, scalability, and usability of the resulting infrastructure. In this paper we report our experience in addressing these issues, that comes from the work we did during the design, implementation, and use of *ShareGrid* [15,16], a P2P Desktop Grid infrastructure that federates the resources provided by a set of small research laboratories, located in the Piemonte and Liguria areas (in Northwestern Italy). At the moment of this writing, ShareGrid federates 11 university laboratories, 1 public research center, and 1 private institution, providing more than 250 computing resources. ShareGrid is based on the *OurGrid* [14] middleware, that provides a set of core services enabling the aggregation and the sharing of large sets of computing resources, and on several additional components that we developed to provide a set of supplementary services. OurGrid (and, hence, ShareGrid) supports only *Bag-of-Tasks* (*BoT*) [17] applications (also referred to as *embarrassingly parallel applications*), that are composed by a set of identical and independent tasks that can be executed in parallel on distinct input data and parameters. It is worth to point out that this restriction is only an apparent limitation. As a matter of fact, this class of applications, despite its simplicity, is used in a variety of domains, such as *parameter sweeping* [18,19], simulations, fractal calculations, computational biology [20], and computer imaging [21]. Furthermore, as will be discussed later, all the applications of interest to the current ShareGrid partners fit within this class.

The primary purpose of this paper is to provide a complete description of ShareGrid, highlighting the techniques, tools, and algorithms that we used to ensure scalability, efficiency, and usability. The ability of ShareGrid of fulfilling its performance and scalability requirements is demonstrated by the results of a thorough performance evaluation activity that we carried out by running various applications characterized by different resource requirements.

Our experience in deploying, managing, and using ShareGrid (that, at the moment of this writing, spans over a three years period), indicates that P2P Desktop Grids can represent an effective answer to the computing needs of small research laboratories, as long as they provide ease of management and use, as well as good scalability and performance. We believe that our findings apply not only to ShareGrid, but also to the whole class of P2P Desktop Grids. Our experience and study, therefore, can be considered an assessment of the pros and the cons of these systems, and can be potentially useful to interested user communities to evaluate whether P2P Desktop Grids are an appropriate solution for their computing needs or not.

The rest of this paper is organized as follows. Section 2 describes the architecture and implementation of ShareGrid, as well as its current status. Section 3 describes the set of applications that currently use ShareGrid. Section 4 reports the results of the performance evaluation study we performed. Section 5 discusses the related works and, finally, Section 6 concludes the paper and outlines future research work.

2 ShareGrid: Design, Architecture, and Implementation

As for any Grid infrastructure, the main goals of ShareGrid are the achievement of satisfactory application performance and the ability to aggregate as many resources as possible without incurring into scalability problems. Additional goals include also the provision of high usability and ease of maintenance of the Desktop Grid infrastructure, since ShareGrid is targeted to user communities that do not necessarily include computer scientists. These goals were translated into

the following set of requirements, that drove its design, configuration, and deployment:

– **Performance and scalability requirements**:
  1. *Lack of centralized components*: in order to avoid possible performance bottlenecks, the architecture of the system must not encompass centralized components (or, at least, must keep them to a bare minimum).
  2. *Adoption of specific scheduling policies*: Desktop Grids are characterized by an extreme resource volatility (since any user can reclaim his/her own resources at any time and without any advance notice) and heterogeneity. In order to ensure satisfactory application performance, scheduling policies able to tolerate the high degree of resource failures and heterogeneity, that typically characterize Desktop Grids, must be adopted.
– **Usability requirements**:
  1. *Ease of use*: executing and monitoring applications, as well as collecting their results, must be as simple as possible and must not require any specific knowledge of the system architecture or configuration.
  2. *Support of multiple platforms*: in order to accommodate the largest possible number of distinct computing platforms, the middleware must support as many combinations as possible of operating systems and processor architectures.
  3. *Application agnosticism*: each user must be able to submit and run his/her own applications without having to set-up the system in an application-specific way prior to the submission.
– **Manageability requirements**:
  1. *Ease of joining and leaving*: the operation of adding or removing a set of resources must be as simple as possible and must not require specialized knowledge of the middleware and/or of the operating systems running on them.
  2. *Ease of configuration and maintenance*: the infrastructure must require minimal efforts (possibly none) to (re)configure and maintain its core components.
  3. *Resilience to resource departure*: the infrastructure must employ specific mechanisms and policies (e.g., task replication) enabling application to tolerate unanticipated resource departures.

To satisfy all the above requirements, the following activities have been performed: (a) choice of a suitable middleware platform providing the necessary mechanisms, (b) deployment of the middleware onto an appropriate system architecture, and (c) configuration of the middleware in order to tailor the various mechanisms (e.g., scheduling, data management, security, and accounting) to the specific needs of the user communities targeted by ShareGrid.

In the rest of this section we will firstly introduce OurGrid and motivate the reasons of our choice (Section 2.1). Then, we will describe the architecture of ShareGrid (Section 2.2). Finally, we will provide an overview of the characteristics of the resources currently belonging to ShareGrid (Section 2.3).

2.1 The choice of OurGrid

As anticipated in the Introduction, ShareGrid is based on the OurGrid middleware platform (and, in particular, on version 3 that was the stable release when the project begun), whose characteristics will be discussed in this section by referring to the architecture of a generic OurGrid-based infrastructure schematically depicted in Fig. 1.

In an OurGrid-based Desktop Grid, a set of *peer nodes*, each one in charge of managing a set of *working machines*, interacts to fulfill execution requests coming from a population of users. These interactions are carried out by means of a set of middleware components provided by OurGrid, namely the *Peer Agent* (running on each peer node), the *User Agent* (executing on each working machine), and the *MyGrid user interface* (running on individual user machines and providing user access to the computing infrastructure). A *core peer* (a machine running the *Core Peer Agent* middleware component) provides a directory service to all the peers in the Desktop Grid. Specifically, each peer, upon start-up, registers itself with the core peer in order to announce to the other peers its presence and the characteristics of the working machines it manages. Subsequently, the peer can query the core peer at anytime to obtain the list of all the currently

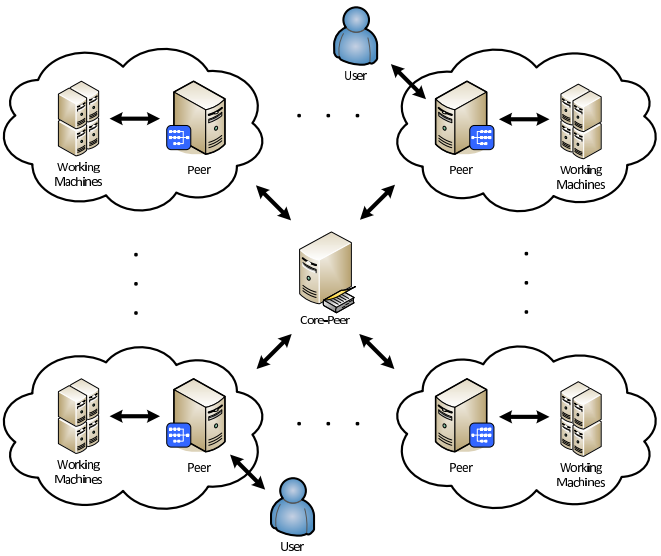


Fig. 1: Architecture of OurGrid.

available peers. In order for a user to submit his/her jobs, (s)he must associate with a specific peer (henceforth referred to as the *local peer*) by means of MyGrid, that will send all the submitted tasks to that peer. This association, however, is not fixed, and the user may change it at anytime as long as (s)he is not waiting for the completion of tasks already submitted to the current local peer.

OurGrid adopts a decentralized scheduling approach, in which each MyGrid manages the jobs submitted by the corresponding user independently from the other MyGrid instances in the system, and cooperates with its local peer to schedule them. Each MyGrid instance uses the *Work Queue with Replication (WQR)* scheduling algorithm [22]. WQR works by submitting, for each task, R distinct replicas (R is called the *replication factor*), that are started concurrently on different machines. When the first replica terminates its execution, the other $R-1$ are killed. Conversely, when a task replica abnormally terminates its execution (typically when the machine executing it is reclaimed by the respective owner), the corresponding MyGrid component automatically resubmits it; each task is resubmitted at most for a given number RS of times (RS is called the *resubmission factor*) before it is declared as failed by the MyGrid component [1]. Task replication provides both resilience to resource failures, and toleration of resource heterogeneity (a poor machine choice for a given task may be compensated by a better choice for one of its replicas).

When a user submits a job (consisting in a bag of N independent application tasks), its corresponding MyGrid instance requests to the local peer $R \times N$ available resources. A resource is considered to be available when (a) it is idle, and (b) it matches the requirements of the application (a boolean expression of one or more software and hardware constraints, specified by the user).

Then, the MyGrid instance and its local peer enter into an event-based loop, where in each step the peer locates a given number X of (either local or remote) resources that are available to execute these tasks, requests X tasks to the MyGrid instance, and dispatches them on these resources. The loop terminates when all the N tasks have been completed.

The local peer always gives preference to local tasks: each time a resource is needed to accommodate the request coming from a local user, and there are no free local resources, the peer first attempts to free busy resources by killing remote tasks (if any). If, even after remote tasks are killed, the numbers of local resources is lower than the number of task replicas pending at the MyGrid instance, the local peer contacts other peers in the system (in a random order), and requests them as many resources as possible.

[1] Note that both R and RS are set by each MyGrid instance independently from the other ones. Therefore, at the same time, different users may choose different values for R and RS when submitting their applications.

To ensure that *free riders* (users that exploit the resources of other participants without contributing their own ones to the system) receive little or no service, OurGrid uses the *network of favors* mechanism [23], a reputation-based resource allocation mechanism designed to promote contributions of resources among peers. In practice, each peer P_i locally maintains, for each other peer P_j in the system, the amount of *favors* it exchanged, a variable that accounts for the amount of time that P_i and P_j devoted to execute tasks of each other (suitably weighted to take into account the computational power of the involved machines, as discussed in [24]). When responding to requests coming from remote peers, P_i proportionally distributes its resources among the various peers according to their favor values; in case of tie, P_i chooses recipients at random. As soon as a donated resource returns to P_i, the appropriated local favor values of P_i and of each requesting peer, for which P_i has donated a resource, are updated.

From the above discussion, it should be clear OurGrid is able to fulfill all the requirements listed at the beginning of this section. More specifically, in OurGrid there are no centralized components (with the exception of the core peer, whose failure does not hinder the ability of the system to complete the already-submitted tasks). Furthermore, it provides scheduling algorithms able to deal with resource failures and heterogeneity. Joining an existing infrastructure requires only to deploy a User Agent on each working machine, and a peer to interface these machines with the rest of the system. Leaving the infrastructure is as simple as shutting down the peer. The departure of a peer does not affect in any way the other peers in the Desktop Grid, and task failures are transparently handled by means of replication and automatic resubmission mechanisms (described in subsequent sections). Consequently, the system is highly usable also by people with little or no background in computer science. Furthermore, OurGrid provides a set of decentralized user and resource management mechanisms that eliminate the need and the costs of centralized management. Additionally, it provides mechanisms and policies aimed at motivating users to donate resources while, at the same time, at discouraging free riders (something that is crucial for the sustainability of an infrastructure formed by peers that do not necessarily trust each other). Finally, all Unix variants, as well as most modern Windows versions, are supported on both 32 and 64 bit architectures.

The above considerations, and the observation that (as discussed in Section 5.1) none of the other middleware platforms available in the literature at the moment of our analysis was able to satisfy all the requirements listed above, motivate our decision of adopting OurGrid as the middleware substrate of ShareGrid.

2.2 ShareGrid Architecture

The ShareGrid infrastructure has been obtained by deploying the various components of OurGrid on a set of physical resources, by suitably using the mechanisms it provides to enforce specific resource management and security policies, and by adding a set of additional components that we specifically designed and developed in order to provide some functionalities that are missing in OurGrid.

More specifically, as indicated by Fig. 2, in addition to the standard OurGrid components, ShareGrid encompasses also three new servers: a *VPN Server* (providing peer visibility to firewall-shielded peers, see later), a *Portal Server* (providing Web-based user access to the infrastructure), and a *Storage Server* (providing users with a persistent storage facility for the input/output files of their applications). As discussed in the rest of this section, each of these components has been added in order to address a specific issue arising from the peculiar nature of ShareGrid, that consists in a federation of independent, heterogeneous, and volatile resources that do not trust each other.

2.2.1 Dealing with firewalls: the VPN Server

Nowadays, the presence of firewalls to filter the ingress and egress network traffic is practically ubiquitous. This means that, in order to make a given peer able to communicate with all the other peers of ShareGrid, it is necessary to properly configure the firewall controlling the traffic of the corresponding network so that it does not block the traffic generated by, and directed to, the peer. In some cases, however, such a configuration may not be possible. For instance, this was the case of some of the laboratories currently involved in ShareGrid whose network is part of a larger networking infrastructure which is globally shielded by a firewall controlled by another IT department.

The solution we devised for this problem consists in hosting each of these peers on a machine placed outside the corresponding network, so that the traffic directed to the peer is not blocked by the firewall, while the corresponding working machines are kept behind the firewall. In order to enable the peer to communicate with its working machines, all peer-to-working-machines communications are conveyed through the *Virtual Private Network (VPN) Server*, which is used as an application-level gateway. More specifically, each working machine opens a VPN connection with the VPN server (usually, the default configuration of firewalls enables the creation of outbound VPN connections), which is used by them as the gateway for all the outbound and inbound traffic. Analogously, the peer routes all the traffic directed to its working machines to the VPN server (as happens for peers A and B in Fig. 2), that forwards it to the appropriate destinations.

2.2.2 User interaction mechanisms: the Portal and the Storage Servers

MyGrid, the user interface provided by OurGrid, is characterized by some peculiarities that, in our opinion, make it awkward to use. As a matter of fact, OurGrid requires that submitting clients remain connected to their local peer for all the duration of the execution of the submitted tasks, for the following reasons:

1. pending tasks are queued locally at the MyGrid instance used to submit them. Therefore, if a MyGrid instance disconnects from its local peer before the job terminates, all the pending tasks are not scheduled;
2. the output files generated by completed tasks are temporarily stored on the peer managing the resources on which they ran, and must be retrieved by the submitting MyGrid instance by using the TCP connection it opened at the dispatching time. A disconnection of the MyGrid instance implies that these TCP connections are teared down, with the consequence that the results generated by completed tasks cannot be retrieved anymore.

Therefore, a crash or a disconnection of the submitting machine is a nefarious event that must be avoided as much as possible.

Another factor that makes MyGrid awkward to use is that it supports only Linux-based machines, hence users cannot use a different operating system on their submitting machines.

The *Portal Server* and the *Storage Server* have been introduced to overcome the above problems. The Portal Server hosts the *ShareGrid Web Portal* [25], that we developed in order to enable users to access ShareGrid by means of a standard Web browser. The ShareGrid Portal accepts submission requests by users, stores task input and output files on the Storage Server, and creates persistent connections with its reference peer through which task input and output files are transmitted. When one of the tasks submitted terminates (either normally or abnormally), the ShareGrid Portal sends an email notification message to the corresponding user, so that (s)he will not have to manually and periodically poll the system to monitor the status of his/her tasks.

By using the ShareGrid Portal, the user does not have to use the standard MyGrid user interface, and consequently can avoid its intrinsic limitations mentioned above. Furthermore, the portal provides additional facilities, such as the support for the automatic creation of submission files for parameter sweep applications (applications structured as a group of multiple independent runs executing the same experiment on a different set of input parameters). The interested reader may refer to [25] for more information on the portal.

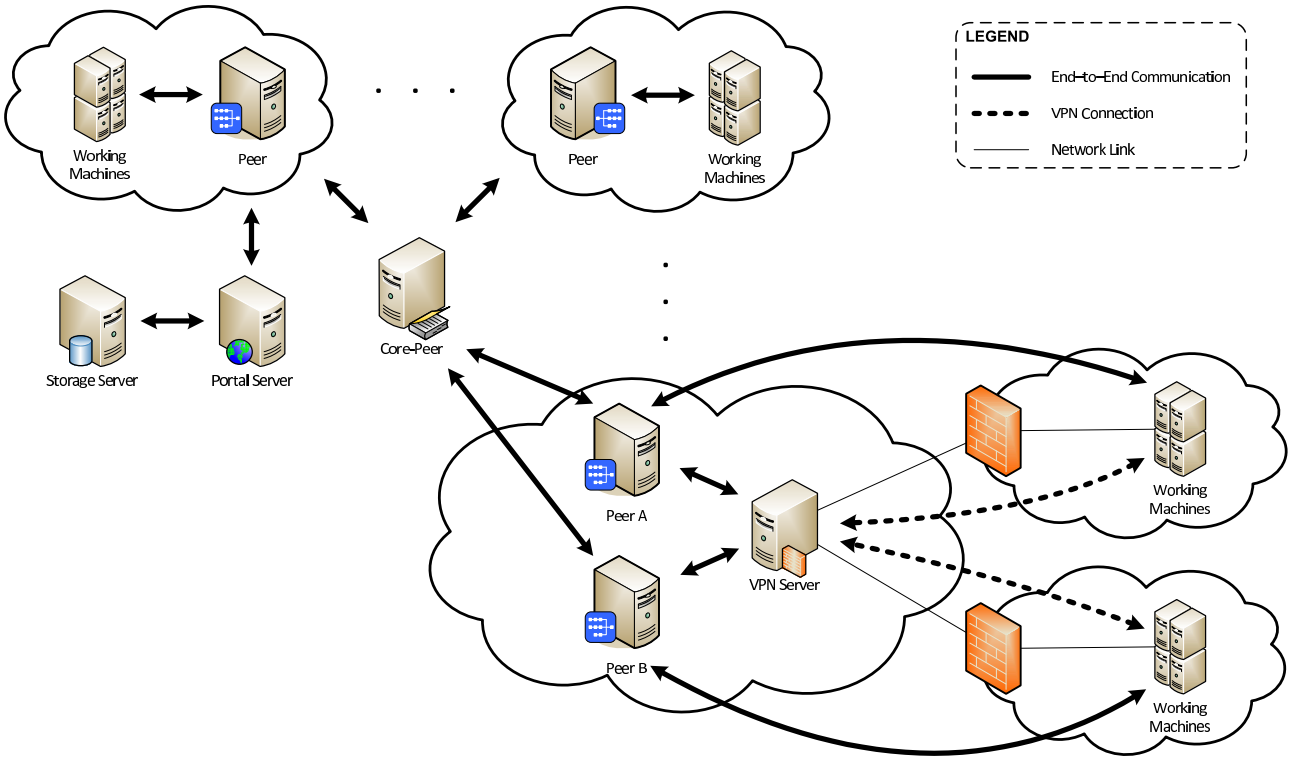


Fig. 2: Architecture of ShareGrid.

2.2.3 Managing multi-core resources

Although practically all relatively recent machines are equipped with multi-core processors, OurGrid provides no mechanisms to take advantage of this feature. Therefore, in order to fully exploit multi-core machines, we had to devise our own solution. In particular, in ShareGrid we adopted two complementary techniques to manage multi-core processors, namely:

– *Use of multiple User Agents per working machine*: this technique consists in running, on the same physical machine, a distinct User Agent for each core of its processor. This solution is very simple, but it may result in performance bottlenecks when different tasks make an intensive use of the same resource (e.g., CPU or memory), or when too many User Agents are started on the same machine. To limit these phenomena, we devised a rule-of-thumb, crafted by considering the characteristics of current ShareGrid applications, in which the number of User Agent is set to $\min\{C, \max\{M/1024, 1\}\}$, where C is the number of cores and M is the amount of RAM (in MB) installed on the machine. In practice, this rule states that each User Agent must receive at least 1,024 MB of RAM, and at most one User Agent per core can be executed on a given machine. However, it should be noted that the amount of RAM allocated to a given User Agent is imperatively controlled by the operating system of the machine, so at any given time some User Agents may receive more that 1,024 MB of RAM, while other ones may receive less than that amount of memory. Furthermore, it should be noted that a given User Agent is not statically allocated to a specific core (this is typically not permitted by standard operating systems). Conversely, the core on which a given task (allocated to the User Agent) runs may change each time the task is rescheduled by the operating system.
– *Use of multiple virtual machines per working machine*: this technique consists in allocating, on the same physical machine, a distinct virtual machine (hosting a User Agent) for each core of its processor. This solution requires more configuration efforts than running several User Agents as user processes, but brings two major advantages: (1) each virtual machine is an isolated runtime environment which contributes to increase security and reliability, and (2) it allows a single machine to simultaneously run different operating systems and, consequently, to dynamically increase or decrease the number of resources running a given operating system in order to match the needs of applications. The main disadvantage is that this solution usually requires an amount of physical resources larger than those necessary to run multiple User Agents without system virtualization, and requires the availability of a processor supporting virtualization. The same rule-of-thumb devised for the multiple User Agent per physical machine case has been used

to determine the number of virtual machines to run on a given node. In this case, however, it is possible to bind a given virtual machine to a specific core.

2.2.4 Enforcing Security

ShareGrid uses various authentication mechanisms to ensure that only legitimate users have access to the infrastructure. These mechanisms control the two points of user access, namely the individual peers and the ShareGrid Portal.

The access to individual peers, performed through MyGrid, is restricted to legitimate users by means of an access control list that, for each peer, contains the network addresses of the machines that are allowed to connect to that peer. Furthermore, the ShareGrid Portal provides an authentication mechanism for registered users, that have to authenticate themselves before being allowed to access its services.

The security mechanisms provided by OurGrid encompass also an application sandboxing technique named *Sandboxing Without A Name* (*SWAN*) [26], based on the Xen [27] virtualization platform, in which user tasks are executed within a Xen virtual machine where access to the network interface is disabled and access to the hard disks of the physical machine is limited to specific areas. The OurGrid software distribution includes also a Debian-based virtual machine already configured to be used within SWAN.

Despite the greater level of security it provides, however, SWAN is not used in ShareGrid for the following reasons. Firstly, the use of SWAN results in a significant overhead, since a virtual machine must be booted each time a task starts its execution, and shut down when the above execution terminates. This delay is particularly harmful for tasks characterized by a relatively short duration, that cannot amortize the high startup and shutdown costs typical of SWAN. Secondly, many ShareGrid machines do not provide hardware support to virtualization, and running SWAN on them would result in an excessive degradation of performance. Thirdly, since Xen runs only on Linux-based systems, SWAN could provide security only to a subset of ShareGrid resources, while for the remaining ones (running alternative operating systems, see Table 1 in Section 2.3) other security mechanisms have to be adopted. Finally, the Debian-based virtual machine distributed with OurGrid is really minimalist in its software endowment, so most ShareGrid applications could not run inside it unless that virtual machine is reconfigured in order to equip it with a larger (virtual) storage device on which to install the additional packages required by applications. Performing this reconfiguration without impacting on the already configured SWAN service is, however, somewhat complicated.

It is worth to point out that, in spite of this relatively simple approach to security, during the three-years period of ShareGrid operation we did not have any security incident. This is not a surprise, given the nature of the ShareGrid user community and the effectiveness of the user authentication mechanisms described above.

2.2.5 Dealing with software dependencies

The heterogeneity of the hardware and software platforms characterizing the resources of ShareGrid, as well as the need of ensuring application agnosticism, naturally yields to software dependency problems. As a matter of fact, a given application may require a specific hardware or software platform, or the availability of specific software components or libraries. Installing these packages a priori is clearly impossible, while doing so on-demand is unfeasible, as OurGrid lacks appropriate mechanisms, and using statically-compiled executables is not always possible (for instance, for legacy applications whose source code is unavailable). As a consequence, when choosing a specific resource to run a given task, the scheduling substrate (provided by OurGrid) must ensure that the task dependencies are satisfied by the resource chosen for its execution.

We addressed this problem by exploiting the matchmaking mechanism provided by OurGrid as follows. The features of resources (including their hardware/software characteristics and the presence of specific software libraries) are expressed as textual attributes, while task requirements are specified as boolean expression involving these attributes. Therefore, the scheduling policy used by OurGrid, that sends a given task only to those resources satisfying all its requirements, ensures that the peer dispatches tasks only on resources providing all the additional software components they require.

2.3 ShareGrid Resources

In this section we provide an overview of the characteristics of the physical resources that are currently included in ShareGrid. At the moment of this writing, ShareGrid sites provide more than 250 machines, whose features are reported in Table 1, hosting 314 distinct User Agents able to fully exploit multi-core processors (when available).

As reported in Table 1, ShareGrid resources are provided by 8 independent institutions distributed in 13 different sites: the *CSP* research center, the *TOP-IX* non-profit organization, the Department of Computer and Information Science at University of Genova (*UniGE – DISI*), the Department of Computer Science at University of Piemonte Orientale (*UniPMN – CS*), the *Re.Te.* telecommunications network division at University of Turin (*UniTO – Re.Te.*), the Department of Computer Science at University of Turin (*UniTO – CS*, 5 laboratories), the Department of Drug Science and Technology at University of Turin (*UniTO – DSTF*, 2 laboratories), and the Department of Economics and Finance at

Site	# Machines	# Cores	# User Agents	CPU Type	Clock (GHz)	Memory (MB)	Disk (GB)	OS
CSP	5	5	5	Intel Pentium 4	1.60	768	20	Linux 2.6.8
	4	4	4	Intel Pentium 4	1.60	512	20	Linux 2.6.8
TOP-IX	3	12	12	Intel Xeon Quad-Core	2.00	4,096	130	Linux 2.6.26
UniGE – DISI	2	4	4	UltraSPARC IIIi	1.34	2,048	73	SunOS 5.10
	24	48	48	Intel Core 2 Duo	2.66	1,024	4	Linux 2.6.24
UniPMN – CS	49	49	49	Intel Pentium 4	2.80	1,024	34	Linux 2.6.27
UniTO – Re.Te.	5	20	9	AMD Opteron Quad-Core	2.30	4,096	7	Linux 2.6.26
UniTO – CS (I)	1	2	1	Intel Xeon Dual-Core	2.33	4,096	250	Linux 2.6.18
	1	2	1	Intel Core 2 Duo	2.00	2,048	100	Linux 2.6.27
UniTO – CS (II)	34	34	34	Intel Pentium 4	2.53	2,048	30	SunOS 5.10
UniTO – CS (III)	12	12	12	Intel Pentium D	2.80	2,048	50	Windows XP
UniTO – CS (IV)	60	60	60	Intel Pentium 4	2.40	2,048	35	Windows XP
UniTO – CS (V)	23	23	23	AMD Athlon XP	2.10	1,024	75	Windows XP
	7	7	7	Intel Pentium 4	3.00	1,024	35	Windows XP
UniTO – DSTF (I)	1	1	1	AMD Athlon XP	2.60	512	10	Linux 2.6.18
	1	1	1	AMD Athlon XP	2.60	256	10	Linux 2.6.18
	2	2	2	Intel Pentium 4	2.80	512	10	Linux 2.6.18
	2	2	2	Intel Pentium 4	2.80	256	10	Linux 2.6.18
	1	2	1	Intel Pentium D	3.20	2,048	300	Windows XP
	2	4	4	AMD Athlon 64 X2	2.50	1,024	70	Linux 2.6.18
	2	2	2	AMD Athlon 64	3.20	512	15	Linux 2.6.18
UniTO – DSTF (II)	10	10	10	Intel Pentium 4	2.80	256	10	Linux 2.6.18
	4	8	8	Intel Pentium Dual	2.00	2,048	70	Linux 2.6.18
	2	4	4	Intel Pentium D	3.00	1,024	120	Linux 2.6.18
UniTO – ECO	1	8	8	Intel Xeon Quad-Core	1.86	4,096	300	Linux 2.6.27
	1	4	2	Intel Core 2 Quad-Core	2.40	4,096	300	Linux 2.6.27

Table 1: ShareGrid resources. The TOP-IX and (partially) the UniTO – DSTF machines exploit multi-core processors by running several User Agent processes, while in the UniGE – DISI, UniTO – Re.Te. and UniTO – ECO cases virtual machines (based either on VMware [28] or Solaris Containers [29]) are used for this purpose.

University of Turin (*UniTO – ECO*). Most of these sites are interconnected by the *GARR-G* network, a 100 Mbps network provided by the *Italian Academic & Research Network* (*GARR*) [30], enabling low latency communications; furthermore, for some of these sites, the capacity of the network will be soon increased by means of the next generation version of GARR-G (named *GARR-X*), that will provide capacities ranging from 2.5 Gbps (in the preliminary phase) to 10 Gbps.

For each site, Table 1 reports the number of physical machines, the total number of cores (the sum of the number of cores provided by individual machines), the total number of User Agents (that determines the total number of ShareGrid tasks that can be simultaneously executed by that site), the processor families and frequencies (in GHz) of those machines, the RAM size (in MB) of each machine, the size of their disks (in GB), and the operating systems installed on them.

Most of the ShareGrid sites follow the rule-of-thumb discussed in Section 2.2.3; however, it is important to note that, being a practical rule and not a policy to which every site must adhere, each site is free to use the User Agent deployment strategy that better fits its needs. This is essentially due to either the nature of the applications submitted by the users of that site (e.g., compute-intensive applications), or the presence on a same machine of different ShareGrid components (e.g., the Peer Agent and the User Agent). For instance, the *UniTO – Re.Te.* site runs 2 User Agents per ma-

chine (instead of 4, like the rule-of-thumb would suggest) on all machines except for one which is used for executing the Peer Agent and one User Agent.

As highlighted by this table, in ShareGrid there is a high degree of machine heterogeneity; indeed, these machines are equipped with different families of Intel, AMD and SPARC processors (e.g., from Intel Pentium 2 to Intel Xeon Quad-Core), offer a diverse range of primary and secondary storage size (e.g., from 256 MB to 4 GB of RAM and from 4 GB to 300 GB of disk), and run various operating systems (including Linux, Windows and SunOS). These machines, as prescribed by the Volunteer Computing paradigm, can be used by ShareGrid only when they are not used by the respective owners, so their actual number may strongly fluctuate over time; in Section 4.1 we will give more insight about resource availability in ShareGrid.

3 ShareGrid Applications

As already anticipated in the Introduction, ShareGrid supports only Bag-of-Tasks applications. At the moment of this writing, the following applications have been ported to ShareGrid, and are actively used by the corresponding users:

- *Distributed scene rendering*. Scene rendering is a typical compute-intensive activity, where a set of scenes of a given movie must be rendered via software in order to add in static and dynamic features (like bitmap or procedural textures, lights, bump mapping, etc.), and to stitch them together for making the final animation. Bag-of-Tasks implementations of *Blender* [31] and *POV-Ray* [32], in which bags correspond to scenes, and tasks of a bag to different frames of the same scene, are used on ShareGrid by professionals working in the computer graphics and animation fields.
- *Agent-based simulation of economic systems*. *Parei* [33, 34] is an agent-based simulation system, modeling over 500,000 production units composing the economy of the Piemonte region in Northwestern Italy, used to understand and estimate economic dynamics and for the evaluation of economic effects of proposed public policies. A Bag-of-Tasks implementation of Parei is used to perform parameter sweeping experiments, in which each combination of scenarios and model parameters corresponds to an independent task.
- *Simulation of molecular systems*. Various research activities in Drug Science, such as *virtual screening* and the reliable estimation of the free energy of binding between a ligand and its target, require to perform large set of simulation of the dynamic behavior of molecular systems. ShareGrid is currently being used to perform such studies [35] by means of a Bag-of-Tasks implementation of molecular simulation, where each simulation corresponds to a distinct task.
- *Simulation of scheduling algorithms for distributed systems*. Discrete-event simulation is often used in Computer Science to study the behavior of a scheduling algorithm for different scenarios, or the comparison of different algorithms for the same set of scenarios. These studies can be naturally performed by simultaneously executing many independent simulations in parallel. A Bag-of-Tasks discrete-event simulation engine is being used to perform parameter sweep studies of scheduling algorithms for Desktop Grids [36].
- *Evaluation of Classifier Systems*. Classification is the task of recognizing an object or event as an instance of a given class and represents one of the problems most frequently found in computer applications. For instance, medical and fault diagnosis, prognosis, image recognition, text categorization, adaptive user profiling are well known instances of classification tasks. The evaluation of algorithms able to perform classification is an important step for the selection of the best alternative for a given application domain, and can be naturally implemented as a Bag-of-Tasks application in which each task corresponds to a different setting of the algorithm parameters. ShareGrid is used to perform parameter sweep studies concerning the performance of *Support Vector Machine* (*SVM*) [37] classifiers when classifying distinct users of a computer system.
- *Evaluation and suppression of noise caused by flows over a cavity*. The evaluation and suppression of noise caused by flows over a cavity is an important problem in aeronautical research. A novel method (developed by people of the Politecnico di Torino), which allows the prediction of the emitted noise based on 2-dimensional velocity field data, experimentally obtained by *Particle Image Velocimetry* (*PIV*), has been implemented as a Bag-of-Tasks application running on ShareGrid. In this application, distinct tasks correspond to the different parameter settings required to calculate the hydrodynamic pressure from the PIV velocity data (one for each point of a time series representing velocity fields).
- *Automatic annotation of 3D multi-modal Magnetic Resonance images*. Automatic image annotation (also known as *automatic image tagging*) is the process by which a computer system automatically assigns metadata, usually in the form of captioning or keywords, to a digital image, in order to make possible its semantic (i.e., content-based) retrieval. A novel method [38], based on the integration of supervised and unsupervised learning techniques, has been implemented as a parameter sweep application and is being run on ShareGrid.

4 Performance Evaluation

In this section we present an experimental evaluation of the performance attained by ShareGrid when running some of the applications described in Section 3. In particular, we focused on distributed scene rendering (*DSR*), agent-based simulation of economic systems (*SES*) , and discrete-event simulation of scheduling algorithms (*SSA*). All the experiments were performed in the period elapsing from June 15^{th} to October 10^{th}, 2009. In each experiment, we executed a specific *application job* consisting in a bag-of-tasks characterized by the same *granularity* (i.e., the amount of total work assigned to each task). For each experiment, we measured the *average job completion time* (i.e., the average time taken to complete all the tasks in a job) and the *average task completion time* (i.e., the average time taken by a task to complete its work), and we compared it against the performance obtained by running a single application task (performing all the work) on a dedicated machine equipped with an Intel Xeon Quad-Core X3220 CPU and with 4 GB of RAM (henceforth referred to as the *reference machine*). In order to take into account fluctuations in the performance delivered by working machines, and in their availability, each experiment was repeated for 5 times, and average values of the performance indices of interest were computed by using the values collected during the individual runs.

Furthermore, in order to gain a better insight into the performance results we collected, we computed – for each application and experimental scenario – the number of *equivalent reference machines* (*ERM*) [39] provided by ShareGrid to that application, which is defined as the number of dedicated reference machines corresponding to the set of distinct machines actually used for the execution. The ERM, that depends on the number of machines satisfying the requirements of the tasks that are available at submission time, and on the relative performance delivered by these machines, gives a quantitative measure of the average performance gain that can be achieved by ShareGrid in the time frame in which the experiments were performed. The value $\mathrm{ERM}(A)$ for the application A is defined as:

$$\mathrm{ERM}(A) = \rho(A) \times \pi(A) \tag{1}$$

where $\rho(A)$ denotes the *average relative performance* delivered by ShareGrid to each task of application A, and $\pi(A)$ denotes the *average parallelism* achieved by that application (i.e., the average number of tasks of A that were executed in parallel). The quantity $\rho(A)$ is in turn defined as:

$$\rho(A) = \sum_{S_i \in Sites} w_i \cdot RelPerf(A, S_i) \tag{2}$$

where w_i denotes the normalized fraction of machines used to execute the tasks of A that were provided by site S_i, while $RelPerf(A, S_i)$ denotes the *relative performance* attained by these machines when running A. In this paper we computed $RelPerf(A, S_i)$ as the ratio of the execution time attained by the sequential version of A when running on the reference machine over the time obtained by running it on machines of site S_i [2].

For what regards the scheduling parameters described in Section 2.1, we used, in all the experiments, 2 replicas per task (i.e., we set $R = 2$), while, upon failure, each task was resubmitted for at most 3 times (i.e., we set $RS = 3$); the rationale for these choices is that using more than 2 replicas per task brings marginal performance benefits at a price of a much higher replication overhead [40], while $RS = 3$ is the default setting of OurGrid that we decided not to change.

Generally speaking, for a fixed amount of work that is to be carried out, the performance of a job depends on the ratio of the time taken to execute each task and the time required to stage-in and stage-out its input and output data. More precisely, each task must perform enough computation in order to amortize the time taken to transfer its input and output data. The execution time of a task depends in turn on the amount of total work assigned to that task (i.e., its granularity) and the speed of the machine on which it is executed.

Choosing the right granularity on ShareGrid is nontrivial for several reasons. First, determining the execution time of a given task requires the knowledge of the performance that the resource chosen for its execution will deliver when the task will be running; obtaining this information is complicated by the heterogeneity of machines, and by the fluctuations in performance that may arise when several tasks are running on the same physical machine. Second, determining stage-in and stage-out times can be challenging as well, as it depends on the specific network conditions at the moment of the transfer, and on the specific *transfer mode* chosen by the user to transfer data to and from the application. More precisely, OurGrid provides two distinct file transfer modes that the user can choose for his/her application, namely:

- the *PUT* mode, in which all the input files required by each task are transferred to the corresponding computing machine before the task starts, and removed once the computation is done. This mode saves disk space on the working machine, but requires the retransmission of the same input files in case they are required by another task allocated on it;
- the *STORE* mode, whereby input files are removed from working machines only when a prescribed maximum size (possibly infinite) is reached, so it is highly probable that it is not necessary to transfer them again in case they are required by another task. This, of course, comes at the price of a higher disk space occupancy on working machines.

[2] Possible fluctuations of these values were taken into account by computing the average of the execution time values collected during 5 independent runs.

In order to assess the performance delivered by ShareGrid under different application settings, we performed, for each application, a set of experiments in which we varied the granularity of the constituent tasks, and the transfer mode chosen to stage-in and stage-out data. The granularity of the tasks composing the application was set by keeping fixed the total amount of work that had to be accomplished, and by choosing a suitable number of elementary units of work (*work units*) assigned to each task.

The rest of this section is structured as follows. In Section 4.1 we firstly characterize the availability of ShareGrid resources, using information collected during the experimental period, in order to better understand the performance results we observed. Next, in the following three subsections we describe the performance results observed for distributed scene rendering (Section 4.2), agent-based simulation of economic systems (Section 4.3), and discrete-event simulation of scheduling algorithms (Section 4.4), respectively.

4.1 Resource Availability

As mentioned in Section 2.3, a ShareGrid resource corresponds to a distinct User Agent. A resource is said to be *connected* if the corresponding peer has joined the ShareGrid infrastructure, while it is said to be *disconnected* in the opposite case. A connected resource is said to be *available* if both the User Agent and its hosting machine are up and its owner is not using it. Conversely, a connected resource is said to be *unavailable* if either the User Agent or the corresponding machine is down, or if its owner has reclaimed it. An available resource that is currently executing a task is said to be *busy*, while it is said to be *idle* if it is not executing any application task. Note that a computing machine can have some of its hosted User Agents busy and other ones idle or even unavailable.

In order to study the availability of ShareGrid resources during the experimental period, we collected the status of each one of them with a sampling time of 30 minutes. The values of the resulting aggregated statistics are plotted in Fig. 3, where each point represents the average of all the observations collected in a given day, and the associated error-bar shows the 95% confidence interval. More specifically, every circle-shaped point denotes the average number of available resources, each square-shaped point indicates the average number of busy resources, and the dashed line represents the average number of idle resources, on a single day of the experimental period.

Fig. 3 shows that, for about the three-fourths of the experimental period (from the beginning to the mid of July, and from the end of August to the end of October), there were on average more than 150 available resources per day, corresponding to more than 59% of the average total number of ShareGrid resources (both connected and disconnected), and more than 72% of the average number of connected resources. Out of these 150 resources, more than 100 were idle per day, on average. Conversely, between the middle of July and the end of August (approximately, one-fourth of the experimental period) the number of available resources considerably decreased. This period of low availability (marked with two dot-dashed vertical lines) was probably due to the summer vacation period, during which the machines belonging to laboratories used for student activities were taken offline in order to save energy. Finally, the unavailability of any resource on July 18^{th} and 19^{th} was due to a scheduled maintenance, that required to take down all the infrastructure.

In order to gain a better insight about resource availability, we computed the (empirical) probability distribution of the number X of available resources (that is considered a random variable). More precisely, we computed the *Complementary Cumulative Distribution Function* $CCDF_X(x) = P\{X \geq x\}$ (plotted in Fig. 4a), that gives the probability that *at least* x machines are available, and some sample statistics of X (that are reported in Fig. 4b). From Fig. 4a we can observe that the maximum number of resources that are available with a probability close to 1 is 32 (the probability of such event is 0.983), and that the CCDF values quickly decrease for larger values of x. For instance, the probability of finding at least 150 available resources is 0.52, and becomes 0.18 if at least 180 resources are requested. The average number of available resources (see Fig. 4b) during the experimental period was 128.44, which corresponds to 62% of the average number of connected resources (207.16), and 50.2% of the maximum number of connected resources (256). It is worth to point out that these average numbers should be taken with caution, given the relatively high values of the coefficient of variation (the row labeled as *CoV*).

The results discussed above, although representative of the operational conditions under which we performed our experiments, cannot be considered typical of the ShareGrid infrastructure, as they incorporate also both the vacation and maintenance periods, that are exceptional events occurring only once in a while. Therefore, in order to obtain a characterization of the "typical" availability of ShareGrid resources, we censored the observations collected from the middle of July to the end of August, and we computed the CCDF and the sample statistics using only the remaining data (that are reported in Fig. 5a and Fig. 5b). As shown in Fig. 5a, when both the vacation and maintenance periods are not considered, the maximum number of resources that are available with probability close to 1 is 120 (while this number was 32 in the uncensored case). Furthermore, for values of x larger than 120, the CCDF values decreases in a way smoother than in the uncensored case (Fig. 4a), at least until the value of $x = 170$. For instance, the probability of finding at least 150 available resources is 0.85, while the one of

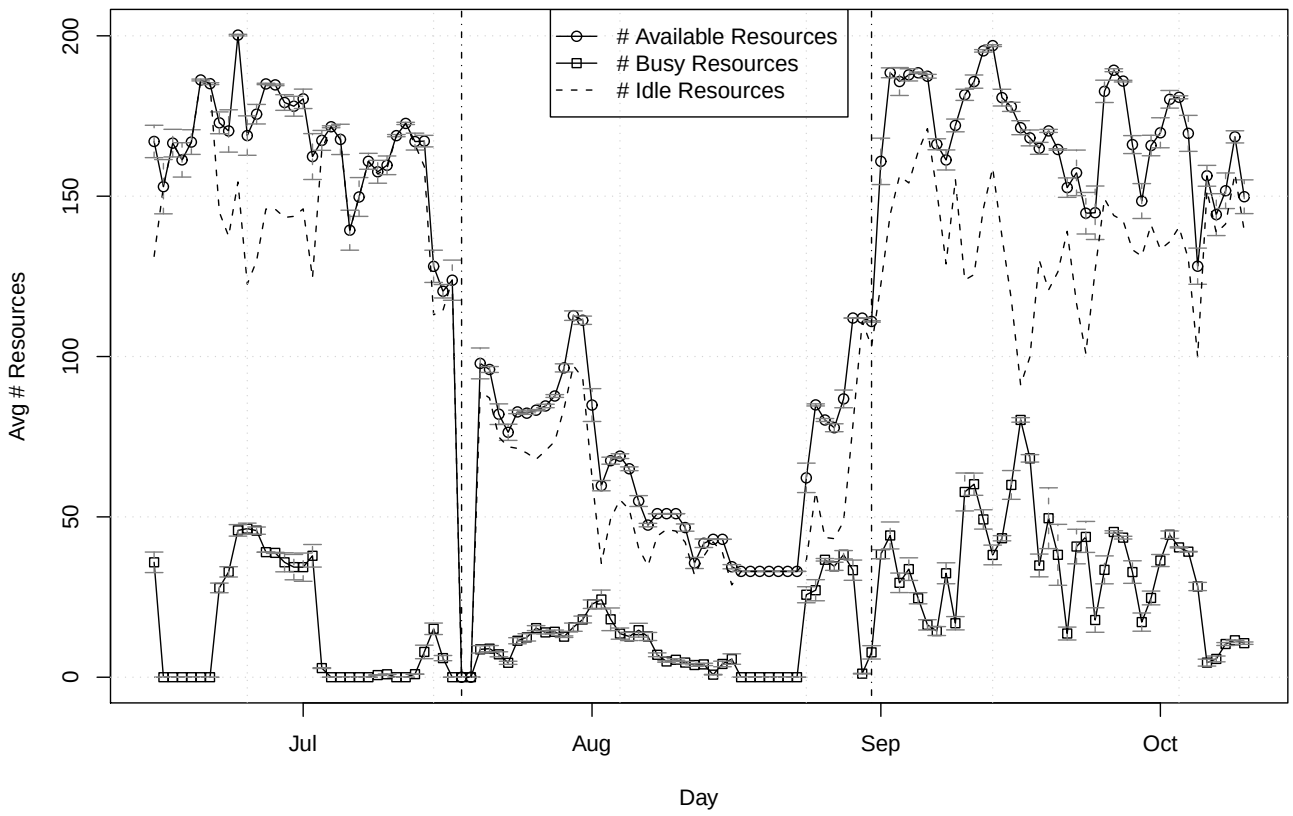


Fig. 3: ShareGrid daily availability and usage pattern. Each point is the average of daily observations (taken every half-hour) with the associated 95% confidence interval. The dot-dashed lines mark the begin and the end of the low-availability period.

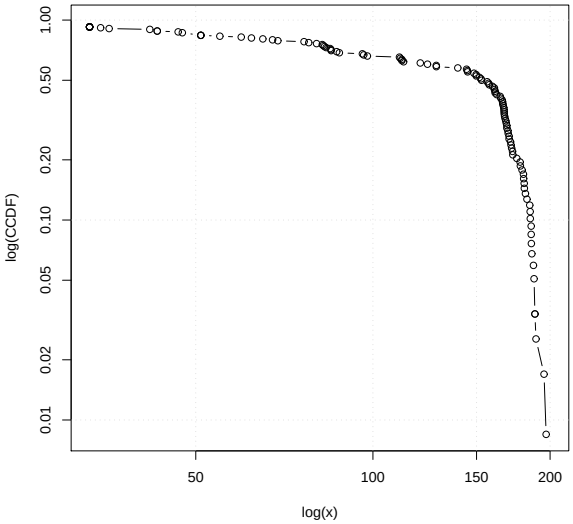
finding at least 180 resources is 0.29. The average number of available resources (see Fig. 5b) now becomes 167.92, corresponding to 70.3% of the average number of connected resources (238.69), and to 65.6% of the maximum number of connected resources (256). The low coefficient of variation indicates that these two values might better represent the typical ShareGrid availability.

Finally, for what regards usage patterns, the results in column *# Busy Resources* of both Fig. 4b and Fig. 5b show that daily utilization was rather low. More specifically, the average number of busy resources is approximately 15% of the average number of available ones in both cases.

4.2 Distributed Scene Rendering (DSR)

The experiments involving the DSR application were performed by means of the Blender [31] open-source 3D content creation suite (version 2.49*a*). In all these experiments, the work to be carried out by the application (the rendering of an animated scene) was split into 200 work units, each one corresponding to an independent scene frame. The size of the input data for each task was independent from its granularity, and amounted to about 50 MB, including both the scene file and the Blender executable (together with a collection of third-party libraries and tools required by Blender). We performed experiments with task granularities of 100, 40, 20, 10, 5, 2, and 1 frames/task, corresponding to jobs composed of 2, 5, 10, 20, 40, 100, and 200 tasks, respectively. Furthermore, for each task granularity we performed experiments with both the *STORE* and the *PUT* transfer mode, for an overall number of 14 experiments (each one consisting of 5 sequential runs). In all the experiments we restricted tasks to use only Linux-based machines equipped with at least 1,024 MB of RAM, since we used the Linux version of Blender, and this application is memory-intensive.

Fig. 6 and Fig. 7 show the results of the experiments for the *PUT* and the *STORE* transfer mode, respectively; for the sake of comparison, we also report the results of the experiments performed on the reference machine (denoted by label *R* in the figures). For each scenario, we report the av-

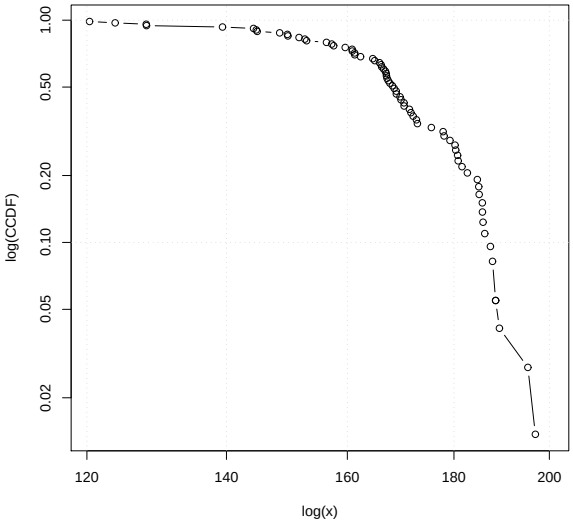


(a) Complementary cumulative distribution function (log scale).

Statistics	# Connected Resources	# Available Resources	# Busy Resources	# Idle Resources
Mean	207.16	128.44	20.08	108.40
S.D.	57.80	55.26	18.56	48.75
CoV	0.28	0.43	0.92	0.45
Median	236.25	154.66	14.53	125.00
IQR	92.99	88.67	31.74	82.47
Max	256.00	200.23	80.23	186.20
Min	0.00	0.00	0.00	0.00

(b) Sample statistics.

Fig. 4: ShareGrid availability during the overall experimental period.



(a) Complementary cumulative distribution function (log scale).

Statistics	# Connected Resources	# Available Resources	# Busy Resources	# Idle Resources
Mean	238.69	167.92	25.52	142.40
S.D.	11.61	16.94	20.25	19.91
CoV	0.05	0.10	0.79	0.14
Median	239.33	168.50	29.44	143.40
IQR	10.67	19.88	34.60	27.10
Max	256.00	200.23	80.23	186.23
Min	190.83	120.35	0.00	91.13

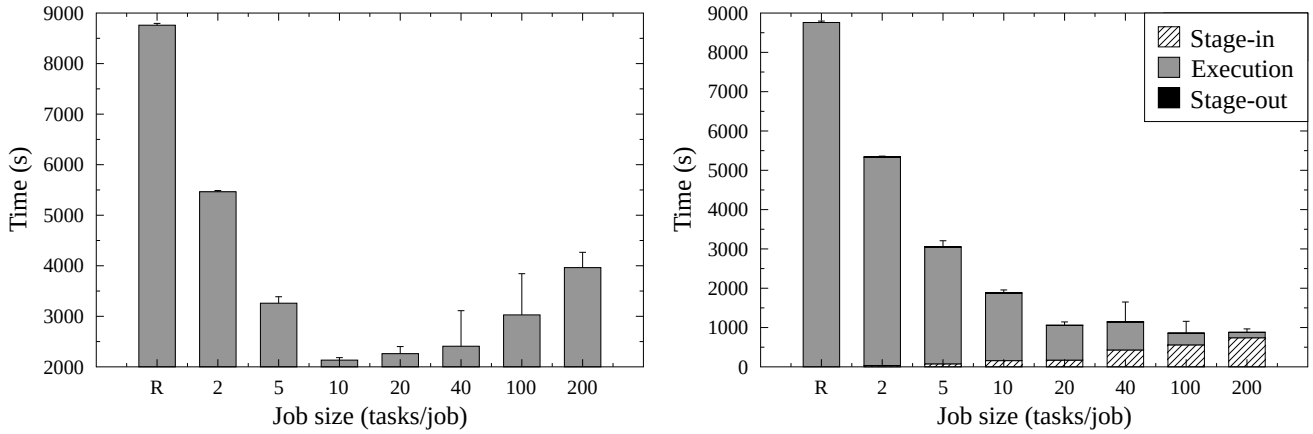
(b) Sample statistics.

Fig. 5: ShareGrid availability without the maintenance/vacation periods.

erage job completion time and average task completion time for increasing values of the job size (expressed as the number of tasks composing each job), along with their associated upper 95% confidence interval (shown as half error-bars).

Let us consider the results measured for the *PUT* transfer mode first, shown in Fig. 6, that clearly indicate that scene rendering with ShareGrid always outperforms the one done on the reference machine. Fig. 6a shows indeed that the average job completion time decreases as the job size increases up to 10 and 20 tasks/job; for these sizes, we observe a 4-fold reduction of the job completion time with respect to the sequential execution on the reference machine. For values of job size larger than 20 tasks/job, however, we observe that the average job completion time monotonically increases.

This phenomenon can be explained by looking at the results depicted in Fig. 6b, reporting the average task comple-



(a) Average Job Completion Time.

(b) Average Task Completion Time.

Fig. 6: DSR (*PUT* transfer mode) – label "R" represents the reference machine experiment.

tion time, broken down into stage-in, execution, and stage-out time. As a matter of fact, as can be seen from Fig. 6b, the higher the number of tasks/job (i.e., the smaller the amount of work assigned to each task), the higher the amount of stage-in time, that grows from 0.6% with 2 tasks/job to 84.1% with 200 tasks/job. This is due to the combination of the two following phenomena:

- the size of the input data (~50 MB) is independent of the task granularity, so very short tasks are not able to amortize the long stage-in times;
- the larger the number of tasks that are started concurrently, the larger the number of stage-in operations that must be performed at the same time. All these operations, however, compete for the outbound network bandwidth of the machine running the submitting peer (that, in our experiments, was the one associated with the ShareGrid Portal). Therefore, the network bandwidth available to each stage-in operation amounts to $1/n$, where n is the number of stage-in operation performed simultaneously.

Therefore, while the relative increase of stage-in time with respect to execution time is due to the first phenomenon, its absolute increase is due to the second one. The effects of the stage-out time are instead less evident (its value never exceeded 1% of the completion time) since (a) the amount of output data, for each task, is directly proportional to the task granularity (i.e., the finer the granularity, the smaller the amount of data that have to be staged-out for each task), and (b) each working machine sends the output data of its running tasks separately from the other ones, thus avoiding the bottleneck affecting the stage-in operations.

One might wonder whether a 4-fold reduction of the execution time can be considered a good result, provided that ShareGrid offered – in the experimental period – about 130 resources, on average. To answer this questions, let us consider the ERM values (see Eq. (1)) delivered by ShareGrid for the various scenarios considered in our experiments.

The values $RelPerf(\text{DSR})$ of the relative performance delivered by the machines of the various sites for the DSR application are listed in Table 2. The fractions w_i of machines provided by each site for the execution of DSR tasks are instead reported in Table 3a, while the resulting values of ρ, as well as the values of π and the final ERM values, are reported in Table 3b.

Site	$RelPerf(\text{DSR})$
TOP-IX	0.83
UniGE – DISI	0.16
UniTO – Re.Te.	0.78
UniTO – CS (I)	0.75
UniTO – ECO	0.98

Table 2: DSR – relative performance of involved machines (grouped by site).

As can be observed by Table 3b, the best average ERM value (6.16) was obtained for a job size of 40 tasks/job, meaning that in the best case ShareGrid provided to the application, on average, the equivalent of about 6 reference machines. Hence, the 4-fold reduction of the job execution time that we observed as maximum performance gain attained by the DSR application can be considered a good achievement, especially if also the stage-in and stage-out times are taken into account.

Let us discuss now the results concerning the *STORE* transfer mode, that are shown in Fig. 7. As for the *PUT* transfer mode, we observe that ShareGrid enables the application to achieve performance better than its sequential version for all the granularity values, but in this case the per-

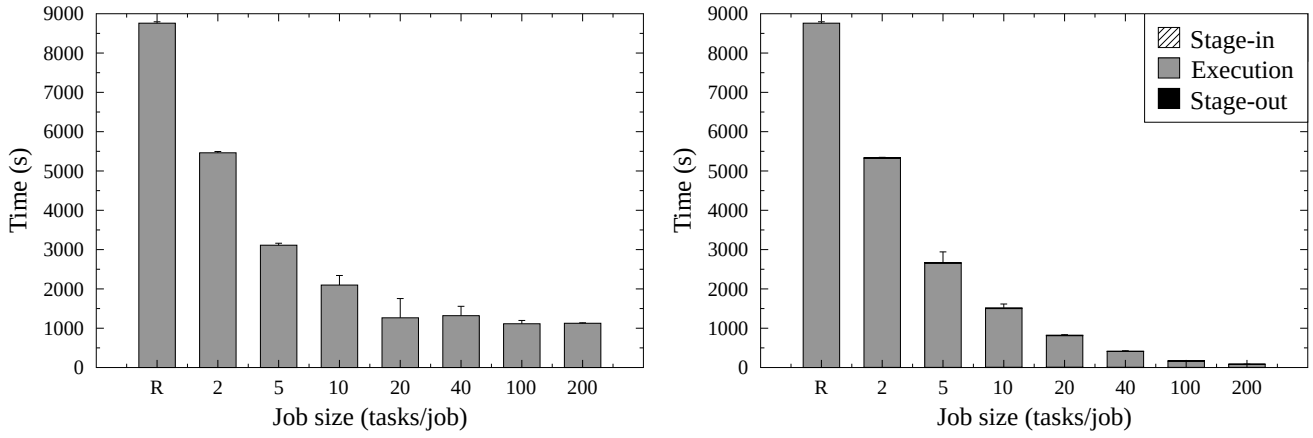
Job Size (tasks/job)	Site TOP-IX	UniGE – DISI	UniTO – Re.Te.	UniTO – CS (I)	UniTO – ECO
2	1.00	0.00	0.00	0.00	0.00
5	1.00	0.00	0.00	0.00	0.00
10	0.42	0.00	0.58	0.00	0.00
20	0.44	0.02	0.54	0.00	0.00
40	0.30	0.29	0.27	0.06	0.08
100	0.21	0.28	0.52	0.00	0.00
200	0.32	0.33	0.33	0.00	0.03

(a) w_i values.

Job Size (tasks/job)	ρ	π	ERM
2	0.83	2.00	1.66
5	0.83	5.00	4.15
10	0.80	5.80	4.64
20	0.79	4.60	3.63
40	0.63	9.78	6.16
100	0.62	5.02	3.11
200	0.60	6.10	3.66

(b) ρ, π and ERM values.

Table 3: DSR (*PUT* transfer mode) – w_i values and the equivalent reference machine (ERM) metric.



(a) Average Job Completion Time.

(b) Average Task Completion Time.

Fig. 7: DSR (*STORE* transfer mode) – label "R" represents the reference machine experiment.

formance gains are even larger, as can be seen from Fig. 7 that shows – for job sizes larger than 20 tasks/jobs – an 8-fold performance gain with respect to the reference machine. Furthermore, unlike the *PUT* case, the average job completion time remains practically constant for job sizes larger than 20. The reason for this behavior lies in the weaker influence of the stage-in time on the completion time. This can be observed from Fig. 7b, where the stacked-bar for the stage-in time seems to have disappeared; in fact, its influence accounts, on average, only for 0.07% with 2 tasks/job, and increases to just 3.8% with 200 tasks/job. This depends on the fact that, although the number of tasks started concurrently does not change with respect to the *PUT* mode, the number of concurrent stage-in operations is much smaller, since input data have to be transferred on a given resource only the first time a task is scheduled. Furthermore, as for the *PUT* transfer mode case, the stage-out time has a negligible impact on the completion time (it never exceeded 1%).

In order to assess whether the 8-fold performance gain observed for DSR when using the *STORE* transfer mode, let us consider again the ERM values measured for this case, reported in Table 4b, that have been computed by using the w_i values listed in Table 4a.

As can be observed from Table 4b, the maximum ERM value we observed (corresponding to the job size of 200 tasks/job) was 6.94, meaning that on average ShareGrid provided to DSR the equivalent of about 7 dedicated reference machines. Therefore, an 8-fold performance gain can be considered a very good achievement.

As a final note, we report that the we observed a relatively low number of failures for both transfer modes. More specifically, we observed a number of failures ranging from 0.4 to 8.4 failures per run (for the *PUT* transfer mode), and from 0.4 to 1.6 (for the *STORE* transfer mode).

4.3 Simulation of Economic Systems (SES)

The second application we considered for our performance evaluation study was the *Parei* application that, as discussed in Section 3, simulates the behavior of economic systems by using an agent-based model.

Job Size (tasks/job)	Site TOP-IX	UniGE – DISI	UniTO – Re.Te.	UniTO – CS (I)	UniTO – ECO
2	1.00	0.00	0.00	0.00	0.00
5	0.84	0.00	0.00	0.00	0.16
10	0.52	0.00	0.28	0.06	0.14
20	0.33	0.00	0.45	0.07	0.15
40	0.35	0.00	0.46	0.08	0.12
100	0.29	0.00	0.45	0.09	0.17
200	0.26	0.00	0.46	0.09	0.18

(a) w_i values.

Job Size (tasks/job)	ρ	π	ERM
2	0.83	2.00	1.66
5	0.85	3.90	3.32
10	0.83	4.70	3.90
20	0.82	6.53	5.35
40	0.82	6.20	5.08
100	0.83	5.90	4.90
200	0.83	8.36	6.94

(b) ρ, π and ERM values.

Table 4: DSR (*STORE* transfer mode) – w_i values and the equivalent reference machine (ERM) metric.

Each execution of Parei takes as input a tuple of parameters describing the initial conditions of the systems under analysis. In all the experiments the total amount of work to be performed consisted of 1,000 distinct tuples (representing the work units), that were split among a set of identical tasks, each one receiving the same amount of work units. We performed experiments with granularities of 100, 20, 10, 2, and 1 tuples/task, corresponding to jobs composed of 10, 50, 100, 500, and 1,000 tasks, respectively. For the sake of brevity, in this paper we report only the results obtained for the *STORE* transfer mode. The results corresponding to the *PUT* transfer mode, however, do not differ significantly, since the negligible size of the input and output data (less that 5 MB) implies that there is very little differences with respect to the *STORE* mode. Again, all our experiments were executed on Linux-based machines, as dictated by the requirements of the Parei application.

The results of these experiments are reported in Fig. 8, where we show both the average job completion time and the average task completion time. As for the distributed scene rendering case (see Section 4.2), we observe that the Bag-of-Tasks version of Parei always outperforms its sequential version, regardless of the task granularity, and in the best case the performance gain is about 4-fold. Furthermore, we observe again that the average job completion time decreases for increasing numbers of tasks/job until the optimal value of 100 tasks/job is reached, and then it monotonically increases for larger tasks/job values. Also in this case, this phenomenon is due to the impact of the stage-in and stage-out time on the task execution time which, as shown in Fig. 8b (where a logarithmic scale is used for the results in order to enhance readability), is always rather large, and grows from about 42% with 10 tasks/job to about 85% with 1,000 tasks/job. This rather steep increase is due, as already discussed for the distributed rendering case (see Section 4.2), to the bottleneck arising when multiple stage-in operations are performed simultaneously.

The ERM values (see Eq. (1)) measured for the SES applications, reported in Table 6b, corresponding to the *RelPerf* and w_i values listed in Table 5 and Table 6a, respectively, indicate that the maximum ERM values is 6.67, meaning that on average ShareGrid provides to this application the equivalent of about 6.5 reference machines. Therefore, a 4-fold performance gain can still be considered a good achievement, although less satisfactory than those observed for the DSR application.

Site	*RelPerf*(SES)	
TOP-IX	0.86	
UniGE – DISI	0.85	
UniTO – Re.Te.	0.51	
UniTO – CS (I)	0.88	
UniTO – ECO	0.77	(± 0.07)

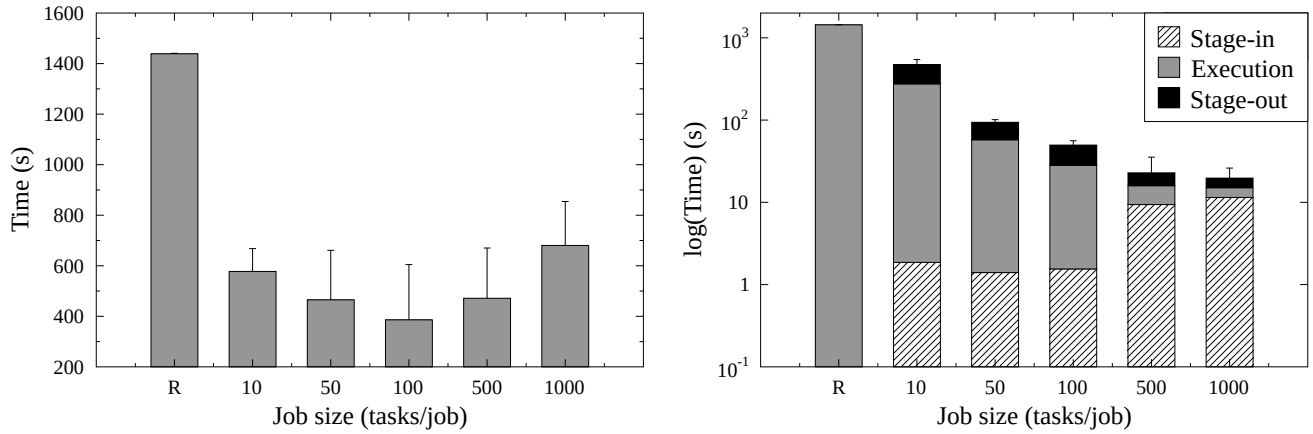
Table 5: SES – relative performance of involved machines.

As a final note, we observe that for the SES we observed a relatively low number of failed tasks. More specifically, we experienced failures only for job sizes of 1,000 tasks, where on average we had 0.20 ± 0.45 failures for each run, with a maximum of 1 failure.

4.4 Simulation of Scheduling Algorithms (SSA)

The last set of results we report in this paper are concerned with the discrete-event simulation of scheduling algorithms for distributed systems. In order to study the behavior of these algorithms, different system scenarios need to be simulated. The input for each simulation is a set of parameters describing a specific scenario, and the simulation of each scenario is totally independent from the other ones.

In all the experiments the total amount of work to be performed consisted in the simulation of 100 scenarios (representing the work units), that were split among a set of identical tasks, each one receiving the same amount of work units. We performed experiments for granularities of 20, 10,



(a) Average Job Completion Time.

(b) Average Task Completion Time.

Fig. 8: SES (*STORE* transfer mode) – label "R" represents the reference machine experiment.

Job Size (tasks/job)	Site TOP-IX	UniGE – DISI	UniTO – Re.Te.	UniTO – CS (I)	UniTO – ECO
10	0.60	0.04	0.18	0.00	0.18
50	0.43	0.00	0.35	0.00	0.22
100	0.52	0.00	0.30	0.00	0.17
500	0.39	0.26	0.23	0.00	0.12
1,000	0.23	0.42	0.25	0.01	0.09

(a) w_i values.

Job Size (tasks/job)	ρ	π	ERM
10	0.78	2.87	2.24
50	0.72	2.70	1.94
100	0.74	3.41	2.52
500	0.77	2.21	1.70
1,000	0.76	8.78	6.67

(b) ρ, π and ERM values.

Table 6: SES (*STORE* transfer mode) – w_i values and the equivalent reference machine (ERM) metric.

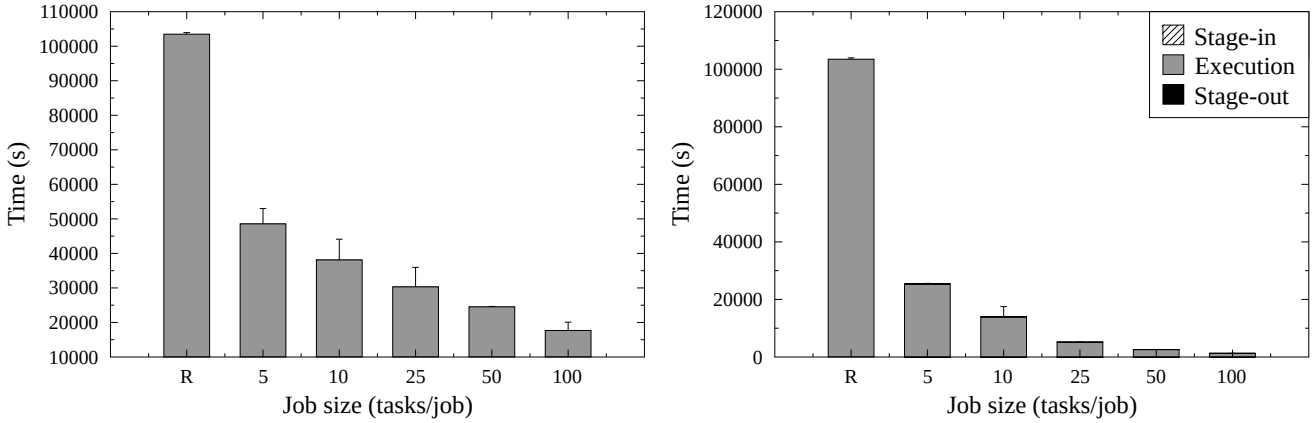
4, 2, and 1 scenarios/task, corresponding to jobs consisting of 5, 10, 25, 50, and 100 tasks, respectively. The stage-in and stage-out requirements for this applications are really minimal, as the total amount of data that needs to be transferred does not exceed 2 MB. As in the SES case (see Section 4.3), we report only the results obtained with the *STORE* transfer mode only, since those corresponding to the *PUT* transfer mode do not differ significantly. All the experiments were performed on Linux-based resources, since the discrete-event simulator was available for this platform only.

The results of the experiments are reported in Fig. 9, where again we note the significantly better performance provided by the Bag-of-Tasks version of the simulator with respect to the sequential one, that in the best case results in a 6-fold increase. Unlike the majority of the experiments discussed in previous subsections, however, in this case we do not observe the "bathtub"-like shape of the average job completion time. Conversely, we see that the average job completion time linearly decreases for increasing number of tasks per job (i.e., for lower granularity values). This is due to the fact that the time to complete the job is dominated by the job execution time, since the stage-in and stage-out times are negligible (in all our experiments it never exceeded 1% of the total execution time), as can be seen in Fig. 9b, where the corresponding areas of each vertical bar are not even visible. This is due to the small size of input and output files, resulting in negligible transfer times even in the presence of contention for the network capacity of the submitting peer.

The ERM values (see Eq. (1)) observed for the SSA application, reported in Table 8b, corresponding to the values of w_i and *RelPerf* reported in Table 8a and Table 7, indicate that in the best case ShareGrid provides, on average, the equivalent of about 24 reference machines.

Site	*RelPerf*(SSA)
TOP-IX	0.80
UniTO – CS (I)	0.96
UniTO – DSTF (I)	0.58
UniTO – DSTF (II)	0.70
UniTO – ECO	0.76

Table 7: SSA – relative performance of involved machines.



(a) Average Job Completion Time.

(b) Average Task Completion Time.

Fig. 9: SSA (*STORE* transfer mode) – label "R" represents the reference machine experiment.

Therefore, the 6-fold performance gain cannot be considered, per se, a good achievement if only the ERM values are considered, since this quantity does not take into account the number of failures experienced by the running tasks, that can greatly reduce performance. As a matter of fact, if failures occur relatively often, the ability of providing a relatively high ERM is thwarted by the fact that most of the tasks executed simultaneously do not complete their execution and must be therefore resubmitted. This is precisely what happened for the SSA application, that experienced a number of failures ranging from 4% (for jobs of size 5) to 70% (for jobs of size 10) of the job size. As a result, the performance gain attained by ShareGrid was significantly lower than the ERM values that were observed.

5 Related Work

The work described in this paper has its roots in the areas of Desktop Grids and of Grid Portals. In this section we review the previous works carried out in these areas, and we compare them with our work.

5.1 Desktop Grid Middleware Platforms

The Desktop Grid paradigm has been received an increasing attention in the past few years. Most of the previous work in this field has focused on the design and development of suitable middleware platforms able to support the deployment, use, and management of Desktop Grid infrastructures.

A large part of research efforts have focused on Volunteer Computing systems, and have lead to the development of various middleware platforms supporting this paradigms, such as *BOINC* [41] and its variants [42], and *XtremWeb* [43]. Compared to OurGrid (and, hence, to ShareGrid), these systems present a centralized architecture in which all the resource donors are coordinated by a single master server. Furthermore, they are based on an operational model (often referred to as *pull* model) in which a user that needs to execute an application has to register it into an *application repository*, has to submit work units to the system, and then the other contributing clients have to download these units to process them. Therefore, these platforms are not application agnostic (as we required in Section 2), since the submitting user has to set-up the system in an application-specific way prior to the submission. Conversely, OurGrid supports the operational model adopted in batch scheduling systems (often referred to as *push* model), where a user simply submits a batch of tasks, that are subsequently spread over the various resources belonging to the system, without having to perform any set-up operation before the submission.

A similar operational model is provided by Desktop Grids created with the *Condor* system [44, 45], a High-Throughput Computing system that can be used both to manage workload on a dedicated clusters and to farm out work to idle desktop computers. In Condor Desktop Grids, jobs are submitted into resource pools that can be either directly managed by Condor or, by means of the *Condor-G* [46] system, controlled by others systems. The scalability of Condor Desktop Grids is limited by the centralized management, in that jobs and resources are under control of a single server. Furthermore, resources are generally provided by a single institution. For these reasons, Condor Desktop Grids are commonly referred to as centralized institutional Desktop Grids [6]. In contrast, OurGrid allows the creation of, distributed, multi-institutional Desktop Grids (as ShareGrid), in that there is no central server and resources are usually provided by different institutions.

Significant research efforts have also been focused on the interoperability between service and opportunistic Grids in order to bring each other the best of both worlds. This kind of interoperation has been pioneered by the *Lattice*

Job Size (tasks/job)	Site TOP-IX	UniGE – DISI	UniTO – Re.Te.	UniTO – CS (I)	UniTO – ECO
5	1.00	0.00	0.00	0.00	0.00
10	0.90	0.02	0.02	0.02	0.04
25	0.70	0.04	0.02	0.15	0.10
50	0.52	0.05	0.01	0.17	0.26
100	0.70	0.02	0.00	0.00	0.27

(a) w_i values.

Job Size (tasks/job)	ρ	π	ERM
5	0.80	5.00	4.00
10	0.80	7.00	5.60
25	0.79	19.80	15.64
50	0.78	31.00	24.18
100	0.79	19.60	15.48

(b) ρ, π and ERM values.

Table 8: SSA (*STORE* transfer mode) – w_i values and the equivalent reference machine (ERM) metric.

project [47] at University of Maryland and by the *SZTAKI Desktop Grid* [42]. However, both of them offer interoperability between a limited set of middlewares; furthermore, the proposed approach (namely, the *superworker approach*) has several drawbacks, primarily due to the centralized nature of its architecture. The *EDGeS* project [48] focuses on the design and development of a middleware-agnostic framework that is able to provide interoperability between potentially any kind of Grid middleware (at the moment, between *EGEE* [49], BOINC and XtremWeb). This is achieved by means of the *Generic Grid to Grid* (3G) bridge [50], a middleware-independent framework which provides a set of four components, including interfaces for job management and components for storing job information. With the 3G bridge, in order to enable a middleware to accept jobs coming from other middlewares, it is sufficient to provide an implementation of the 3G bridge *Grid handler* interface for that specific middleware. Furthermore, to enable a middleware to submit jobs to other middlewares, it is sufficient to provide an implementation of the 3G bridge *Grid plug-in* interface for that specific middleware. Unfortunately, at the time of this writing, there is no such implementation for the version of the OurGrid middleware used by ShareGrid; so, currently we are unable to take any advantage of the EDGeS project.

In addition to Volunteer Computing systems, in the recent past several research efforts have been devoted to the investigation of P2P Desktop Grid middleware platforms. As a result, several platforms have been proposed in the literature, such as *JNGI* [51] (which is based on the JXTA technology [52] and is focused on scalability, reliability, and self-organization), the *Personal Power Plant* [53] (also based on JXTA), and *Cohesion* [54]. However, some of these platforms (notably, JNGI and the Personal Power Plant) support only Java-based applications (in contrast, ShareGrid does not suffer from this limitation), and require centralized management operations to manage and maintain the infrastructure.

Other P2P Desktop Grid platforms, e.g. the *Organic Grid* [55], *Messor* [56], and the one described in [57], are still at the prototype stage and, as such, do not provide a solid substrate enabling the deployment, use, and management of a production Desktop Grid.

5.2 Grid Portals

Most of the existing Grid platforms provide a Web-based access to their user communities through a *portal*, in order to enable them to use Grid resources and services, and to submit and monitor their Grid applications. The majority of these portals are based on the *GridSphere* project [58], an open-source portal framework which provides a *portlet*-based JSR-168 compliant Web portal [59]. GridSphere supports various middleware platforms, like the *Globus toolkit* [60], *Unicore* [61], and *gLite* [62], through portlet components called *GridPortlets*, that define a single and consistent high-level API between portlets and underlying Grid services.

The *Open Grid Computing Environments* (OGCE) project [63] is another open-source collaborative project that leverages Grid portal research and development from various universities and research institutions. Similarly to GridSphere, the *OGCE Portal* is a portlet-based JSR-168 compliant portal framework which uses the *Java CoG* toolkits [64] and *GridPort* [65] as its main service APIs for accessing Grid resources.

The ShareGrid Portal is a portal framework too. The main architectural difference with the above two portal frameworks, is the mechanism used for supporting a new Grid middleware. As a matter of fact, while the GridSphere and OGCE approach makes use of portlet components for extending the range of supported middlewares, the ShareGrid Portal extension mechanism consists in a series of *Plain Old Java Object* (POJO) [66] interfaces, defining the high-level behaviour of a middleware, which are deployed in the portal by means of simple Java libraries (JARs). For what concerns the functional aspect, a possible difference is that GridSphere and OGCE delegate each middleware portlet for providing its user interface, while the ShareGrid Portal provides a uniform view independent from the underlying mid-

dleware. Another difference is that, compared to GridSphere and OGCE, users are not aware of the underlying Grid services they are actually using; this behavior has been intentionally provided in order to make user interaction as easy as possible.

The *P-GRADE Portal* [67] is a workflow-oriented Grid portal which supports different types of middleware like the Globus toolkit, *LCG* [68], gLite and XtremWeb. It is built upon GridSphere, for the Web interface, *JavaGAT* [69], for interacting with the middleware, and *Condor DAGMan* [70] for managing a workflow application. The main difference with the ShareGrid Portal is the nature of the supported Grid applications; the P-GRADE Portal is oriented to workflow applications, with some extensions for parameter sweep applications; the ShareGrid Portal currently supports Bag-of-Tasks applications, which are a superset of the family of parameter sweep applications but are more limited than workflow applications. Another difference is the range of supported middlewares and the type of interaction with them. The P-GRADE Portal is a Globus-based, multi-Grid collaborative portal. It is able to simultaneously connect to different Globus-based Grid systems and let their user communities to migrate applications between Grids. Furthermore, through the integration with the *Grid Execution Management for Legacy Code Applications* (GEMLCA) project [71], it enables both the deployment of legacy code applications as a Grid service (without the need of rewriting them) and the interoperability between different types of service-oriented Grid middleware (like Globus toolkit version 4). The ShareGrid Portal can be considered a multi-Grid portal as well but with three important differences: 1) it is not strictly focused to Globus-based middlewares since it relies to a high-level middleware abstraction layer, 2) a running instance of it is currently able to support only one type of middleware at a time (which however can be changed at deployment time), and 3) it does not take care of the multi-Grid collaboration aspect by itself, but this is relied upon the underlying middleware.

6 Conclusions and Future Work

In this paper we described ShareGrid, a Peer-to-Peer Desktop Grid that aggregates computational resources contributed by independent research laboratories, and that enables its users to use them in a very simple and transparent way. At the moment of this writing, ShareGrid includes more than 250 machines contributed by 8 distinct institutions located in the Northwestern Italian regions. Our experience with ShareGrid indicates that P2P Desktop Grids can provide an effective answer to the computing needs of small research laboratories that do not have enough financial and human resources to set up or participate to a traditional Grid infrastructure. As a matter of fact, the high usability levels due to both OurGrid and to the ShareGrid Portal, and the ease of management brought by the proper configuration of OurGrid mechanisms, resulted in a high degree of user productivity (and satisfaction as well). Furthermore, as demonstrated by the results we obtained by running three distinct applications (characterized by quite different resource requirements and Bag-of-Tasks representations), ShareGrid is able to provide satisfactory performance even with low-end machines characterized by a high degree of volatility. We believe that these results, although specific of ShareGrid, hold in general for P2P Desktop Grids.

In ShareGrid still there are some issues that need to be solved. One of the most important avenue of research we plan to pursue is concerned with the provision and integration of an efficient data transfer mechanism within ShareGrid in order to better support data-intensive applications in addition to compute-intensive ones. In principle, data management in ShareGrid would not be an issue when jobs are submitted through the MyGrid installed on the user local machine. However, it becomes an issue when the ShareGrid Portal is used as the primary mean for job submission. Indeed, the ShareGrid Portal and the Storage Server can be viewed as centralized components and thus they may become a bottleneck when multiple data-intensive jobs are simultaneously transmitting or receiving data, since each of them competes with the others for network bandwidth capacity (as discussed in Section 4.2). A natural way to mitigate this issue is to deploy instances of the Portal and of the Storage Server on several sites. In this way, the inbound and outbound data, that originally went through the single Portal and Storage Servers, would be distributed among these instances. However, it is worth noting that there still might be "local" (i.e., per-site) bottlenecks if users of some of these sites prevalently submit data-intensive jobs.

The problem of efficiently handling the transfer of input and output files affects not only ShareGrid, but is considered of general relevance by the scientific community, as demonstrated by the current research efforts devoted to its solution. Most of these efforts are focused on the integration of Peer-to-Peer file transfer protocols into the Desktop Grid middleware. For instance, in [72] the BOINC middleware is properly modified in order to efficiently manage huge data transmission through the *BitTorrent* protocol [73]. For ShareGrid, however, we are interested in a more modular approach, where a Peer-to-Peer data distribution technology can be easily integrated without the need to change the underlying middleware. A similar approach is pursued in [74], where the BitTorrent protocol is integrated into the XtremWeb middleware, and in [75], where the *GridTorrent* protocol is proposed as a variant of the BitTorrent protocol (using a Replica Location Service for managing and finding file replicas, in place of traditional ".torrent" files), and integrated into the *PlanetLab* infrastructure [76]. Our future

work thus will be concerned with the integration in ShareGrid of efficient data transfer systems, like *File Mover* [77], *BitDew* [78], *JuxMem* [79], or *FreeLoader* [80], just to name a few.

Finally, we are considering to upgrade ShareGrid to the version 4 of the OurGrid middleware which, at the time of this writing, has just reached its stable version. There are several reasons to migrate to this new version. The most important ones include a new and more stable underlying communication protocol, and the possibility to interoperate with service Grids (actually, with the gLite middleware) by means of the EDGeS-like 3G bridge developed under the *EELA-2* project [81], a multidisciplinary project involving more than 50 institutions both in Europe and in Latin America, and aiming at enhance the e-infrastructure of Latin American countries.

Acknowledgments

The authors wish to thank the members of the ShareGrid team, and in particular Guglielmo Girardi, Marco Botta, and Sergio Rabellino for their contributions to the development and deployment of ShareGrid. The authors are also specially grateful to Francisco Vilar Brasileiro for providing helpful insights on the OurGrid middleware. Finally, the authors would like to thank the anonymous reviewers for their feedback on earlier drafts of this paper.

References

1. Foster, I. and Kesselman, C., editors. *The Grid: Blueprint for a Future Computing Infrastructure*. Morgan Kaufmann Publishers, (1999).
2. The DataGrid Project: High Energy Physics Data Grid Initiative. http://eu-datagrid.web.cern.ch/eu-datagrid/. Visited on Dec. 18^{th}, 2009.
3. The Grid Physics Networks (GriPhyN) project. http://www.griphyn.org. Visited on Dec. 18^{th}, 2009.
4. The Grid5000 Project. http://www.grid5000.fr. Visited on Dec. 18^{th}, 2009.
5. The NEESgrid Project. http://www.neesgrid.org. Visited on Oct. 18^{th}, 2009.
6. Choi, S., Kim, H., Byun, E., Baik, M., Kim, S., Park, C., and Hwang, C. Characterizing and Classifying Desktop Grid. In *Proc. of the* 7^{th} *IEEE International Symposium on Cluster Computing and the Grid (CCGRID'07)*, 743–748. IEEE Computer Society, (2007).
7. Kondo, D., Chien, A. A., and Casanova, H. Resource management for rapid application turnaround on enterprise Desktop Grids. In *Proc. of the 2004 ACM/IEEE conference on Supercomputing (SC'04)*, 17 (Pittsburgh, PA, USA, 2004).
8. Anderson, D. P., Cobb, J., Korpela, E., Lebofsky, M., and Werthimer, D. SETI@home: An experiment in public-resource computing. *Comm. ACM* **45**(11), 56–61 (2002).
9. The Great Internet Mersenne Prime Search. http://www.mersenne.org. Visited on Dec. 18^{th}, 2009.
10. The FightAids@Home Project. http://fightaidsathome.scripps.edu. Visited on Dec. 18^{th}, 2009.
11. The GRID.ORG Project. http://www.grid.org. Visited on Dec. 18^{th}, 2009.
12. Anderson, D. P. and Fedak, G. The computational and storage potential of volunteer computing. In *Proc. of the* 6^{th} *IEEE International Symposium on Cluster Computing and the Grid (CCGRID'06)*, 73–80 (Singapore, 2006).
13. Andrade, N., Cirne, W., Brasileiro, F., and Roisenberg, P. OurGrid: An approach to easily assemble Grids with equitable resource sharing. In *Proc. of the* 9^{th} *Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP'03)*, 61–86 (Seattle, WA, USA, 2003).
14. Cirne, W., Brasileiro, F., Andrade, N., Costa, L., Andrade, A., Novaes, R., and Mowbray, M. Labs of the world, unite!!! *J. Grid Computing* **4**(3), 225–246 (2006).
15. Anglano, C., Canonico, M., Guazzone, M., Botta, M., Rabellino, S., Arena, S., and Girardi, G. Peer-to-peer Desktop Grids in the real world: The ShareGrid project. In *Proc. of the* 8^{th} *IEEE International Symposium on Cluster Computing and the Grid (CCGRID'08)*, 609–614 (Lyon, France, 2008).
16. The ShareGrid project. http://dcs.di.unipmn.it/sharegrid. Visited on Dec. 18^{th}, 2009.
17. Cirne, W., Paranhos, D., Costa, L., Santos-Neto, E., Brasileiro, F., Sauvé, J., Silva, F. A. B., Barros, C. O., and Silveira, C. Running Bag-of-Tasks applications on computational Grids: The MyGrid approach. In *Proc. of the 2003 International Conference on Parallel Processing (ICPP'03)*, 407–416 (Kaohsiung, Taiwan, 2003).
18. Abramson, D., Giddy, J., and Kotler, L. High Performance Parametric Modeling with Nimrod/G: Killer Application for the Global Grid? In *Proc. of the* 14^{th} *International Parallel and Distributed Processing Symposium (IPDPS'00)*, 520–528 (Cancun, Mexico, 2000).
19. Casanova, H., Obertelli, G., Berman, F., and Wolski, R. The AppLeS parameter sweep template: User-level middleware for the Grid. *Sci. Program.* **8**(3), 111–126 (2000).
20. Stiles, J., Thomas M. Bartol, J., Salpeter, E. E., and Salpeter, M. M. Monte Carlo simulation of neuro-transmitter release using MCell, a general simulator of cellular physiological processes. In *Proc. of the* 6^{th} *annual conference on Computational neuroscience: trends in research (CNS'97)*, 279–284 (Big Sky, MT, USA, 1998).
21. Smallen, S., Casanova, H., and Berman, F. Applying Scheduling and Tuning to On-line Parallel Tomography. In *Proc. of the 2001 ACM/IEEE conference on Supercomputing (SC'01)*, 12 (Denver, CO, USA, 2001).
22. Silva, D. P. D., Cirne, W., and Brasileiro, F. V. Trading Cycles for Information: Using Replication to Schedule Bag-of-Tasks Applications on Computational Grids. In *Proc. of the International Conference on Parallel and Distributed Computing (Euro-Par'03)*, 169–180 (Klagenfurt, Austria, 2003).
23. Andrade, N., Brasileiro, F., Cirne, W., and Mowbray, M. Automatic grid assembly by promoting collaboration in peer-to-peer grids. *J. Parallel. Distr. Comput.* **67**(8), 957–966 (2007).
24. Santos, R., Andrade, A., Cirne, W., Brasileiro, F., and Andrade, N. Relative autonomous accounting for peer-to-peer Grids. *Concurrency Comput. Pract. Ex.* **19**(14), 1937–1954 (2007).
25. Anglano, C., Canonico, M., and Guazzone, M. The ShareGrid Portal: An easy way to submit jobs on computational Grids. Technical report TR-INF-2008-10-08-UNIPMN, University of Piemonte Orientale, October (2008).
26. Cavalcanti, E., Assis, L., Gaudencio, M., Cirne, W., and Brasileiro, F. Sandboxing for a free-to-join Grid with support for secure site-wide storage area. In *Proc. of the* 2^{nd} *International Workshop on Virtualization Technology in Distributed Computing (VTDC'06)*, 11 (Tampa, FL, USA, 2006).

27. Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., and Warfield, A. Xen and the art of virtualization. In *Proc. of the* 19^{th} *ACM Symposium on Operating Systems Principles (SOSP'03)*, 164–177 (The Sagamore, Bolton Landing, NY, USA, 2003).
28. VMware: Business Infrastructure Virtualization. http://www.vmware.com. Visited on Dec. 18^{th}, 2009.
29. Solaris Containers. http://www.sun.com/software/solaris/containers/. Visited on Dec. 18^{th}, 2009.
30. GARR: The Italian Academic & Research Network. http://www.garr.it. Visited on Dec. 18^{th}, 2009.
31. Blender. http://www.blender.org. Visited on Dec. 18^{th}, 2009.
32. POV-Ray: The Persistence of Vision Raytracer. http://www.povray.org. Visited on Dec. 18^{th}, 2009.
33. Boero, R. Un modello dell'economia piemontese come sistema complesso: interdipendenze economiche e territoriali tra centro e periferia. In *Produrre a Torino*, Russo, G. and Terna, P., editors. Torino, Italy (2006). (In Italian).
34. Boero, R. Dinamiche espansive del settore ict in piemonte: alcune analisi esplorative tramite simulazioni basate su agenti. Technical report, IRES Piemonte, Torino, Italy, (2006). (In Italian).
35. Tosco, P., Marini, E., Rolando, B., Lazzarato, L., Cena, C., Bertinaria, M., Fruttero, R., Reist, M., Carrupt, P.-A., and Gasco, A. Structure-Antioxidant Activity Relationships in a Series of NO-Donor Phenols. *ChemMedChem* **3**(9), 1443–1448 (2008).
36. Anglano, C. and Canonico, M. Scheduling algorithms for multiple Bag-of-Task applications on Desktop Grids: A knowledge-free approach. In *Proc. of the* 22^{nd} *IEEE International Symposium on Parallel and Distributed Processing (IPDPS 2008)*, 1–8 (Miami, Florida USA, 2008).
37. Cortes, C. and Vapnik, V. Support-Vector Networks. *Mach. Learn.* **20**(3), 273–297 (1995).
38. Basso, C., Ferrante, M., Santoro, M., and Verri, A. Automatic annotation of 3D multi-modal MR images on a Desktop-Grid. In *Proc. of the MICCAI-Grid 2009 Workshop (in conjunction of the MICCAI'09 conference)*, 56–65 (London, UK, 2009).
39. Kondo, D., Taufer, M., III, C. L. B., Casanova, H., and Chien, A. A. Characterizing and evaluating Desktop Grids: An empirical study. In *Proc. of the* 18^{th} *International Parallel and Distributed Processing Symposium (IPDPS 2004)* (Santa Fe, NM, USA, 2004).
40. Anglano, C. and Canonico, M. Fault-Tolerant Scheduling for Bag-of-Tasks Grid Applications. In *Proc. of the 2005 European Grid Conference (EuroGrid'05)*, number 3470 in Lecture Notes in Computer Science (Amsterdam, The Netherlands, 2005).
41. Anderson, D. P. BOINC: A system for public-resource computing and storage. In *Proc of the* 5^{th} *IEEE/ACM International Workshop on Grid Computing (in conjunction of the SC'04 conference)*, 4–10 (Pittsburgh, PA, USA, 2004).
42. Kacsuk, P., Kovacs, J., Farkas, Z., Marosi, A. C., Gombas, G., and Balaton, Z. SZTAKI Desktop Grid (SZDG): A flexible and scalable Desktop Grid system. *J. Grid Computing* **7**(4), 439–461 (2009).
43. Fedak, G., Germain, C., Neri, V., and Cappello, F. XtremWeb: A Generic Global Computing System. In *Proc. of the* 1^{st} *IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'01)*, 582–587 (Brisbane, Qld., Australia, 2001).
44. Thain, D. and Livny, M. Building reliable clients and servers. In *The Grid: Blueprint for a New Computing Infrastructure*, Foster, I. and Kesselman, C., editors. Morgan Kaufmann, 2nd edition (2003).
45. Thain, D., Tannenbaum, T., and Livny, M. Distributed computing in practice: the Condor experience. *Concurrency - Practice and Experience* **17**(2-4), 323–356 (2005).
46. Frey, J., Tannenbaum, T., Foster, I., Livny, M., and Tuecke, S. Condor-G: A computation management agent for multi-institutional grids. In *Proc. of the* 10^{th} *IEEE Symposium on High Performance Distributed Computing (HPDC10)* (San Francisco, CA, USA, 2001).
47. Myers, D. S., Bazinet, A. L., and Cummings, M. P. Expanding the reach of Grid computing: Combining Globus- and BOINC-based systems. In *Grid Computing for Bioinformatics and Computational Biology*, 71–84. John Wiley & Sons, Inc. (2008).
48. Cárdenas-Montes, M., Emmen, A., Marosi, A. C., Araujo, F., Gombás, G., Terstyanszky, G., Fedak, G., Kelley, I., Taylor, I., Lodygensky, O., Kacsuk, P., Lovas, R., Kiss, T., Balaton, Z., and Farkas, Z. EDGeS: bridging Desktop and Service Grids. In *Proc. of the* 2^{nd} *Iberian Grid Infrastructure Conference (IBERGRID'2008)*, 212–224 (Porto, Portugal, 2008).
49. Gagliardi, F. The EGEE European Grid infrastructure project. In *High Performance Computing for Computational Science - VECPAR 2004*, Lecture Notes in Computer Science, 194–203. (2005).
50. Urbah, E., Kacsuk, P., Farkas, Z., Fedak, G., Kecskemeti, G., Lodygensky, O., Marosi, A., Balaton, Z., Caillat, G., Gombas, G., Kornafeld, A., Kovacs, J., He, H., and Lovas, R. EDGeS: Bridging EGEE to BOINC and XtremWeb. *J. Grid Computing* **7**(3), 335–354. Special Issue: Grid Interoperability.
51. Verbeke, J., Nadgir, N., Ruetsch, G., and Sharapov, I. Framework for peer-to-peer distributed computing in a heterogeneous, decentralized environment. In *Proc. of the* 3^{rd} *International Workshop on Grid Computing (GRID'02)*, 1–12 (Baltimore, MD, USA, 2002).
52. Seigneur, J.-M., Biegel, G., and Jensen, C. D. P2P with JXTA-Java pipes. In *Proc. of the* 2^{nd} *international conference on Principles and practice of programming in Java (PPPJ '03)*, 207–212 (Kilkenny City, Ireland, 2003).
53. Shudo, K., Tanaka, Y., and Sekiguchi, S. P3: P2P-based middleware enabling transfer and aggregation of computational resources. In *Proc. of the* 5^{th} *IEEE International Symposium on Cluster Computing and the Grid (CCGRID'05)*, volume 1, 259–266. IEEE Computer Society, (2005).
54. Blochinger, W., Dangelmayr, C., and Schulz, S. Aspect-Oriented Parallel Discrete Optimization on the Cohesion Desktop Grid Platform. In *Proc. of the* 6^{th} *IEEE International Symposium on Cluster Computing and the Grid (CCGRID'06)*, 49–56 (Singapore, 2006).
55. Chakravarti, A. J., Baumgartner, G., and Lauria, M. The Organic Grid: Self-Organizing Computation on a Peer-to-Peer Network, May (2005).
56. Montresor, A., Meling, H., and Babaoglu, O. Messor: Load-Balancing through a Swarm of Autonomous Agents. In *Proc. of the Int. Workshop on Agents and Peer-to-Peer Computing (AP2PC 2002)*, number 2530 in LNAI, Jul. (2003).
57. Kim, J.-S., Nam, B., Marsh, M., Keleher, P., Bhattacharjee, B., Richardson, D., Wellnitz, D., and Sussman, A. Creating a robust Desktop Grid using peer-to-peer services. In *Proc. of the IEEE International Parallel and Distributed Processing Symposium (IPDPS'07)*, 1–7 (Long Beach, CA, USA, 2007).
58. Novotny, J., Russell, M., and Wehrens, O. GridSphere: A portal framework for building collaborations. *Concurrency Comput. Pract. Ex.* **16**(5), 503–513 (2004).
59. Abdelnur, A. and Hepper, S. JSR 168: Java Portlet specification version 1.0. Technical report, Sun, October (2003). Visited on Dec. 18^{th}, 2009.
60. Foster, I. and Kesselman, C. Globus: A metacomputing infrastructure toolkit. *The International Journal of Supercomputer Applications and High Performance Computing* **11**(2), 115–128 (1997).
61. Romberg, M. The UNICORE Grid infrastructure. In *Proceedings of 1st Worldwide SGI Users' Conference*, 144–153 (Krakow, Poland, 2000).
62. Laure, E., Hemmer, F., Aimar, A., Barroso, M., Buncic, P., Meglio, A. D., Guy, L., Kunszt, P., Beco, S., Pacini, F., Prelz, F., Sgaravatto, M., Edlund, A., Mulmo, O., Groep, D., Fisher, S., and Livny, M. Middleware for the next generation Grid infrastructure. In *Proc. of the* 14^{th} *International Conference on Computing in High*

Energy Physics and Nuclear Physics (CHEP 2004), Aimar, A., Harvey, J., and Knoors, N., editors, 826 (Interlaken, Switzerland, 2004).

63. Alameda, J., Christie, M., Futrelle, G. F. J., Gannon, D., Hategan, M., Kandaswamy, G., von Laszewski, G., Nacar, M. A., Pierce, M., Roberts, E., Severance, C., and Thomas, M. The Open Grid Computing Environments collaboration: portlets and services for science gateways. *Concurrency Comput. Pract. Ex.* **19**(6), 921–942 (2007).
64. von Laszewski, G., Foster, I., Gawor, J., and Lane, P. A Java Commodity Grid kit. *Concurrency Comput. Pract. Ex.* **13**(89), 643–662 (2001).
65. Dahan, M., Thomas, M., Roberts, E., Seth, A., Urban, T., Walling, D., and Boisseau, J. R. Grid Portal Toolkit 3.0 (GridPort). In *Proc of the* 13^{rd} *IEEE International Symposium on High-Performance Distributed Computing (HPDC 2004)*, 272–273 (Honolulu, Hawaii USA, 2004).
66. Fowler, M. POJO: An acronym for Plain Old Java Object. http://www.martinfowler.com/bliki/POJO.html. Visited on Dec. 18^{th}, 2009.
67. Kacsuk, P. and Sipos, G. Multi-Grid, multi-user workflows in the P-GRADE grid portal. *J. Grid Computing* **3**(3), 221–238 (2005).
68. Berlich, R., Kunze, M., and Schwarz, K. Grid computing in Europe: from research to deployment. In *Proc. of the 2005 Australasian workshop on Grid computing and e-Research (AusGrid 2005)*, volume 44 of *CRPIT*, 21–27 (Newcastle, NSW, Australia, 2005).
69. van Nieuwpoort, R. V., Kielmann, T., and Bal, H. E. User-friendly and reliable grid computing based on imperfect middleware. In *Proc. of the ACM/IEEE Conference on Supercomputing (SC'07)* (Reno, NV, 2007).
70. Frey, J. Condor DAGMan: Handling inter-job dependencies. Technical report, University of Wisconsin, (2002).
71. Kacsuk, P., Kiss, T., and Sipos, G. Solving the grid interoperability problem by P-GRADE portal at workflow level. *Future Gener. Comput. Syst.* **24**(7), 744–751 (2008).
72. Costa, F., Silva, L., Kelley, I., and Taylor, I. Peer-to-peer techniques for data distribution in Desktop Grid computing platforms. In *Proc. of the CoreGRID Workshop on Programming Models Grid and P2P System Architecture Grid Systems, Tools and Environments* (Heraklion, Crete, Greece, 2007).
73. Cohen, B. Incentives build robustness in BitTorrent. In *Proc. of the* 1^{st} *Workshop on Economics of Peer-to-Peer Systems* (Berkeley, CA, USA, 2003).
74. Wei, B., Fedak, G., and Cappello, F. Towards efficient data distribution on computational desktop grids with BitTorrent. *Future Gener. Comput. Syst.* **23**(8), 983–989 (2007).
75. Zissimos, A., Doka, K., Chazapis, A., and Koziris, N. GridTorrent: Optimizing data transfers in the Grid with collaborative sharing. In *Proc. of the* 11^{th} *Panhellenic Conference on Informatics (PCI'07)* (Patras, Greece, 2007).
76. Chun, B., Culler, D., Roscoe, T., Bavier, A., Peterson, L., Wawrzoniak, M., and Bowman, M. PlanetLab: an overlay testbed for broad-coverage services. *Comput. Comm. Rev.* **33**(3), 3–12 (2003).
77. Anglano, C. and Canonico, M. The File Mover: high-performance data transfer for the Grid: Research Articles. *Concurrency Comput. Pract. Ex.* **20**(1), 99–123 (2008).
78. Fedak, G., He, H., and Cappello, F. BitDew: a programmable environment for large-scale data management and distribution. In *Proc. of the 2008 ACM/IEEE conference on Supercomputing (SC'08)*, 1–12 (Austin, TX, USA, 2008).
79. Antoniu, G., Bougé, L., and Jan, M. JuxMem: An adaptive supportive platform for data sharing on the Grid. *Scalable Computing: Practice and Experience* **6**(3), 45–55 September (2005).
80. Vazhkudai, S. S., Ma, X., Freeh, V. W., Strickland, J. W., Tammineedi, N., and Scott, S. L. FreeLoader: Scavenging desktop storage resources for scientific data. In *Proc. of the 2005 ACM/IEEE conference on Supercomputing (SC'05)*, 56 (Seattle, WA, USA, 2005).
81. Brasileiro, F., Duarte, A., Carvalho, D., Barbera, R., and Scardaci, D. An approach for the co-existence of service and opportunistic Grids: The EELA-2 case. In *Proc. of the* 2^{nd} *Latin-American Grid Workshop (LAGrid 2008)* (Campo Grande, Brazil, 2008).