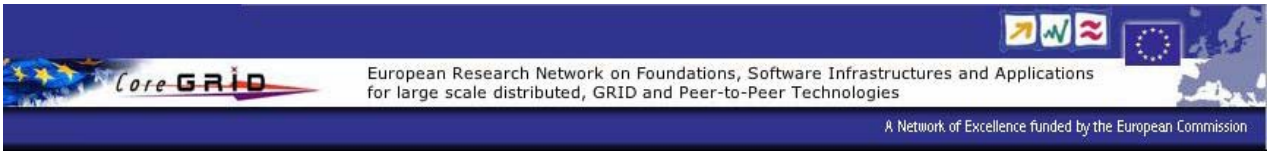

Grid Architectural Issues: State-of-the-art and Future Trends

A. Andrzejak[1], C. Mastroianni[2], P. Fragopoulou[3], D. Kondo[4], P. Malecot[5], A. Reinefeld[1], F. Schintke[1], T. Schütt[1], G.-C. Silaghi[6], L.M. Silva[6], P. Trunfio[7], D. Zeinalipour-Yazti[8,9], E. Zimeo[10]

[1]*Zuse-Institute Berlin, Takustr. 7, 14195 Berlin, Germany*
andrzejak@zib.de

[2]*ICAR-CNR Via P. Bucci 41C, 87036 Rende (CS) Italy*
mastroianni@dns.icar.cnr.it

[3]*Foundation for Research and Technology-Hellas, Institute of Computer Science*
N. Plastira 100, Vassilika Vouton, 71 113 Heraklion, Crete, Greece
fragopou@ics.forth.gr

[4]*Laboratoire LIG, ENSIMAG - antenne de Montbonnot, ZIRST 51, avenue Jean Kuntzmann*
38330 MONBONNOT SAINT MARTIN, France
dkondo@imag.fr

[5]*Laboratoire LRI, Bâtiment 490, Université de Paris-Sud, 91405 ORSAY Cedex, FRANCE*
paul.malecot@lri.fr

[6]*CISUC - Centre for Informatics and Systems of the University of Coimbra, 3030-290 Coimbra, Portugal*
luis@dei.uc.pt, gsilaghi@dei.uc.pt

[7]*DEIS – University of Calabria Via P. Bucci 41C, 87036 Rende (CS) Italy*
trunfio@deis.unical.it

[8]*Department of Computer Science, University of Cyprus, P.O. Box 20537, CY-1678 Nicosia, Cyprus*
[9]*School of Pure and Applied Sciences, Open University of Cyprus, P.O. Box 24801, CY-1304, Nicosia, Cyprus*
zeinalipour@ouc.ac.cy

[10]*Department of Engineering, University of Sannio, Benevento, Italy*
zimeo@unisannio.it

CoreGRID White Paper
Number WHP-0004
May 30, 2008

Institute on Architectural Issues: Scalability, Dependability, Adaptability

CoreGRID - Network of Excellence
URL: http://www.coregrid.net

CoreGRID is a Network of Excellence funded by the European Commission under the Sixth Framework Programme
Project no. FP6-00426

Grid Architectural Issues: State-of-the-art and Future Trends

A. Andrzejak[1], C. Mastroianni[2], P. Fragopoulou[3,11], D. Kondo[4], P. Malecot[5], A. Reinefeld[1], F. Schintke[1], T. Schütt[1], G.-C. Silaghi[6], L.M. Silva[6], P. Trunfio[7], D. Zeinalipour-Yazti[8,9], E. Zimeo[10]

[1]*Zuse-Institute Berlin, Takustr. 7, 14195 Berlin, Germany*
andrzejak@zib.de

[2]*ICAR-CNR Via P. Bucci 41C, 87036 Rende (CS) Italy*
mastroianni@dns.icar.cnr.it

[3]*Foundation for Research and Technology-Hellas, Institute of Computer Science*
N. Plastira 100, Vassilika Vouton, 71 113 Heraklion, Crete, Greece
fragopou@ics.forth.gr

[4]*Laboratoire LIG, ENSIMAG - antenne de Montbonnot, ZIRST 51, avenue Jean Kuntzmann*
38330 MONBONNOT SAINT MARTIN, France
dkondo@imag.fr

[5]*Laboratoire LRI, Bâtiment 490, Université de Paris-Sud, 91405 ORSAY Cedex, FRANCE*
paul.malecot@lri.fr

[6]*CISUC - Centre for Informatics and Systems of the University of Coimbra, 3030-290 Coimbra, Portugal*
luis@dei.uc.pt, gsilaghi@dei.uc.pt

[7]*DEIS – University of Calabria Via P. Bucci 41C, 87036 Rende (CS) Italy*
trunfio@deis.unical.it

[8]*Department of Computer Science, University of Cyprus, P.O. Box 20537, CY-1678 Nicosia, Cyprus*
[9]*School of Pure and Applied Sciences, Open University of Cyprus, P.O. Box 24801, CY-1304, Nicosia, Cyprus*
zeinalipour@ouc.ac.cy

[10]*Department of Engineering, University of Sannio, Benevento, Italy*
zimeo@unisannio.it

CoreGRID WHP-00xx
May 30, 2008

Abstract

Grid architecture is one of the cornerstones for successful development and proliferation of Grid computing. The scale, dynamism and openness of the Grid, together with demands on its reliability, security and manageability, pose unique challenges on software architecture. The main objective of the Architectural Issues Institute of the CoreGRID NoE is to perform a significant improvement of architectural designs of future Grids by focusing on three particular key aspects: scalability of resources, adaptability, and dependability of Grid architectures and services. These research directions address the mandatory architectural properties of the Next Generation Grids as identified by the NGG reports: simplicity, resilience, scalability of services, and straightforward administration and configuration management.

This paper presents the current state-of-the-art on the research topics of the partners involved in the Architectural Issues Institute of the CoreGRID NoE, with special focus on scalable resource discovery, fault tolerance for Grid systems, and adaptability and performance predictions mechanisms for a self-manageable Grid infrastructure. A newly emerged area of research, the one of large-scale volunteer computing using desktop Grid platforms constitute an active area of research in this Institute. A significant problem is to render these platforms resilient to open up to commercial applications. Special focus is given to the future research trends on these topics as they emerge from the involvement of the CoreGRID partners.

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

[11] With the Department of Applied Informatics and Multimedia, Technological Educational Institute of Crete, Greece.

1 Introduction

Grid architecture is one of the cornerstones for successful research, development and proliferation of Grid computing. The scale, dynamism and openness of the Grid, together with demands on its reliability, security and manageability, pose unique challenges on software architecture. Grid systems are composed from many components which number and diversity increases rapidly over time. The main objective of the Architectural Issues Institute of the CoreGRID NoE is to perform a significant improvement of architectural designs of future Grids by focusing on three particular key aspects: scalability, adaptability, and dependability of Grid architectures and Grid services. These research directions address the mandatory architectural properties of the Next Generation Grids as identified by the NGG reports: simplicity, resilience, scalability of services, and straightforward administration and configuration management. Next Generation Grids will be open, large-scale, pervasive and heterogeneous, and will have to deal with diverse types of resources. Yet, in order to exploit their full potential, Grids have to be simple, transparent, reliable, persistent, secure, and easily configurable and manageable. This is clearly a novel combination of demands on software architecture, and guiding principles for building such systems are not known.

In recent years we witnessed the development of Service Grid infrastructures, like the EGEE in Europe and the TeraGrid in USA. Bringing together PetaFlops of computing power, service Grid systems interconnect computational resources available under different administrative domains and operate under strict administrative control. On the other side of the spectrum, Desktop Grids utilize the free resources available on the Intranet and the Internet to support large-scale applications and storage. Although desktop Grids sustain immense computational power it still remains a major challenge to fully exploit the offered computational power due to the high-volatility and extreme heterogeneity of resources.

Despite the astonishing growth of large-scale architectures and platform design, there is still a long way to go before large-scale distributed computing platforms become available to the industry for commercial applications demanding, not only best effort behaviour, but also a flexible, dependable, trustworthy, and autonomic, self-manageable environment, able to meet the needs of today's business demands in a highly competitive society. The ultimate challenge we will have to face in the immediate future is to equip Grid infrastructure with those components that will allow their usage not only in academic and non commercial applications, but also in business applications, thus becoming valuable to the industry community. This opening will demolish the barrier between the academic and business communities and will open up large-scale distributed computing platforms in a whole new world of applications.

With the emergence of Cloud computing, several of the techniques that have been developed in the context of Grid systems will find immediate use. Cloud computing aims to exploit the resources of large data centres (service provision, computational power, storage, bandwidth) for business applications through resource leasing. This could greatly benefit small and medium size service providers that need to temporarily increase their resources. The notion of resource leasing employed by Clouds leverages the need for advanced virtualization techniques, and appropriate Service Level Agreements in order to guarantee certain QoS requirements. Some of the Cloud computing infrastructures (i.e. Amazon's Elastic Grid) target computation rather than service provision, which is also the main focus of the techniques presented in this white paper.

It will not be long before Cloud computing expands its application domain to commodity PCs in order to take advantage of the abundance of available resources they posses, mostly in terms of computation, available storage, and bandwidth. Rendering desktop PC platforms reliable, and adapting them to exploit their capabilities while overcoming their shortages will open up their use in various application scenarios related to Cloud computing, resource leasing, and use by appropriate service providers. In order to effectively employ these techniques, methods for scalable, dependable, and adaptable large-scale computing infrastructures like the ones elaborated in this paper will be of great use. Techniques for scalable resource discovery, possibly using the P2P approach, dependability for desktop Grids, or techniques for distributed storage and replication, survivability of checkpoints of computations in volatile computing environments are vital to secure dependability of large-scale computing infrastructures based on commodity hardware. Furthermore, adaptability for dynamic relocation of jobs is also very important. This paper addresses some crucial issues in relation to vital Grid architectural issues, indispensable for the emergence of critical future applications of large-scale distributed computing systems.

This paper presents the current state-of-the-art in the aforementioned areas with special focus on open topics of research and future trends. The remaining of the paper is organized as follows: Following the introduction in Section 1, Section 2 presents the scalable resource discovery topic using the peer-to-peer approach in Grid systems. Section 3 presents fault-tolerance issues in service Grids. Section 4 presents desktop Grids for large-scale volunteer computing a topic that gained considerable attention during the four years of research activity in the Architectural issues Institute. The various types of desktops Grid platforms and their common characteristics are presented. Section 5 elaborates on dependability mechanisms for desktop Grid computing, an area of significant interest. Several open research topics are presented such as validity of computational results, accountability of resources used by

computations, as well as trust and security issues. In Section 6, another topic of major importance is presented, related to the adaptability of Grids in an effort to achieve performance prediction techniques and self-management of Grid infrastructure. Special attention is paid to the feedback loop approach. Finally, in section 7, a note on service oriented computing is presented in relation to the activities of the Institute. We conclude in Section 8.

2 Scalable P2P-based Resource Discovery for Grid Systems

The goal of resource discovery in Grids is to provide the way for locating resources of interest, available across Grid nodes, on the basis of users and systems requirements as needed for the execution of distributed applications. In most Grid systems, resource discovery is implemented according to centralized or hierarchical approaches, mostly because the client/server approach is still used today in the largest part of distributed systems and in Web service-based frameworks. However, a hierarchical resource discovery service is viable within a single organization or in a small-scale Grid, but it becomes impractical in a large multi-institutional Grid for several reasons, among which:

- scalability is limited because a significant amount of memory space must be reserved at the tree root node to maintain information about a large number of resources;
- information servers belonging to different levels of the hierarchy must carry very different computation and traffic loads, which leads to challenging problems concerning load imbalance;
- the hierarchical organization can hinder the autonomous administration of different organizations;
- fault-tolerance is limited by the presence of a single point of failure at the root of the hierarchy.

In the last years, the Peer-to-Peer (P2P) paradigm emerged as an alternative to centralized and hierarchical approaches for implementing scalable and reliable resource discovery in Grid systems. P2P systems use diverse connectivity between participating nodes and the cumulative bandwidth of network participants rather than conventional centralized systems where a relatively low number of servers provide the core value to a service or application. Several P2P-based Grid resource discovery systems have been proposed so far (for a survey, see [1]). Such systems are generally classified into *unstructured* and *structured*, based on the way nodes are linked to each other and data about resources is placed on nodes.

In unstructured systems, links among nodes can be established arbitrarily and data placement is unrelated from the topology of the resulting overlay. In such systems, when a node wishes to find a given resource, the query must be distributed through the network using flooding-like techniques to reach as many nodes as needed. Each node reached by the query processes it on the local data items and, in case of match, replies to the query initiator. Some examples of Grid resource discovery systems based on unstructured P2P approaches are presented in [2–5]. Structured systems strictly control both overlay topology and data placement. Most of such systems use a Distributed Hash Table (DHT). Each node and each piece of data is mapped through a distributed hash function. Thus, each node is assigned the responsibility for a specific part of the resources in the network. When a node wishes to find a resource with a given identifier, such systems are able to locate the node responsible for that identifier typically in *O(logN)* hops using only *O(logN)* neighbors per node and can scale to extremely large numbers of nodes. Some examples of structured P2P-based Grid resource discovery systems are described in [6–10].

Structured systems are generally more efficient than unstructured systems in terms of search time and number of messages. Compared to unstructured systems, however, structured systems provide a limited support to complex queries. Although several extensions to basic DHT schemes have been proposed to allow, for instance, range queries [6] and multi-attribute search [7], DHT-based lookups still do not support arbitrary queries (e.g., regular expressions [11]) since it is infeasible to generate and store keys for every query expression. On the other hand, unstructured systems can do it effortlessly since all queries are processed locally on a node-by-node basis [12]. In the context of CoreGRID, several researchers worked with the aim of designing and implementing P2P systems for scalable resource discovery in Grid environments.

A first result of this work was the design of an unstructured Grid resource discovery system based on the *super-peer* approach [13]. In super-peer systems, each super-peer node acts as a centralized server for a number of regular peers, while super-peers connect to each other to form a flat P2P network at a higher level. The proposed system defines how the super-peer model is exploited to manage membership management (i.e., the problem of adding a new node to the network, and assigning this node a set of neighboring nodes) and resource discovery in a multi-organizational Grid. Simulation results showed that the super-peer model is naturally appropriate to the organization-based nature of Grids, ensuring limited network load and reduced response time with respect to pure-decentralized P2P systems.

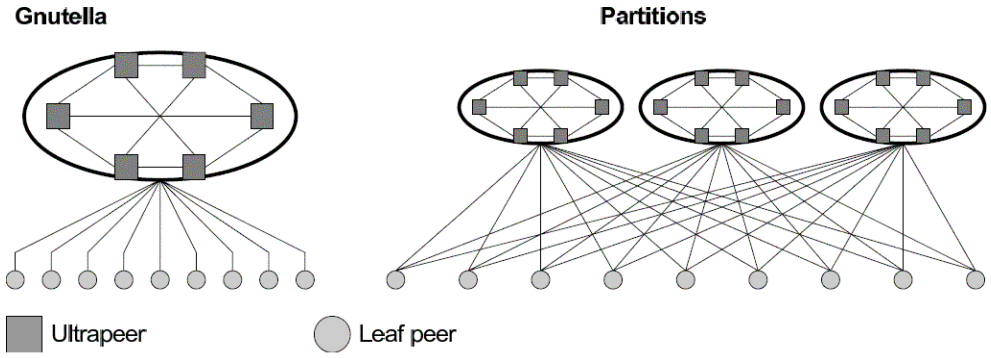

Figure 1: Illustration of the Gnutella and the Divide-et-Impera architectures.

The *Divide-et-Impera* scheme proposed in [14] is also an unstructured system, Fig. 1. The aim of such system is to reduce the overwhelming volume of traffic generated by flooding, thus increasing the scalability of unstructured P2P systems. Using a simple hash-based content categorization method the overlay network is partitioned into a relatively small number of distinct subnetworks. By employing an index splitting technique each peer is effectively connected to each different subnetwork. The search space of each individual flooding is restricted to a single partition, and is thus considerably limited. This reduces significantly the volume of traffic produced by flooding without affecting at all the accuracy of the search method. Experimental results demonstrate the efficiency of the proposed method.

Another proposal aimed to improve the scalability of flooding in unstructured systems is the *feedback-based approach* presented in [15]. The main idea behind this algorithm is to monitor the ratio of duplicate messages transmitted over each network connection, and not forward query messages over connections whose ratio exceeds some predefined threshold. Simulation results show that this algorithm exhibits significant reduction of traffic in random and small-world graphs, the two most common types of graph that have been studied in the context of P2P systems, while conserving network coverage.

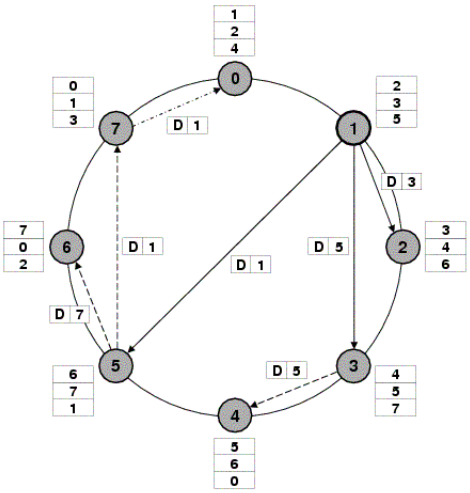

Figure 2: Illustration of the DQ-DHT algorithm.

The *DQ-DHT* algorithm proposed in [16] is designed to support *dynamic querying*-like searches in DHT-based structured networks, Fig. 2. Dynamic querying is a technique adopted in unstructured P2P networks to minimize the number of nodes that is necessary to visit to reach the desired number of results. DQ-DHT is based on a combination of the dynamic querying technique with an algorithm for efficient broadcast over a DHT overlay. The aim of DQ-DHT is two-fold: 1) allows to perform arbitrary queries in structured P2P networks; 2) provides dynamic adaptation of the search according to the desired number of results and the popularity of resources to be located. The original DQ-DHT algorithm has been implemented using Chord [17] as basic overlay. Recently, an extension of DQ-DHT has

been proposed to work in *k*-ary DHT-based overlays [18]. In a *k*-ary DHT, broadcast takes only $O(\log_k N)$ hops using $O(\log_k N)$ pointers per node. This "*k*-ary principle" has been exploited in DQ-DHT to improve the search time with respect to the original Chord-based implementation.

The main application that guided the design of DQ-DHT is the implementation of a resource discovery system for a large-scale Grid environment [19]. The system is based on a super-peer architecture, in which super-peer nodes are linked together to form a DHT overlay. Such DHT overlay is used to perform both key-based searches, as in structured systems, and dynamic querying-like searches, as in unstructured systems, this way combining the advantages of both models. A prototype of such system has been implemented using OpenChord [20] and deployed on the Grid'5000 platform [21] for evaluation. The experimental performance results demonstrated the efficiency of the system both in terms of network traffic and search time, showing its effectiveness as an alternative to centralized and hierarchical architectures.

Antares [22] is a bio-inspired algorithm that exploits ant-like agents to build a P2P information system in Grids. The work of agents is tailored to the controlled replication and relocation of metadata documents that describe Grid resources. These descriptors are indexed through binary strings that are the result of the application of a locality preserving hash function that maps similar resources into similar keys. Agents travel the Grid through P2P interconnections and copy and move descriptors so as to locate descriptors represented by identical or similar keys into neighbor Grid hosts. The resulting information system is referred to as *self-structured*, because it exploits the self-organizing characteristics of ant-inspired agents, and the association of descriptors to hosts is not pre-determined but easily adapts to the varying conditions of the Grid. This self-structured organization aims to combine the benefits of both *unstructured* and *structured* P2P information systems. Indeed, being basically unstructured, Antares is easy to maintain in a dynamic Grid, in which joins and departs of hosts can be frequent events. On the other hand, the aggregation and spatial ordering of descriptors can improve the rapidity and effectiveness of discovery operations, and also enables range queries, which are beneficial features typical of structured systems.

Most Grid platforms are currently based on a service-oriented model, namely the Open Grid Services Architecture (OGSA), in which a common representation for both real resources (processors, memories, etc.) and logical resources is adopted. All these resources are treated as *services*: network-enabled entities that provide some capabilities though the exchange of messages. Due to this service-oriented view, resource discovery and service discovery in service-oriented Grid platforms, such as Globus Toolkit 4, are generally provided through a common framework which addresses the need for standard interface definition mechanisms and uniform service semantics. In this service-oriented scenario, it is fundamental to devise strategies and mechanisms to drive discovery based on metadata and semantic annotations through which services are described [1]. Adding semantic information to resource descriptions also allows improving precision of a discovery service that should be able to find best approximate matches usable for the requester as it is unrealistic to expect requested and offered services to be exactly identical. Using semantic information for precise resource discovery in large-scale, dynamic and heterogeneous environments is a relatively new and fragmented research topic. We believe that more studies should be devoted to comparing relative merits of proposed approaches and architectures. Even though the use of semantic information in resource discovery is very important for interoperability, it raises a problem of semantic interoperability, i.e. it requires using common ontologies in service descriptions in order to reach semantic agreement. Incorporating semantic annotations to resources is a step towards the SOKU model.

3 Fault-tolerance and Robustness for Grid Systems

One of the main challenges in realizing the full potential of Grids is making these systems fault tolerant, robust and dependable. As a measure of dependability of Grids we usually refer to the ratio of successfully fulfilled job requests over the total number of jobs submitted to the resource brokers of a Grid infrastructure. The FlexX and Autodock data challenges of the WISDOM [48] project, conducted in August 2005, have shown that only 32% and 57% of the jobs completed successfully (with an "OK" status). Additionally, the work in [28] has shown that only 48% of the jobs submitted to a South-Eastern-Europe resource broker have completed successfully. Consequently, the dependability of large-scale Grids needs to be improved substantially.

In this section we start out by identifying the causes of failures in Grid Systems, provide an overview of Grid Monitoring and Ranking systems which aim to proactively identify failures and finally provide an elaborate overview of fault injection systems that have been proposed in the literature.

3.1. Causes of Failures in Grid Systems

In this section we identify the main causes of failures in Grid infrastructures. Our observations are extrapolated from other research works, in particular [48, 35, 28].

- **Hardware faults:** if the job is running on the specified machine at the time of an unrecoverable hardware error, e.g. a hard drive burns (most common), RAM or motherboard failures, power supply failure, etc.
- **O/S misconfiguration:** this relates mainly to operating system services that are not properly configured. One common example is implementing firewall changes on a site. This can lead to closing ports that are needed for site inter-node communication, or blocking site hosts from communicating with each other altogether. The Grid Gate may lose communication with a worker node (WN) running a user job or a WN may lose communication with the site Storage Element (or with another site's Storage Element) and be unable to make necessary data transfers. A closely related issue is the inability to run MPI [43] jobs when 'inter-worker-node' communication is blocked.
- **Network access disruption/misconfiguration:** a factor that can lead to job failure (or more accurately in this case, leading to CPU time being wasted), is a site losing Internet access. The firewall example mentioned above also applies to network misconfiguration, if the changes are implemented on the network 'perimetric' firewall. A user waiting for too long to retrieve the job output may decide to resubmit the job, rendering the previous one useless when the site recovers from network access failure, if we assume that the previous job was still running while the network was down.
- **Security breaches/attacks:** Computing Element, Storage Element or Resource Broker takeover by an unauthorized user (commonly known as a 'cracker') can result to malicious acts like corruption of job data, job termination, sandbox deletion etc. Such attacks are usually related to security holes of the operating system and Grid middleware, weak root passwords and inappropriate firewall configuration. Denial of Service (DoS) attacks may also disrupt job completion or temporarily prevent access to job output by cutting a site off the Resource Broker that has delegated the job and is waiting for the output. Note that discussing actual security incidents related to the EGEE infrastructure is not possible under most circumstances, since this could provide potential attackers with useful information.
- **Middleware misconfiguration:** attempts to correct problems or perform updates on a Grid site can lead to job failure or a more general service disruption. This relates to Grid site administrator errors, as well as to bugs in new releases of the middleware that introduce unwanted configuration. According to [24], a large number of service disruption occurrences is the direct result of a regular performance or security software upgrade that leads to configuration errors. Some examples of misconfiguration: setting too short a wallclock time for a job queue and as a result jobs die before completion; publishing wrong resource data and matchmaking results in accepting a job while no compatible resources exist to satisfy it, causing bandwidth loss and adding overhead to the overall time needed for serving the user; killing the wrong job by issuing a scheduler or resource manager command (as root on the Grid Gate node).
- **Middleware bugs:** Grid job failures can result from bugs in middleware code; for instance, failures relating to the Grid Workload Management System (WMS), including the components residing on the Resource Broker and the submitting User Interface nodes, observed in [48]. Further information can be given on this particular type of failures once the appropriate experiments are conducted on the EGEE Grid infrastructure and the aborted jobs are analysed.
- **User mistakes:** such failures can result from (a) JDL file problems, for example the user may include an inaccurate specification of job requirements that will result in the job failing to start; (b) user software can cause errors during job execution leading to the job being terminated abnormally; and (c) problems with user certificate proxies attached to the job, most commonly the absence of a valid proxy during submission, as well as the expiration of an originally valid proxy while the job is running. All the cases mentioned here are under user control and have nothing to do with the malfunctioning of other components of the system, so corrective action can only be assumed on part of the user, and not by any form of automatic job resubmission mechanisms.

3.2 Failure Monitoring and Ranking

Detecting and managing failures is an important step toward the goal of a dependable Grid. Currently, this is an extremely complex task that relies on over-provisioning of resources, ad-hoc monitoring and user intervention. Adapting ideas from other contexts such as cluster computing [38], Internet services [36, 37] and software systems [39] seems also difficult due to the intrinsic characteristics of Grid environments. Firstly, a Grid system is not administered centrally; thus it is hard to access the remote sites in order to monitor failures. Moreover we cannot

easily encapsulate failure feedback mechanisms in the application logic of each individual Grid software component, as the Grid is an amalgam of pre-existing software libraries, services and components with no centralized control. Secondly, these systems are extremely large; thus, it is difficult to acquire and analyze failure feedback at a fine granularity. Lastly, identifying the overall state of the system and excluding the sites with the highest potential for causing failures from the job scheduling process, can be much more efficient than identifying many individual failures. In order to efficiently monitor the state of a Grid system the community has recently suggested the deployment of Meta-Monitoring Systems and Active Benchmarking Systems.

- **Meta-Monitoring Systems:** Several methods for detecting failures have been deployed so far (for a detailed description see [42]). Examples include: (i) Information Index Queries: these are performed on the Information Service and enable the extraction of fine-grained information regarding the complete status of a Grid site; (ii) Service Availability Monitoring (SAM) [49]: a reporting web site that is maintained for publishing periodic test-job results for all sites of the infrastructure; (iii) Grid statistics: provided by services such as GStat [33]; (iv) Network Tomography Data: these can be obtained by actively pinging and tracerouting other hosts in order to obtain delay, loss and topological structure information. Network tomography enables the extraction of network-related failures; (v) Global Grid User Support (GGUS) ticketing system [31]: system administrators use this system to report component failures as well as needed updates for sites. Such tickets are typically opened due to errors appearing in the SAM reports; (vi) Core Infrastructure Center (CIC) broadcasts [27]: allow site managers to report site downtime events to all affected parties through a web-based interface; and (vii) Machine log-files: administrators can use these files to extract error information that is automatically maintained by each Grid node.
- **Active Benchmarking Systems:** Deploying a number of lower level probes to the remote sites is another direction towards the extraction of meaningful failure semantics. In particular, one can utilize tools such as GridBench [41, 47], the Grid Assessment Probes [26] and DiPerF [30], in order to determine in real time the value of certain low level and application-level failure semantics that can not be furnished by the meta-information sources. For example, GridBench provides a corpus of over 20 benchmarks that can be used to evaluate and rank the performance of Grid sites and individual Grid nodes.

Both the Meta-Monitoring Systems and the Active Benchmarking Systems have a major drawback: their operation relies heavily on human intervention. As Grid infrastructures become larger, human intervention becomes less feasible and efficient.

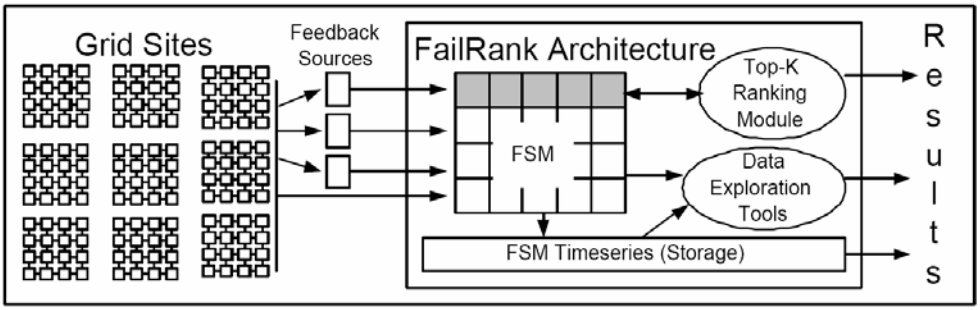

Figure 3: The Failrank architecture.

In the FailRank architecture [51,52], seen in Fig. 3, feedback sources (i.e., websites, representative low-level measurements, data from the Information Index, etc.) are continuously coalesced into a representative array of numeric vectors, the FailShot Matrix (FSM). FSM is then continuously ranked in order to identify the K sites with the highest potential to feature some failure. This allows the system to automatically exclude the respective sites from the job scheduling process. The advantages of the approach are summarized as follows: (i) it is a simple yet powerful framework to integrate and quantify the multi-dimensional parameters that affect failures in a Grid system; (ii) is is tunable, allowing system administrators to drive the ranking process through user-defined ranking functions; (iii) it eliminates the need for human intervention, thus the approach gives space for automated exploitation of the extracted failure semantics; (iv) it can be implemented as a filter outside the Grid job scheduler, thus imposing minimum changes to the Grid infrastructure.

3.3 Fault-Injection and Dependability Benchmarking

Another direction towards achieving a fault-tolerant Grid is by conducting fault injection and dependability benchmarking [46, 44]. In particular, given that Grid systems support a variety of fault-tolerance mechanisms, like replication, software rejuvenation, component-based reconfiguration, failure-detection, checkpointing-recovery and others, the objective is to synthetically add faults in the system operation and then experimentally assess the dependability metrics of the target system through some automated system. We next provide an overview of such fault injection tools.

- **CECIUM [23]:** is a testing environment that simulates a distributed application and fault injections. The distributed execution is thus simulated on a unique machine with a unique addressing space. The source code of the tested application is not required.
- **DOCTOR [34]:** is a system that permits to inject faults in real time systems. It is capable to inject faults and to synthesize the workload by supporting three kinds of faults: processor faults, memory faults and communication faults. The injected faults can be permanent, transient, or intermittent. DOCTOR has been implemented in the HEART real-time system.
- **ORCHESTRA [29]:** is a fault injection tool that allows the user to test the reliability and the liveliness of distributed protocols. A fault injection layer is inserted between the the tested protocol layer and the lower layers, and allows to filter and manipulate messages exchanged between the protocol participants. Messages can be delayed, lost, reordered, duplicated, modified and new messages can be spontaneously introduced into the tested system to bring it into a particular global state.
- **NFTAPE [45]:** arose from the double observation that no tool is sufficient to inject all fault models and that it is difficult to port a particular tool to different systems. NFTAPE [28] provides mechanisms for fault-injection, triggering injections, producing workloads, detecting errors, and logging results. Unlike other tools, NFTAPE separates these components so that the user can create his own fault injectors and injection triggers using the provided interfaces. NFTAPE introduces the notion of Lightweight Fault Injector (LWFI). LWFIs are simpler than traditional fault injectors, because they do not need to integrate triggers, logging mechanisms, and communication support. This way, NFTAPE can inject faults using any fault injection method and any fault model. Interfaces for the other components are also defined to facilitate portability to new systems.
- **LOKI [25]:** is a fault injector dedicated to distributed systems that is based on a partial view of the global state of the distributed system. The faults are injected based on a global state of the system. An analysis is executed at the end of the test to infer a global schedule from the various partial views and then verify if faults were correctly injected (i.e. according to the planned scenario). Normally, injecting faults based on a global state of a distributed application leads to a high impact on its execution time. However, the technique used by LOKI is used to verify the validity of the injected faults while limiting their impact on the execution time

3.4 Challenges of Realizing Fault Tolerance in Grid Systems

In this section we identify some future challenges in realizing efficient fault tolerance mechanisms in a Grid environment.

- **Failure Exchange Interfaces:** The first challenge is to develop efficient interfaces and protocols to exchange fault information between Grid sites. The development of such protocols is currently difficult as the lack of a centralized authentication and administration scheme makes it intrinsically difficult to access the remote sites and monitor failures. Furthermore, it is currently also very hard to encapsulate failure feedback mechanisms in the application logic of individual Grid software as the Grid is an amalgam of pre-existing software libraries, services and components with no centralized control. What is required is a generic component that can be statically or dynamically linked to the software stack of Grid software and which can enable the communication of relevant data through a common interface.
- **Failure Information Schema & Granularity:** The second challenge is to develop a global schema that will define the nature of information to be exchanged. The lack of common parameters that characterize failures makes it too difficult to obtain a global understanding and advance fault tolerance. For instance, a given monitoring system might count I/O reads and writes, defined as the distinct number of I/O operations performed, while another Grid monitoring system might count I/O bytes read and write, defined as the accumulative number of bytes that were spent on the given I/O operations. Additionally, it also remains to be shown what the right granularity of failure feedback is such that this can be utilized in the prognosis of failures. It is known that large content distribution networks collect low-level probes (e.g., ping and trace

route data) in order to enable a variety of network tomography operations. Although such probes are essential in the establishment of these services, they incur enormous network traffic. In the context of Grid Computing, it is still not shown that such low-level information is efficient, practical and that it can be obtained in a viable fashion.

4 Desktop Grids and Volunteer Computing

Desktop Grids utilize the free resources available in Intranet or Internet environments for supporting large-scale computation and storage. For over a decade, Desktop Grids have been one of the largest and most powerful distributed computing systems in the world, offering a high return on investment for applications from a wide range of scientific domains (including computational biology, climate prediction, and high-energy physics). While desktop Grids sustain up to PetaFLOPS of computing power from hundreds of thousands to millions of resources, fully leveraging the platform's computational power is still a major challenge because of the immense scale, high volatility, and extreme heterogeneity of such systems. In the sections below, we describe the state-of-the-art in meeting such challenges.

We classify Desktop Grids in different groups according to how resources are distributed. We will start from the Enterprise Desktop Grid that is used in only one administrative domain, and end with the Internet Volunteer-based Desktop Grid. We start with a general description of common features found in desktop Grids.

4.1 General Characteristics of Desktop Grids

A Desktop Grid is composed of software and an infrastructure of ordinary PCs which are shared with their owner. The network is a regular IP based network ranging from an enterprise LAN to the Internet. Here we will list features and design goals of those Grids then we will describe the general process for running a computation on those systems.

Desktop Grids architects have defined several desirable properties for their systems. Depending on their research topic, some systems' authors choose to emphasize some of the properties in their software. Some of the systems that will be described in this document may not implement all of those features. Here is a short list of those properties:

- **Scalability:** Desktop Grids must scale to a huge number of nodes. The scale will depend mostly on the number of expected *computing nodes*. An enterprise Grid will gather less computing nodes than a volunteer desktop Grid.
- **Heterogeneity:** Resources may vary in processor architecture, available memory, operating systems, network connectivity and many other hardware characteristics. Platforms must accommodate this heterogeneity. Manually porting the applications to all available platforms should be avoided.
- **Connectivity:** The systems have to provide connectivity for all resources to the desktop Grid, even if those resources are behind a firewall or in private address space. Systems may provide mechanisms for storing messages while nodes are off-line.
- **Volatility:** Available nodes and network connectivity vary quickly and incessantly. Network bandwidth and latency also vary. Computational parallelism should adapt when platforms grow or shrink.
- **Fault Tolerance:** Increasing the number of participating nodes and, in the case of volunteer computing, using nodes that are not professionally set up, also increases the number of potential faults that may occur. A failure or loss of one server, client or computing node must have no impact on the correct ending of the computation. Due to the size of desktop Grids, faults are very frequent.
- **Unobstructiveness:** Desktop Grid computing nodes are also used by their users who usually want their own program to run with priority greater than Grid applications. Desktop Grid must limit resource usage on computing nodes (including CPU, memory, storage, network usage). Some desktop Grid work focuses on *cycle stealing*. Nodes are monitored for free resources, and an application is run when the owner is not using them.
- **Safety:** The Internet is well known for not being safe. Servers, clients and computing nodes must be protected from all types of attacks. Especially, local resources (there applications and data) should be protected from attacks coming from the running application. This is often done by running the software in a virtual machine or in a sandbox.
- **Correctness:** Results obtained from desktop Grid should be validated. The computing node may produce wrong results or its owner may be cheating.
- **Ease of use:** Especially in volunteer desktop Grid, the software should be easy to use in order to be widely deployed.
- **Performances:** Performance of the system should be reasonable.

- **Anonymity:** If needed and accepted by the computing node, sensitive and proprietary data shouldn't be usable by the computing node.
- **Hierarchy:** The Grid may exploit the existing infrastructure (network infrastructure, administrative domains).
- **Rewarding:** In desktop Grid, computational resources may be shared by different people. Volunteers may want to advertise the amount of resources they offer. (They get credits for the work they compute.) Other users may want to charge for their resources or get favored for running their own work when they will need to.

In the following sections, we will describe how this general strategy is done in the different types of desktop Grids.

4.2 Enterprise Desktop Grids

Enterprise Desktop which consists of volatile hosts within a LAN. LANs are often found within a corporation or university, and several companies such as Entropia and United Devices have specifically targeted these LANs as a platform for supporting desktop Grid applications. Enterprise desktop Grids are an attractive platform for large scale computation because the hosts usually have better connectivity with 100Mbps Ethernet for example, and have relatively less volatility and heterogeneity than desktop Grids that span the entire Internet. Nevertheless, compared to dedicated clusters, enterprise Grids are volatile and heterogeneous platforms, and so the main challenge is then to develop fault-tolerant, scalable, and efficient scheduling.

XtremWeb [59,61,62] is an open source research project at LRI and LAL that belongs to a class of light-weight Grid systems. It is primarily designed to explore scientific and research issues about Desktop Grid, Global Computing and Peer-to-Peer distributed systems but has been also used to execute real computations, especially in physics. The first version was released in 2001.

XtremWeb also belongs to the Cycle Stealing Environment family. Various *Activators* (for CPU activity, keyboard and mouse, work hours) are available to decide when cycles can be stolen. The platform is written mostly in Java and can run applications written in Java or that use the native platform architecture. (Versions for Linux, MacOS X and Windows are available.) A sandbox is available for isolating the host system from both types of applications.

The architecture is similar to most well-known platforms. It is a three-tier architecture with clients, servers and workers. Several instances of those components might be used at the same time. Clients allow platform's users to interact with the platform by submitting stand-alone jobs, retrieving results and managing the platform. Workers are responsible for executing jobs. The server is a coordination service that connects clients and workers. The server accepts tasks from clients, distributes them to workers according to the scheduling policy, provides applications for running them and supervises the execution by detecting worker crash or disconnection. If needed tasks are restarted on other available workers. At the end, it retrieves and stores results before clients download them.

Clients and Workers are the initiators of all connections to the server . Thus only the server needs to be accessible from behind firewalls. Multiples protocols are supported and can be used depending on the type of workload. Communication may also be secured both by encryption and authentication.

Since its first version, XtremWeb has been deployed over networks of common Desktop PCs providing an efficient and cost effective solution for a wide range of application domains: bioinformatics, molecular synthesis, high energy physics, numerical analysis and many more. At the same time, there has been much research around XtremWeb: XtremWeb-CH, [55,56] funded by the University of Applied Sciences in Geneva, is an enrichment of XtremWeb in order to better match P2P concepts. Communication is distributed, i.e. direct communications between workers are possible. It provides a distributed scheduler that takes into account the heterogeneity and volatility of workers. There is an automatic detection of the optimal granularity according to the number of available workers and scheduling tasks. There is also a monitoring tool for visualizing the executions of the applications.

4.3 Collaborative Desktop Grids

Collaborative Desktop Grids consist of several Enterprise Desktop Grids which agree to aggregate their resources for a common goal. The OurGrid project [60,54] is a typical example of such systems. It proposes a mechanism for laboratories to put together their local Desktop Grids. A mechanism allows the local resource managers to construct a P2P network. These solutions are attractive because utilization of computing power by scientists is usually not constant. When scientists need an extra computing power, this setup allows them to access easily other university resources. In exchange, when their resources are idle, it can be given or rented to other university. This requires a cooperation of the local Desktop Grid systems, usually at the resources manager's level, and mechanisms to schedule

several applications. A similar approach has been proposed by the Condor team under the term “flock of condor” [63].

OurGrid has been in production since December 2004 and today aggregates computing resources from about 180 nodes shared by 12 peers. The platform has been limited to supporting bag-of-tasks. Local users have always the priority for their tasks on their local resources, only the unused local resources are shared with other peers. Local jobs kill remote jobs if needed. For promoting cooperation among peers, OurGrid use a *network of favors*. Each peer maintains a matrix of the computing time that it gets granted from other peers. Then, if a processor is requested by more than one peer, it allocates it to the peer with the greatest favor. The favor computation is protected against malicious peers that, for example, would reset its state in order to gain more computing time.

Other peers are discovered through a centralized discovery system. The network is a free-to-join Grid, so remote peers are untrusted. To address this issue, a sandbox mechanism is proposed (Sand-boxing Without A Name). It is built using Xen. Access to local network is also disabled by this sandbox.

Several scheduling policies have been experimented. The first one, Workqueue with Replication (WQR), was simply sending a random task to the first free processor found. Bad allocation was corrected using replication. In version 2.0 of OurGrid, a new scheduler tries to avoid communication cost by introducing storage affinity. Tasks are sent to computing nodes that are closest to the data. This algorithm tries to avoid the need for redundant information about tasks such as expected completion time. The first algorithm was found to be still more efficient on some CPU-intensive workloads.

4.4 Internet Volunteer Desktop Grids

Internet Volunteer Desktop Grids systems have been among the largest distributed systems in the world. Projects such as SETI@Home or distributed.net are able to provide PetaFLOPS of computing power for applications from hundred of thousands nodes. In this case of Grid, owners of resources are end-user Internet volunteers who provide their personal computer for free. IVDG are an extremely attractive platform because there offer huge computational power at relatively low cost.

BOINC, the Berkeley Open Infrastructure for Network Computing (BOINC) [57] is the biggest volunteer computing platform. More than 900 000 users from nearly all countries participate with more than 1 300 000 computers [53]. More than 40 projects, not including private projects, are available including the popular SETI@Home project.

Each *client* (computing node) is manually attached by the user to one or more *projects* (servers). There is no central server and most of the scheduling is done by clients. The server is made of several daemons. Each project needs to provide a very small number of different applications but that may be often updated (jobs have to be very homogeneous). Each project must last several months mainly because of the time needed to obtain volunteers and their computing resources.

First, *workunits* are produced by a *generator*. Then, the *transitioner*, the daemon that will take care of the different states of the *workunit* life cycle, will replicate (redundancy) the *workunit* in several *results* (instances of *workunits*). Each *result* will be executed on a different client. Then, after being returned back to the server, each result will be checked by the *validator* before being stored in the project science database by the *assimilator*.

All communication is done using CGI programs on the project server. So, only port 80, and client-to-server connections are needed. Each user is rewarded with *credits* (virtual money) for the count of cycles used on its computer.

The client maintains a cache of *results* to be executed between connections to the Internet. The scheduler tries to enforce many constraints: First, the user may choose to run the applications according to its activity (screen-saver), working hours, and resources available. Second, the user assigns a resource share ratio to the projects. Third, sometimes, some projects may run out of work to distribute.

Some other projects were inspired by the BOINC platform. SLINC (Simple Light-weight Infrastructure for Network Computing, http://slinc.sourceforge.net/) [58] tries to fix the main limitations of BOINC. They tried to make project creation easier by adding a better documentation. This software is also operating system independent as it runs on the Java platform. It is also database independent (they use Hibernate) while BOINC runs only with MySQL. All communications between components are done with XML-RPC and for simplifying the architecture they have removed the validator component. User’s applications are also programming language independent, but only Java and C++ available for now. Two versions of the same application, the first one written in Java, the second one written in C++ have almost the same performance. Some BOINC issues have not been fixed here, such as the time needed to have all the volunteers register their resources.

5 Dependability Mechanisms for Desktop Grids

In this section, we briefly present some avenues of research in the topic of dependability of desktop Grids.

5.1 Task-Dependability in Large-Scale Desktop Grids

While Grids in general and desktop Grids in particular have become very popular and powerful in recent years, they raise many issues related to different aspects of dependability. In particular, desktop Grids with their huge size, often in the scale of millions of distrusted and autonomous computers pose new challenges of great interest. Just to name a few of these challenges, we can refer to the lack of credibility of results computed by anonymous workers, the need to ensure safety to workers running unknown tasks submitted by third parties or the need to find scalable solutions that resist to saboteurs.

In desktop Grid systems, one of the main problems is precisely to ensure validity of results. The current generic approach to solve this problem relies on an *n*-redundancy scheme (using *n* computers to compute the same result). Unfortunately, this is not optimal, because it strongly reduces the available computing power. As one can easily see, even setting *n=2* halves the total computing power. One approach to reduce the computational burden of simple replication is to embed short tests in the tasks given to workers. This approach is known as "spot-checking". At the end of the computation, results of all tests are returned jointly with results of the main tasks. This should be done in a transparent manner, such that workers are unaware of the tests. Although the test approach is not new, a current problem is the lack of support in desktop Grid middleware for embedding such tests without burdening the programmer. Ideally, embedding tests should be semi-automatic: the middleware framework should provide a set of tests, with the application programmer selecting the tests that he or she would deem most appropriate. The middleware framework would then embedded the tests and provide support for automating the assessment of dependability of the remote worker. Note that the complexity and number of embedded tests should depend on the credibility that the worker node has on the desktop Grid system. A newly joined node should be highly scrutinized, while a worker with a history of successful tests would be less tested. Conversely, worker nodes having failed some or all embedded tests would be treated as suspicious (and possibly black-listed as we refer ahead).

In a previous study within CoreGRID (involving UCO and INRIA) we have used an experiment in a real desktop Grid to demonstrate some of the limitations of spot-checking. First, it was observed that there is a minority of nodes that cause most of the errors in computation (frequent offenders). Spot-checking is quite effective to detect this kind of nodes and can easily improve the error rate of the overall computation with a limited effort (in terms of tests given to workers). Unfortunately, after spot-checking detects all frequent offenders it becomes almost powerless against occasional errors of the remaining nodes. Unlike this, the error-rate of a technique like replication falls exponentially with increasing values of *n*. This raises the possibility of using some hybrid approach, capable of simultaneously using spot-checking and replication. If the project could accept a relatively high error-rate it should use spot-checking, then switching to replication and majority voting if the project would need even lower error-rates. Additionally, it would be interesting to have some measure on the number of replicas and on the amount of tests needed to achieve some specific thresholds for the error-rate.

One idea to defend desktop Grid systems from frequent offenders is to black-list those workers. Unfortunately, blacklisting is often difficult to do, because malicious users can very easily register with different identities and use a myriad of fraudulent schemes to re-enter computation. One interesting approach here would be to follow, the reciprocal path of building a network of trustworthy desktop Grid nodes. We could resort to social-like organizations, where each node knows only some subset of the community. In this community, nodes can give ranks to each other, can invite new peers, share responsibilities with newcomers they invited, etc. The goal here would be to build databases holding evaluations of dependability for worker nodes, with the databases being shared among the users that aim to resort to desktop Grid resources for executing some tasks. A challenge to overcome is the need to prevent shared databases from being contaminated with fake and/or malicious evaluations.

Interestingly it is also possible to mitigate some of the costs associated to replicated computation, namely one can try to use results coming from replicas to reduce turn-around time of tasks or to increase the overall throughput of the system. For instance, replicated worker nodes can create message digests from the intermediate checkpoints of the same task. Each node involved in the same task computes its own message digest concerning the same equivalent intermediate execution point. By comparing small intermediate message digests, the master can assess the correctness of each node involved in the computation as this node makes progress in the task. A node that presents message digests different from the majority is flagged as erroneous. When this occurs, the central master can immediately reschedule a replica (avoiding the penalty of only detecting the mishap at the end of the whole computation). Additionally, it can also degrade the dependability mark for the worker that was flagged as incorrect.

5.2 Dependability using P2P Techniques in Desktop Grids

Volunteer Computing systems, such as BOINC [64] and XtremWeb [65], rely on centralized architectures. This centralized model is a potential bottleneck because of the central point of failure, making it very difficult to achieve full dependability. As stated by Ian Foster in [66], there is an increasing convergence between Grid and P2P systems. P2P addresses failure but lacks support from a standard infrastructure, whereas the Grid addresses infrastructure but lacks support for fault-tolerance. The use of P2P techniques in Desktop Grids creates new possibilities for dependability mechanisms such as fault-tolerance schemes.

The architecture of central servers normally implies a separation between data-management and task-scheduling. Therefore it seems opportunistic to apply P2P techniques at the data-distribution module in a desktop Grid infrastructure.

Several Grid systems using a P2P network have been proposed in the literature. Such systems adopt different models and solutions for resource discovery in Grid environments, including fully decentralized [67] or super-peer architectures [68], and diverse strategies for improving routing performance and search precision.

Fully decentralized architectures are typically based on Distributed Hash Tables which implement their own dependability mechanism. Chord, for example, employs backup links and data replication. The response to a node failure is to update the links in the routing tables of the affected nodes to reflect the change, and also to restore the number of replicas of data items stored at the failed node by copying them to other nodes. The biggest disadvantage of this approach is that the self-repair algorithm permanently increases the load on the surviving nodes and reduces the total amount of redundancy in the overlay.

On a Desktop Grid environment it is not desirable that nodes would be simply discarded. Therefore, it is debatable whether fully decentralized systems should only be applied to Grids in controlled environments, with heavy replication of work units, and small jobs submitted by end-users. The application of this approach to a Desktop Grid environment would have to be coupled with failure prediction algorithms and optimized replication mechanisms, to avoid the loss of work units, while preventing the flooding of the network. None of this has been done so far and seems to be a good avenue of research.

Super-peer architectures, on the other hand, incorporate peer selection techniques that allow for reliable peers to be selected for scheduling and task distribution. However, most proposals are more concerned with task execution time and network/task throughput, while experiments do not take into account node failures (either worker or super-peer). Dependability mechanisms typically are limited to data replication and the use of time-outs to know when a worker has failed during a task execution. Experiments with different parameters such as availability, network and processing capacity are needed to evaluate how well a super-peer network can perform on a Desktop Grid environment. Finally, more complex dependability mechanisms should be used, such as optimized replication of task execution, methods to predict task execution time, and data replication mechanism to ensure that connections to the central/dedicated servers are kept to a minimum.

However, applying P2P techniques to increase the fault-tolerance is not always the best solution. For instance, BOINC has a heavy dependence on a centralized database, and that makes a full migration to a P2P system raise more problems than bringing advantages. There have been a few proposals for data distribution in desktop Grids using Peer-to-Peer techniques, such as super-peer architectures [69] or BitTorrent [70].

Super-peer architectures suffer limitations mentioned previously, using limited dependability mechanisms that do not include any fault detection or sabotage tolerance techniques. The lack of complex mechanisms is not as problematic in this case, since the scheduler remains centralized and can coordinate the downloads: for instance, by using file hash codes it is possible to check for incorrect downloads and decrease the chance of sabotage. The super-peer selection process has not been studied with enough depth, leaving many doubts as to whether availability is guaranteed in these systems. Future research should focus on super-peer selection techniques to provide availability, as well as failure detection mechanisms (more complex than simple time-outs mechanisms).

Research on data distribution protocols has been limited so far. Experiments on BitTorrent have only considered a centralized tracker, which creates a single point of failure. If the tracker crashes, there is no failure recovery, since the clients lose the possibility to identify new peers. To address this problem a distributed tracker should be considered. Future research should focus on applying existing fault-recovery mechanisms on distributed tracker scenarios, and developing methods to choose clients that will work as trackers while maintaining availability and tolerating sabotage.

Furthermore, the difficulties in creating a realistic environment have made the experiments rely on simulation or small-scale scenarios which limit the scope and generalization of what is concluded. Tests on a more realistic deployment should be pursued in the future, using for example BOINC projects such as BOINC Alpha and instrumenting the BOINC client to log the necessary data.

5.3 Accountability of Resources

A paramount issue for desktop Grids relates to the proper accountability of resource usage. Without a solid methodology to properly assess the resource usage consumption, no commercial model can be seriously considered, confining desktop Grids to non-commercial environments such as academic applications. Therefore, a key avenue to explore with desktop Grids is the creation and certification of dependable accounting methods for proper assessment of resource usage. This is a major challenge for the future of desktop Grids.

5.4 Trust and Security Issues

In the Internet world, trust has been recognised to play an important role in decision making 72, 73. Customers and sellers must trust each other, and the services they advertise; they should not disclose private information such as name, address, credit card details etc. Josang et al. 73 define trust as "the extent to which one party is willing to depend on something or somebody in a given situation with a feeling of relative security, even though negative consequences are possible".

Fundamental to Grids is the idea of resource sharing [74], where various providers collaborate and contribute with resources for supporting various computations. As the Grid is indented for usage in business environments, trust and security issues become of prime importance, while it is necessary to share resources with unknown parties. In general, a security mechanism provides protection against malicious users. Traditional security typically protects resources by restricting access only to authorized users. However, within distributed applications and in Grid and P2P systems in particular, the resource consumer needs to be protected from those who offer resources, so that the problem is in fact reversed.

Internet Desktop Grids represents an important research axis for the Institute on Architectural Issues. These systems acquire increasing popularity for demanding applications, supplying with a huge computing power collected from Internet volunteers. As desktop Grids rely on anonymous volunteers without any mean of control, effective techniques for sabotage tolerance and trust management become of essential importance. Sabotage tolerance should be mixed together with scalable protocols for resilient task and data distribution, which induces further complexity in the architectural design of the DG systems. Domingues et al. [78] reviews existing sabotage tolerance protocols for desktop Grids. They also propose a volunteer invitation system, in order to tackle the anonymity issues of DGs.

P2P architectures are considered in the attempt to make efficient and cost-affordable the DG data distribution problems like peer collusion come into study. Silaghi et al. [79] investigates the use of reputation to enhance sabotage tolerance. They propose the usage of a reputation-based weighted voting decision criterion during task replication, to improve the efficiency of the sabotage tolerance protocol.

Directly targeting the BOINC environment equipped with a P2P data distribution layer, Silaghi et al. [80] propose a very effective protocol to complement the actual replication, in order to tackle the peers' collusion threat. This protocol postpones the decision moment regarding the assessment of the validity of a result until the master collects enough data about the voting behaviour of DG nodes. Having a good history of votes, the master can infer with a good certainty if a worker is behaving honestly or not. Thus, performing further auditing of the suspect results, the master can identify the malicious workers and keep low the error rate of the system.

Marosi et al. [81] presents the security issues related with the development of the STAKI Desktop Grid. STAKI DG hierarchically builds on top of BOINC, which provides the open infrastructure for the public resource computing. In order to support the hierarchy model and the industrial nodes, the authors extended the security model of BOINC. The new security model should provide enough information for allowing the user to trust

- any workunit installed locally on the BOINC project,
- any workunit from a given project regardless of how many levels of hierarchy the workunit travelled through
- any specific application, regardless of where in the hierarchy it is hosted and regardless of other applications.

They realized the above-mentioned trust relations by digital signature verification, using the X.509 Public Key Infrastructure. The protection of the input data from unauthorized download is achieved by giving every user a certificate. The Project manager acts as a Certification Authority and signs the certificates. The query for certificates is done via the HTTP(S) protocol, depending on the trust model selected by the user.

6 Adaptability and Prediction Mechanisms for Grid Systems

During the last four decades, system architects have mainly focused on performance issues. Their quest for performance was overly successful, but only at the cost of an increased software complexity that makes computer systems difficult to operate and maintain. Vertical software integration like the popular Service Oriented Architecture

(SOA) helped reducing the barriers for system use, but if something fails, only experienced software experts are able to trace down through the many software layers to the source of failure.

The aggregation of many independent heterogeneous subsystems to a well-functioning Grid causes much administration overhead. Since human operators are costly, slow and error-prone, advanced self-management properties are needed that are able to cope with resource variability, changing user needs and system faults.

To master this challenge, future Grid systems must exhibit *adaptability* to a much stronger degree than today. Adaptable middleware is able to mask changes in the execution environment. Such changes may be caused by variations in the availability of processors, networks, storage. With the increasing size and complexity, adaptability is among the most badly needed properties in today's Grid systems. Adaptability is closely related to the concept of *self-management*. This vision suggests that specialized software components take care of performance tuning, application configuration, failure compensation and recovery, capacity planning, and other duties [93].

In this section, which is based on [82], we overview concepts, methods, algorithms, and implementations that are deemed useful for designing adaptable Grid systems. Our inventory is done along the stages of the feedback loop known from control theory. These stages include monitoring, analyzing, predicting, planning, decision taking, and finally executing the plan.

6.1 Feedback Loop

Open/Closed Loop: Adaptable systems are in a state of continuous self-regulation [89] through a feedback loop. Deviations of output from some ideal or desired state are fed back into the control unit, which then acts to minimize the discrepancy. Open loops are commonly used when services act on static data or when they quickly return some status information. Closed loops, in contrast, are used for continuously running services, like the load-balancing of file availability in peer-to-peer systems [95]. In large Grid systems, closed loops with external input are perhaps the most important model. This is because local site administrators may change resources or services at any time without prior notice.

Autonomic Grids: The described feedback loop is sometimes referred to as 'autonomic' a term with a biological connotation. It indicates the unconscious self-regulation [93] within human bodies and economic systems. Each autonomic element consists of one or more managed systems and a control cycle, as shown in Fig. 4. Sensors (not shown in Fig. 4) observe behavioral characteristics of the system and report them to a monitor. The monitor collects, aggregates and filters the data and logs them for further use. An analyzer provides functions and mechanisms for correlating complex situations. The planner constructs actions that are needed to achieve the user-specified goal from the current status. As there are often multiple ways to achieve the goal, the planner may be guided by the results of a predictor which determines forecasts based on time-series analysis. This allows the system to 'interpret' situations and predict future behavior.

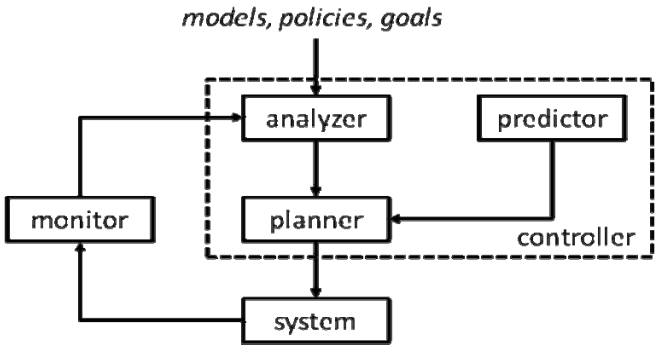

Figure 4: Feedback loop in more detail.

6.2 Modeling Grid Systems

The actors of a feedback loop operate in the context of managed Grids, and therefore require a model of the managed entities, their relationships and possible actions. Also the flow of information between the stages of the feedback loop and to/from the external interfaces requires a specification of an exchange format, both in syntax and in semantics. Such models and their description can be application dependent, for example having a form of C-like records or a file with a proprietary structure. However, for the sake of re-usability it is preferable to make them generic and to standardize them.

The mentioned model and its description ideally would be able to capture the architecture of the managed system, its state, the allowed management actions, desired target system states and the optimization goals. Currently, none of the existing specification frameworks adheres to all of these requirements, but the specification standards discussed below are likely candidates to be extended in this direction.

- **CIM** The *Common Information Model (CIM)* [87] is a conceptual model and language standard for describing computing and business entities in Grid, enterprise and service provider environments. It is an ongoing effort by the Distributed Management Task Force (DMTF), a non-profit collaborative body comprised of academic and industrial members that is leading the development of management standards for computing system environments.
- **SDL** The *Specification and Description Language (SDL)* [91] has been introduced by the Telecommunication and Standardization Sector of ITU as a means to describe behavior, data and structure of particularly larger systems. SDL was originally targeted for specifying telecommunications real time systems, for example call and connection processing in switching systems, maintenance and fault treatment in such systems, or data communication protocols.
- **Policies** A *policy* is a definite goal, course or method of action to guide and determine present and future decisions [92]. Traditionally, the term policy is used to describe parts of the system configuration that controls system behaviors such as in security policies or quality-of-service policies. There are many approaches to describe policies, including logic-based languages or Role-Based Access Control (RBAC) specification in the security area, and CIM, Policy Description Language (PDL), or event-trigger-rules in the system management area [85]. The most sophisticated policy description language is perhaps Ponder [86] which can be used for specifying management and security policies.

6.3 Planning and Decision Taking in Adaptive Grids

Analysing and Predicting System State Modeling and predicting the demand of individual servers or their clusters is one of the key supporting techniques for the automated management and scheduling of computing resources. In Grid environments, modeling and prediction of the future application demand facilitates the performance tuning, anomaly detection, scheduling of jobs, sharing of resources, capacity planning, or discovering interdependencies between applications.

Among different approaches in this field three methods have proved to be applicable for these purposes: classical ARIMA/Kalman filter timeseries modeling [94], classification based on data mining methods [90], and mining of repetitive patterns in the demand by means of sequence mining [81].

Automated Planning A plan is a partially ordered collection of actions for performing some task or achieving a certain goal. Automated planning is a field of AI still in development, with many research prototypes and few practical systems [88]. While the research has still to cope with a host of difficult practical problems in this domain, mostly involving computational complexity, there are several successful cases such as the NASA DS1 mission.

In the field of Grid systems, automated planning can be used to automate complex tasks involving heterogeneous resources and having a lot of interdependent steps. Examples of such tasks are the migration of distributed jobs including data transfer and software installation, or the construction of complex resources such as virtualised server farms on demand. Given the specifications of possible actions in such an environment, an automatically generated plan can be translated into a standardized workflow and distributedly executed [83].

Optimization *Optimization* plays a central role in the tuning of system parameters, e.g. for increasing performance or allocating resources. In its most general form, an optimization problem consists of a set of variables for which value assignments are sought, a set of constraints imposing relationships between these variables, and possibly one or more objective functions whose values should be maximized or minimized. Note that if we drop the objective function(s) and just seek an assignment of values to variables which satisfy the constraints, then this definition also covers satisfiability and constraint satisfaction problems.

If the variables take real values, and both the constraints and the objective function is linear in the variables, such an optimization problem is called a *linear program*. As to annoy the computer scientists striving for adaptability of systems, real optimization problems in system management are rarely representable as linear programs.

However, not every problem must be solved optimally in most cases a good or even any solution will do, and here heuristics come into play. While in many instances heuristics such as simulated annealing or genetic algorithms work much faster than the exact algorithms, there is a trade-off: heuristics do not guarantee an optimal solution, and usually it remains unknown by how much the found solution is worse than the optimum.

Expert systems: Another approach for planning and decision taking are traditional expert systems from the field of artificial intelligence, studied since the 1970s. An expert system consist of a model of the problem space and an inference engine which analyses a given state and either proposes actions to improve the state or derives new facts

about the current state (diagnosis). Such systems can be used both for the analysis and the planning inside adaptable Grids.

6.4 Challenges and Limits of Adaptability

The Formalization Challenge We have discussed above the limits of self-managing solutions due to the inherent complexity problems. However, another factor is likely to become a key bottleneck towards self-management: the formalization challenge. Simplifying, the formalization process has to do with the interaction between 'what we want' (what we expect from the program) and how to force the machine to behave in this way [84].

Consider a domain which bears strong similarities with the core of autonomic computing - creation of software. The formalization problem there certainly generalizes the autonomic computing problem, since in all by few exceptions the means to attain the self-managing functionality is software. Does it mean that the effort of formalization for self-management is similarly high as in the programming problem? This is not necessarily the case, since in the domain of self-management the required solutions are simpler (and more similar to each other) than in the field of programming, and so the benefits of domain-specific solutions can be exploited.

Service Semantics In dynamic Grids, services should be exchangeable. This is currently done using catalogs where all services are registered with their interface and name. When two services have the same interface and service name they are assumed to be exchangeable. Of course this is not necessarily the case, because the semantics of the services may differ. Among other problems, this has great impact on the reliability of Grid systems. Taking this to the extreme we could search for some composable services that, in combination, provide the desired semantics. This, however, is equivalent to an automated theorem prover, which is very compute intensive.

Inherent Limits of Management Automation One goal of autonomic computing is to hide faults from the user and to first try to handle such situations inside the system. Yet in some cases it would be not a good idea to try to hide errors. If, for example, the user specified a non-existing file as input data, the system should immediately report this back to the user and should not try to hide this error by waiting until such a file is created sometimes in the future.

Finally, automation of management tasks does not come without cost, and in some cases this effort does not pay off, e.g. if such a task is very rare. Measuring or even estimating the cost of automation can help to decide whether it is cheaper to leave some scenarios not automated.

In this section we have discussed requirements, features, and possible approaches of the adaptive Grid systems in context of the feedback loop. Such a loop is inherent in self-managing systems, and includes as the essential stages monitoring, analysis/modeling, decision taking and execution.

The self-management challenge has created many research activities, yet real 'breakthroughs' seem to remain elusive. Reaching this goal requires a lot of effort and improvements in a multitude of fields, which makes singular 'quantum leaps' unlikely. A partial reason for this is the fact that many problems to be solved are not new, but of a more fundamental nature, for example in the field of automated planning.

7 Towards Adaptive and Scalable Service Oriented Architectures

The diffusion of Grid technology and the desire to use distributed computing as a utility are promoting new business models for providers that would deliver computing functionalities, eventually customized on demand, to host applications and to meet customer needs [97]. In fact, due to the rapid growth of complexity, and the short time-to-market for their effective exploitation in actual scenarios, the applications are no longer built from scratch. A company or an organization is not able to design and develop by itself complex and distributed applications, both for the lack of all the needed competencies and for economical reasons. A direct consequence of such phenomenon is the advent of third-party companies, called *Service Providers* (SPs), which will surely characterize a main future trend of IT industry. The SPs are companies specialized in the remote provision of ready-to-use utilities (from computational resources access to sophisticated instrumentation provision and data storage services, etc.), deployed on their own dedicated resources.

In order to improve the programmer’s productivity in designing and developing complex applications that efficiently exploit Grid environments, new technologies and approaches are required. An architectural approach that is becoming popular to address heterogeneity, distribution and interoperability issues of a Grid environment is the Service-Oriented Architecture (SOA), which is in full contrast with the previous *stand-alone* one. Stand-alone applications were typically designed to support only intra-company access, based on different formats for data exchange and were developed using different platforms and dedicated technologies. This approach was the main reason of the limitation of cooperation and dynamic integration in existing applications.

Even though a SOA model is independent from a specific technology, Web Services are becoming a de-facto standard for SOA implementation since they are spreading rapidly, mainly for their support to ease application programming, portability, maintenance and integration among legacy or new services exposed on the Internet. Standard Web Services technologies in fact extend the advantages of SOA making it possible to employ an existing low-level infrastructure based on Web servers and the HTTP protocol, which are today very pervasive thanks to the high diffusion of the Internet and the Web.

The OGSA framework [98], which follows the SOA approach through the Web Services technologies, considers a Grid as an extensible set of services, called *Grid Services*, which conform to specific conventions. WS-Resource Framework (WSRF), a set of Web service specifications being developed by the OASIS organization, and the WS-Notification (WSN) specification describe how to implement OGSA capabilities using Web Services. The Globus Toolkit 4.0 [99] provides an open source WSRF development kit and a set of WSRF services.

In a SOA environment, Grid applications can be viewed as a composition (workflow) of independent services delivered by distributed providers [99]. A Grid workflow can be obtained by composing domain dependent services, which virtualize the access to specific and high-level utilities typically delivered by providers that leverage high-performance dedicated clusters and software libraries for complex computations.

Differently from other application domains tied to B2B environments, Grid workflows orchestrate services that virtualize the access to resources delivering low-level functionalities, such as data acquisition, computation and storage. Such services are characterized by heterogeneous assets and performance (due to heterogeneity and sharing of resources onto which they are deployed). As a consequence, heterogeneity represents one of the key elements of Grid computing but also one of the main problems for an efficient execution of Grid applications. For this reason, a lot of research in this area has been devoted to the definition of infrastructure components able to hide computing heterogeneity to ensure computing power transparency [101].

In a connection model based on middle agents, such as the model proposed by SOA, a fundamental role for achieving the desired transparency is played by resource managers, matchmakers and brokers that handle the space of services by statically or dynamically selecting the desired services to satisfy application needs. In Grid context, these components have, typically, a slightly different behaviour with respect to the equivalent components in a pure Web Services environment since the selection of services is performed mainly on the basis of QoS attributes that characterize the heterogeneity of resources onto which they are deployed. Therefore, Grid brokers should be able to automatically discover available functions in the network, choose and schedule the ones that satisfy functional requirements and QoS constraints specified by the clients (users or application components).

In order to satisfy non functional requirements of applications (related to time execution, reliability, security, etc.) Grid brokers have to use selection strategies to assign resources to clients and schedule them fulfilling such criteria. With respect to temporal composition, many compute and data-intensive functionalities in scientific and Grid workflows (such as data analysis, system simulation, image processing, database searching, etc.) are characterized by coarse-grained parallelism that allows for increasing performance through the exploitation of a pool of distributed resources. In particular, used patterns can be divided in two main categories: *pipelined execution patterns*, characterized by the sequential execution of dependable tasks, and *parallel execution patterns*, which can be distinguished in simple parallelism pattern (parallel execution of tasks without dependencies) and dynamic data parallelism pattern (concurrent execution of the same task on different parts of the initial input data set dynamically distributed among resources) [99].

Fig. 5 shows a generic example of the dynamic data parallelism pattern in a Grid workflow. In this example data produced by Task B are a set of inputs for Task C, which can be replicated according to an input partitioning in order to exploit multiple distributed resources.

A full exploitation of multiple resources to execute Grid workflows will be reached if the following main issues will be taken into account: 1) the adoption of matching strategies able to find a pool of resources satisfying global constraints on applications; 2) definition of formal languages for QoS description of Grid services in order to avoid ambiguity during matching; 3) mechanisms for dynamic and transparent composition and coordination of services.

An answer to the first two issues is proposed in [103]. Here a framework for QoS brokering of resources virtualized through Web Services and its customization is presented. The framework is able to automatically allocate application tasks based on the data parallelism pattern through a time and cost-based matching strategy, called time minimization matching strategy. The algorithm is based on divisible load theory [104], whereas the resource broker is based on a matchmaking framework [103] to support multi-criteria matching strategies in B2B and Grid based environments. Such framework is extensible and customizable with respect to application scenario through the dynamic configuration of syntactic-, structural- and semantic-based discovery and matching strategies, features that make it a suitable and easy-to-use environment to test new search and selection strategies.

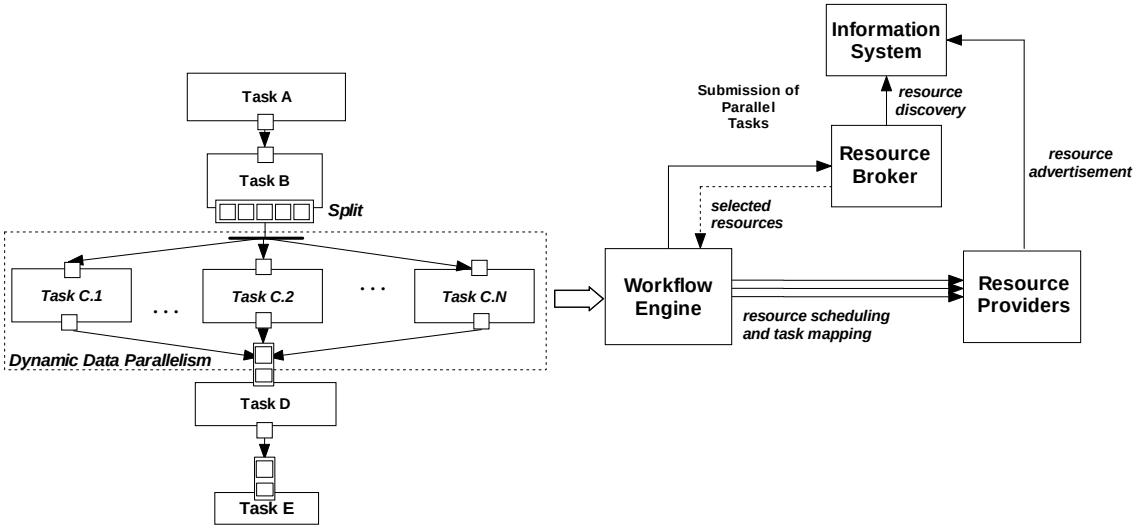

Figure 5: Real example of Grid workflow and resource selection in a SOA environment.

The adoption of SOA and service technologies in Grid computing enables interesting mechanisms towards the automation of workflow composition [105]. A possible approach to achieve this goal is the knowledge formalization through semantic annotations of deployed services with the aim of supporting a composer of services. This has the role of solving a problem by producing a plan that may be translated into an abstract process (functional description). To achieve this objective, a composer accesses to the semantic description of services, which capture some services' aspects such as input, output, preconditions and effects that are used together with the problem definition, the initial state, the goal state, and, in some cases, the constraints for the planned solution. Planning is supported by a knowledge base that manages the information obtained during the planning. The knowledge base allows for taking advantage of discovered relationships or properties that are stored as assertions and are used to increase the efficiency for dealing with problems in a specific domain.

It is worth to note that a common approach used in SOA for service discovery, composition and execution is centralized and supervised and these functions employ some centralized middleware components, i.e. brokers, planners, workflow engines, respectively. All these components, will become bottlenecks when the number of services and organizations using services will grow, since service discovery requires using sophisticated matchmaking algorithms to identify semantic similarities between consumers template and providers target descriptions, automatic composition requires using both matchmaking and search-based algorithms, whereas service execution through orchestration requires a lot of network interactions.

An interesting solution we envision for the above-mentioned problems is a P2P network of services to exploit cooperative and parallel approaches for service discovery, composition and enactment. In such a system of services, nodes should be able to communicate to find each other through discovery mechanisms that ensure high efficiency and scalability, with the aim of reducing response times. To this end, each node should be able to interpret an incoming goal and to give a partial or total contribution to the solution, even individuating other nodes in the network able to contribute with a piece of knowledge.

Fig. 6 shows an example of cooperative composition in the space of services for discovering and composing some services that solve the logical computation problem of the task B of the workflow shown in Figure 2. Each node represents a peer hosting one or more services. Each service published on a peer exposes one operation identified through the label Pr → Po, which means that Pr is the precondition and Po is the postcondition of an operation. Node 1 injects in the network a goal (A →Z) with the aim of discovering and composing the services whose execution changes the state of the system from A to Z. The composition is the shortest path between the post-condition Z and the precondition A of the desired abstract service [106].

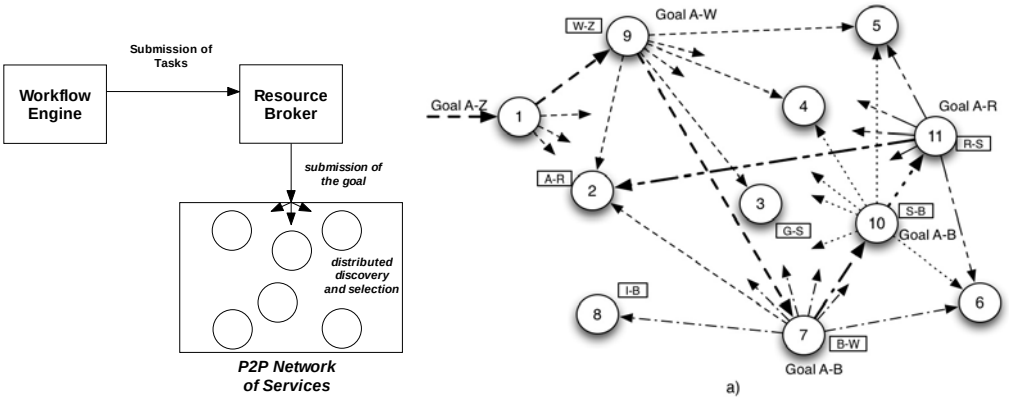

Figure 6: Service discovery and automatic composition in a P2P network of services.

Each node in the network contributes to discover the peers that can originate useful compositions. According to the P2P model, peers become a crucial part of the architecture, since with this model the network lacks of structural components for discovery and composition. Each peer is responsible of receiving requests from other nodes (goals), and fulfilling them (i) by relying on service operations or lower level features available on each peer or (ii) by forwarding the request to other known peers. In many cases a peer can be able to fulfil a request by composing some of its operations with operations made available by other peers (see peers 1, 9, 7, 11). In such a case, a peer is also responsible of composing these operations, to fulfil either partially or totally the request received.
Moving SOA towards a P2P model will enable in the next future new network-based applications and will propose new challenges for Grid computing regarding discovery and matchmaking, service composition and decentralized execution of services. All these problems open new horizons for research in large-scale, distributed systems both at algorithmic and infrastructural levels.

8 Conclusions

Grid computing managed to achieve considerable growth in recent years. However, several key areas of active research remain before Grid computing platforms achieve true dependability, fault-tolerance, autonomic self-management, trust, security, and scalability of services. The Architectural Issues Institute of the CoreGRID NoE offered a major contribution towards this direction rendering improved architectural components for Grid systems. This white paper is an effort to present the current state-of-the-art in this field and to expose the remaining problems and future research trends.

With the emergence of Cloud computing, several of the techniques that have been developed in the context of Grid systems will find immediate use. It will not be long before Cloud computing expands its application domain to commodity PCs in order to take advantage of the abundance of available resources they posses, mostly in terms of computation, available storage, and bandwidth. Rendering desktop PC platforms reliable, and adapting them to exploit their capabilities while overcoming their shortages will open up their use in various application scenarios related to Cloud computing, resource leasing, and use by appropriate service providers. In order to effectively employ these techniques, methods for scalable, dependable, and adaptable large-scale computing infrastructures like the ones elaborated in this paper will be of great use. Techniques for scalable resource discovery, possibly using the P2P approach, dependability for desktop Grids, or techniques for distributed storage and replication, survivability of checkpoints of computations in volatile computing environments are vital to secure dependability of large-scale computing infrastructures based on commodity hardware. Furthermore, adaptability for dynamic relocation of jobs is also very important for the virtualization layer of Cloud computing infrastructures.

This paper addressed some crucial Grid architectural issues, indispensable for the emergence of critical future applications of large-scale distributed computing systems. It is a general belief that the years to come will bring even greater research activity in this domain so that large-scale distributed computing becomes established as an indispensable tool not only for academic applications that exhibit best effort behaviour, but also for demanding commercial application that require high dependability, improved performance predictions mechanisms, adaptability of the infrastructure, and scalable services. These requirements were emphasized as vital architectural issues in all three NGG reports. This includes the NGG3 report, where scalability, dependability, and adaptability are referenced as the three most important properties for the architecture layer of the emerging SOKU model.

References

1 P. Trunfio, D. Talia, H. Papadakis, P. Fragopoulou, M. Mordacchini, M. Pennanen, K. Popov, V. Vlassov, S. Haridi, "Peer-to-Peer resource discovery in Grids: Models and systems". *Future Generation Computer Systems*, vol. 23, n. 7, pp. 864–878, 2007.

2 A. Iamnitchi and I.T. Foster, "A Peer-to-Peer Approach to Resource Location in Grid Environments", In: J. Weglarz, J. Nabrzyski, J. Schopf and M. Stroinski (Eds.), *Grid Resource Management*, Kluwer, 2003.

3 D. Talia and P. Trunfio, "Peer-to-Peer Protocols and Grid Services for Resource Discovery on Grids". In: L. Grandinetti (Ed.), *Grid Computing: The New Frontier of High Performance Computing, Advances in Parallel Computing*, vol. 14, Elsevier Science, 2005.

4 D. Puppin, S. Moncelli, R. Baraglia, N. Tonelotto and F. Silvestri, "A Grid Information Service Based on Peer-to-Peer". *Proc. 11th Euro-Par Conf. (Euro-Per 2005)*, LNCS, vol. 3648, pp. 454-464, Springer, 2005.

5 M. Marzolla, M. Mordacchini and S. Orlando, "Resource Discovery in a Dynamic Grid Environment", *Proc. DEXA Workshop 2005*, pp. 356-360, 2005.

6 A. Andrzejak and Z. Xu, "Scalable, Efficient Range Queries for Grid Information Services". 2nd IEEE Int. Conf. on Peer-to-Peer Computing (P2P 2002), Linköping, Sweden, 2002.

7 M. Cai, M. R. Frank, J. Chen, P. A. Szekely, "MAAN: A Multi-Attribute Addressable Network for Grid Information Services". *Journal of Grid Computing*, vol. 2, n. 1, pp. 3–14, 2004.

8 D. Oppenheimer, J. Albrecht, D. Patterson and A. Vahdat, "Scalable Wide-Area Resource Discovery". TR CSD04-1334, Univ. of California, 2004.

9 D. Spence and T. Harris, "XenoSearch: Distributed Resource Discovery in the XenoServer Open Platform", *Proc. Twelfth IEEE Int. Symposium on High PerformanceDistributed Computing (HPDC-12)*, pp. 216–225, 2003.

10 A.R. Bharambe, M. Agrawal and S. Seshan, "Mercury: Supporting Scalable Multi-Attribute Range Queries", *Proc. ACM SIGCOMM 2004 Conf. on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pp. 353–366, 2004.

11 M. Castro, M. Costa, A. Rowstron, "Debunking Some Myths About Structured and Unstructured Overlays". 2nd Symp. on Networked Systems Design and Implementation (NSDI'05), Boston, USA, 2005.

12 Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, S. Shenker, "Making Gnutella-like P2P Systems Scalable". ACM SIGCOMM'03, Karlsruhe, Germany, 2003.

13 C. Mastroianni, D. Talia, O. Verta, "A super-peer model for resource discovery services in large-scale Grids". Future Generation Computer Systems, vol. 21, n. 8, pp. 1235–1248, 2005.

14 H. Papadakis, P. Fragopoulou, E. Markatos, M. Dikaiakos and A. Labrinidis. Divide et Impera: Partitioning Unstructured Peer-to-Peer Systems to Improve Resource Location. TR-0065, Institute on System Architecture, CoreGRID, 2007.

15 H. Papadakis, P. Fragopoulou, E. Athanasopoulos, E. Markatos, M. Dikaiakos and Labrinidis. A Feedback Based Approach to Reduce Duplicate Messages in Unstructured Peer-to-Peer Systems. TR-0029, Institute on System Architecture, Core-GRID, 2006.

16 D. Talia, P. Trunfio. "Dynamic Querying in Structured Peer-to-Peer Networks". Submitted for publication. Available at: http://grid.deis.unical.it/papers/pdf/DQDHT.pdf.

17 I. Stoica, R. Morris, D. R. Karger, M. F. Kaashoek, H. Balakrishnan. "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications". ACM SIGCOMM' 01, San Diego, USA, 2001.

18 P. Trunfio, D. Talia, A. Ghodsi, S. Haridi, "Implementing Dynamic Querying Search in k-ary DHT-based Overlays". Proc. of the 3rd CoreGRID Integration Workshop, Greece, 2008.

19 H. Papadakis, P. Trunfio, D. Talia, P. Fragopoulou. "Design and Implementation of a Hybrid P2P-based Grid Resource Discovery System". In: Making Grids Work, M. Danelutto, P. Fragopoulou, V. Getov (Eds.), Springer, USA, 2008.

20 Open Chord. http://open-chord.sourceforge.net.

21 Grid'5000. http://www.grid5000.fr.

22 A. Forestiero, C. Mastroianni, G. Spezzano, "Antares: an Ant-Inspired P2P Information System for a Self-Structured Grid". *Proc. BIONETICS 2007 - 2nd International Conference on Bio-Inspired Models of Network, Information, and Computing Systems*, Budapest, Hungary, 2007.

23 G. Avarez and F. Cristian. "Centralized Failure for Distributed, Fault-tolerant Protocol Testing", Proceedings of the 17th IEEE International Conference on Distributed Computing Systems (ICDCS97), May 1997.

24 A. Brown. “Coping with Human Error in IT Systems”, ACM Queue magazine, http://www.acmqueue.com, November 2004.
25 R. Chandra, R.M. Lefever, M. Cukier and W.H. Sanders. “Loki: A State-driven Fault Injector for Distributed Systems”, Proc. of the Int. Conf. on Dependable Systems and Networks, June 2000.
26 G. Chun, H. Dail, H. Casanova and A. Snavely. “Benchmark Probes for Grid Assessment”, IEEE IPDPS 2004.
27 “CIC”, http://cic.gridops.org/
28 G. Da Costa, S. Orlando, M.D. Dikaiakos, “Nine Months in the Life of EGEE: a Look from the South”, In IEEE MASCOTS 2007.
29 S. Dawson, F. Jahanian and T. Mitton. “Orchestra: A Fault Injection Environment for Distributed Systems”, Proc. of the 26th International Symposium on Fault-Tolerant Computing (FTCS), pp. 404-414, Sendai, Japan, June 1996.
30 C. Dumitrescu, I. Raicu, M. Ripeanu, I. Foster. “DiPerF: An automated DIstributed PERformance testing Framework”, In IEEE/ACM Grid 2004.
31 “Global Grid User Support (GGUS) ticketing”, https://gus.fzk.de/pages/home.php
32 “GridICE”, http://grid.infn.it/gridice/
33 Grid Statistics (GStat) http://goc.grid.sinica.edu.tw/gstat/
34 S. Han, K. Shin and H. Rosenberg. “Doctor: An Integrated Software Fault Injection Environment for Distributed Real-time Systems”, Proc. Computer Performance and Dependability Symposium, Erlangen, Germany, 1995.
35 F.P. Junqueira and K. Marzullo. “The Virtue of Dependent Failures in Multi-site Systems”, HotDep2005.
36 E. Kiciman and A. Fox. “Detecting Application-Level Failures in Component-based Internet Services”, IEEE Transactions on Neural Networks, 2004.
37 E. Kiciman and L. Subramanian. “Root Cause Localization in Large Scale Systems”, HotDep 2005.
38 S. Krishnamurthy, W.H. Sanders and M. Cukier. “A Dynamic Replica Selection Algorithm for Tolerating Timing Faults”, DSN 2001.
39 M.E. Locasto, S. Sidiroglou, A.D. Keromytis. “Application Communities: Using Monoculture for Dependability”, HotDep 2005.
40 “TeraGrid”, http://www.teragrid.org/
41 G. Tsouloupas and M.D. Dikaiakos. “GridBench: A Tool for the Interactive Performance Exploration of Grid Infrastructures”, Journal of Parallel and Distributed Computing, vol. 67, pp. 1029-1045, 2007.
42 K. Neokleous, M.D. Dikaiakos, P. Fragopoulou, E.P. Markatos. “Failure Management in Grids: The Case of the EGEE Infrastructure”, Parallel Processing Letters, December 2007.
43 MPI: A Message-Passing Interface Standard. http://www.mpi-forum.org/docs/mpi-11.ps (accessed June 2006).
44 N. Rodrigues, D. Sousa, L.M. Silva, and A. Andrzejak. “A Fault-Injector Tool to Evaluate Failure Detectors in Grid-Services”, TR-0098, Institute on Architectural Issues: Scalability, Dependability, Adaptability, CoreGRID Network of Excellence, September 2007.
45 D.T. Stott et al. “Nftape: a Framework for Assessing Dependability in Distributed Systems with Lightweight Fault Injectors”, Proc. of the IEEE International Computer Performance and Dependability Symposium, pp. 1-100, March 2000.
46 S. Tixeuil, W. Hoarau and L.M. Silva. “An Overview of Existing Tools for Fault-Injection and Dependability Benchmarking in Grids”, TR-0041, Institute on System Architecture, CoreGRID Network of Excellence, October 2006.
47 G. Tsouloupas and M.D. Dikaiakos. “Grid Resource Ranking using Low-level Performance Measurements.”, Euro-Par 2007.
48 “WISDOM”, http://wisdom.eu-egee.fr/
49 “Service Availability Monitoring (SAM)”,
50 http://goc.grid.sinica.edu.tw/gocwiki/SAM
51 D. Zeinalipour-Yazti, K. Neocleous, C. Georgiou and M.D. Dikaiakos. “FailRank: Towards a Unified Grid Failure Monitoring and Ranking System”, in CoreGRID Springer Volume of Selected Papers from the CoreGRID Workshop on Grid Programming Models and P2P Systems Architecture, Heraklion, Greece, June 2007.
52 D. Zeinalipour-Yazti, K. Neocleous, C. Georgiou, M.D. Dikaiakos, Identifying Failures in Grids through Monitoring and Ranking", The 7th IEEE International Symposium on Network Computing and Applications (IEEE NCA'08), 10 - 12 July 2008, Cambridge, MA USA.
53 Boinc statistics for the world! http://www.boincsynergy.com/stats/index.php
54 Ourgrid’s website. http://www.ourgrid.org/
55 Xtremweb-ch’s website. http://www.xtremwebch.net/

56 N. Abdennadher and R. Boesch. “A Scheduling Algorithm for High Performance Peer-to-Peer Platform, *CoreGRID Workshop, Euro-Par 2006*, Dresden, Germany, August 2006.

57 D. Anderson. “Boinc: A System for Public-resource Computing and Storage”, *Fifth IEEE/ACM International Workshop on Grid Computing*, pp. 4-10, Pittsburgh, USA, November 2004.

58 J. Baldassari, D. Finkel, and D. Toth. “Slinc: A Framework for Volunteer Computing, *Proceedings of the 18th IASTED International Conference on Parallel and Distributed Computing and Systems, PDCS 2006*, Dallas, Texas, USA, November 2006.

59 F. Cappello, S. Djilali, G. Fedak, T. Herault, F. Magniette, V. Néri, and O. Lodygensky. “Computing on Large Scale Distributed Systems: Xtremweb Architecture, Programming Models, Security, Tests and Convergence with Grid. *Future Generation Computer Science (FGCS)*, 2004.

60 W. Cirne, F. Brasileiro, N. Andrade, L. Costa, A. Andrade, R. Novaes, and M. Mowbray. “Labs of the world, unite! ! !”, *Journal of Grid Computing*, 2006.

61 G. Fedak. “*XtremWeb: Une Plateforme Générique pour l’étude Expérimentale du Calcul Global et Pair-à-Pair”*, PhD thesis, Université Paris Sud, June 2003.

62 O. Lodygensky. “*Contribution aux Infrastructures de Calcul Global: Délégation Inter plates-formes, Intégration de Services Standards et Application à la Physique des Hautes energies”*, PhD thesis, Université Paris Sud, September 2006.

63 J. Pruyne and M. Livny. “A Worldwide Flock of Condors: Load Sharing Among Workstation Clusters”. *Future Generations of Computer Systems journal (FGCS)*, 12, 1996.

64 D. Anderson. “BOINC: Berkeley Open Infrastructure for Network Computing”, *Technical Report Univ. California Berkeley*, 2002.

65 F. Cappello, S. Djilali, G. Fedak, et al. “Computing on Large Scale Distributed Systems: XtremWeb Architecture, Programming Models, Security, Tests and Convergence with Grid”, *FGCS Future Generation Computer Systems*, 2004.

66 I. Foster, C.Kesselman. “*The Grid: Blueprint for a New Computing Infrastructure*”, Morgan Kaufmann Publishers, Inc., San Francisco, CA, 1999.

67 J.-S. Kim, B. Nam, M. Marsh, P. Keleher, B. Bhattacharjee, D. Richardson, D. Wellnitz, and A. Sussman. “Creating a Robust Desktop Grid using Peer-to-Peer Services”. In *Proceedings of the 2007 NSF Next Generation Software Workshop (NSFNGS 2007)*, Mar. 2007.

68 A. Iamnitchi and I.T. Foster. “A Peer-to-Peer Approach to Resource Location in Grid Environments”. In: J. Weglarz, J. Nabrzyski, J. Schopf and M. Stroinski, Editors, *Grid Resource Management*, Kluwer (2003).

69 P. Cozza, I. Kelley, C. Mastroianni, D. Talia and I. Taylor. “Cache-Enabled Super-Peer Overlays for Multiple Job Submission on Grids”. In *Proc. of the CoreGRID Workshop on Grid Middleware*, Dresden, Germany, June 2007.

70 Baohua Wei, G. Fedak and F. Cappello. “Collaborative Data Distribution with BitTorrent for Computational Desktop Grids*”. ISPDC'05* Lille, France, 2005.

71 G.C. Silaghi, A.E. Arenas, Luis M. Silva. “Reputation-based Trust Management Systems and their Applicability to Grids”. Technical report, TR-0064, *Institutes on Knowledge and Data Management & System Architecture, CoreGRID Network of Excellence*, February

72 T. Grandison, M. Sloman. “A Survey of Trust in Internet Applications” *IEEE Communications Survey and Tutorials*, 3(4), pp. 2-16, 2000.

73 A. Josang, R. Ismail, C. Boyd. “A Survey of Trust and Reputation Systems for Online Service Provision”. *Decision Support Systems*, 43(2), pp 618-644, 2007.

74 I. Foster, C. Kesselman, S. Tuecke. “The Anatomy of the Grid: Enabling Scalable Virtual Organizations”. *International Journal of Supercomputing Applications* 15(3), pp. 200-222, 2001.

75 N. Nagaratnam, P. Janson, J. Dayka, A. Nadalin, F. Siebenlist, V. Welch, S. Tuecke, I. Foster. “Security Architecture for Open Grid Services”, 2002. Available at http://forge.gridforum.org/projects/ogsa-sec-wg/

76 D.S. Wallach. “A Survey of Peer-to-Peer Security Issues”. In *Proceedings of the International Symposium on Software Security - Theories and Systems*, Tokyo, Japan. Springer *Lecture Notes in Computer Science* 2609, pp. 253-258, 2003.

77 M. Cannataro, D. Talia, G. Tradigo, P. Trunfio, P. Veltri “SIGMCC: A System for Sharing Meta Patient Records in a Peer-to-Peer Environment”, *Future Generation Computer Systems*, 24(3), pp. 222-234, 2008.

78 P. Domingues, B. Sousa, L.M. Silva. “Sabotage-Tolerance and Trust Management in Desktop Grid Computing”. *Future Generation Computer Systems*, 23(7), pp. 904-912, 2007.

79 G.C. Silaghi, L.M. Silva, P. Domingues, A.E. Arenas. “Tackling the Collusion Threat in P2P-Enhanced Internet Desktop Grids”. In *Proceedings of the CoreGRID Workshop on Grid Programming Model, Grid and P2P Systems Architecture, Grid Systems, Tools and Environments*, Heraklion, Greece, June 2007.
80 G.C. Silaghi, F. Araujo, L.M. Silva, P. Domingues, A.E. Arenas. “Defeating Colluding Nodes in Desktop Grid Computing Platforms”. Technical report, TR-0124, *Institute on Architectural issues: Scalability, Dependability, Adaptability, Institute on Knowledge and Data Management , CoreGRID Network of Excellence*, 2008.
81 A.C. Marosi, G. Gombás, Z. Balaton, P. Kacsuk. “SZTAKI Desktop Grid: Building a Scalable, Secure Platform for Desktop Grid Computing**”**. Technical report, TR-0100, *Institute on Architectural Issues: Scalability, Dependability, Adaptability, CoreGRID Network of Excellence*, August 2007.
82 A. Andrzejak and M. Ceyran. Characterizing and Predicting Resource Demand by Periodicity Mining. Journal of Network and System Management, Vol. 13, No. 1, Mar 2005.
83 A. Andrzejak, A. Reinefeld, F. Schintke, and T. Schütt. On Adaptability in Grid Systems. In Proceedings of Dagstuhl Seminar 04451, Future Generation Grids, November 2004, V. Getov, D. Laforenza, and A. Reinefeld (eds.), Springer CoreGRID Series, 2006.
84 A. Andrzejak, U. Hermann, and A. Sahai. Feedbackflow - An Adaptive Workflow Generator for System Management, 2nd IEEE International Conference on Autonomic Computing (ICAC-05), 2005.
85 M. Broy and R. Steinbrüggen. Modellbildung in der Informatik. Springer-Verlag, Berlin, 2004, ISBN 3-540-44292-8.
86 N. Damianou, A. K. Bandara, M. Sloman, and E. C. Lupu. A Survey of Policy Specification Approaches, 2002.
87 N. Damianou, N. Dulay, et al. The Ponder Policy Speci_cation Language. Policy 2001: Workshop on Policies for Distributed Systems and Networks, Bristol, UK, Springer-Verlag, 2001.
88 Distributed Management Task Force (DMTF). DMTF CIM Concepts White Paper. http://www.dmtf.org/standards/published_documents.php
89 M. Ghallab, D. Nau, and P. Traverso. Automated Planning - theory and practice. Morgan Kaufmann Publishers, 2004, ISBN 1-55860- 856-7.
90 T. Glad and L. Ljung. Control Theory: Multivariable and Nonlinear Methods. CRC Press, June 2000.
91 J. Han and M. Kamber. Data Mining: Concepts and Techniques. Morgan Kaufmann Publishers, 2001.
92 International Telecommunication Union (ITU). Specification and description language (SDL). TU-T Recommendation Z.100, August 2002.
93 The Internet Society. RFC 3198 - Terminology for Policy-Based Management. 2001.
94 J.O. Kephart and D.M. Chess. The vision of autonomic computing. IEEE Computer, Jan. 2003, pp. 41_50.
95 S. Makridakis, S. C. Wheelwright, and R. J. Hyndman. Forecasting - Methods and Applications. 3rd edition, John Wiley & Sons, Inc., 1999.
96 A. Reinefeld, F. Schintke, and T. Schütt. Scalable and Self-Optimizing Data Grids. Chapter 2 (pp. 30 - 60) in: Yuen Chung Kwong (ed.), Annual Review of Scalable Computing, vol. 6, June 2004.
97 I. Foster, C. Kesselman, J. Nick, S. Tuecke. “The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration”, Technical Report, Open Grid Service Infrastructure WG, Global Grid Forum, 2002.
98 OGSA, http://www.globus.org/ogsa/
99 I. Foster. ”Globus Toolkit Version 4: Software for Service-Oriented Systems”. IFIP International Conference on Network and Parallel Computing, Springer-Verlag LNCS 3779, pp 2-13, 2005.
100 C. Pautasso, G. Alonso. “Parallel Computing Patterns for Grid Workflows”, HPDC2006 Workshop on Workflows in Support of Large-Scale Science, France, 2006.
101 K. Krauter, R. Buyya, and M. Maheswaran.“A Taxonomy and Survey of Grid Resource Management Systems for Distributed Computing”. Software: Practice and Experience (SPE), ISSN: 0038-0644, Wiley Press, USA, pp. 32(2):135-164, February 2002.
102 Q. Morante, N. Ranaldo, A. Vaccaro, and E. Zimeo, “Pervasive Grid for Intensive Power System Contingency Analysis”, IEEE Transactions on Industrial Informatics, 2(3), pp. 165-175, 2006.
103 N. Ranaldo and E. Zimeo. “A Framework for QoS-based Resource Brokering in Grid Computing”. In C. Pautasso and T. Gschwind, Editors, Preliminary Proceedings of the 5th IEEE European Conference on Web Services, the 2nd Workshop on Emerging Web Services Technology, Halle, Germany, 2007.
104 N. Ranaldo, E. Zimeo. “An Economy-driven Mapping Heuristic for Hierarchical Master-Slave Applications in Grid Systems”. IEEE IPDPS’06, Greece, 2006.
105 R. Zhang, I.B. Arpinar, B. Aleman-Meza. “Automatic Composition of Semantic Web Services. ICWS, pp. 38-41, 2003.
106 A. Forestiero, C. Mastroianni, H. Papadakis, P. Fragopoulou, A. Troisi, E. Zimeo, “A Scalable Architecture for Discovery and Composition in P2P Service Networks, Grid Computing: Achievements and Prospects, 2008.