A Simulation Model for Mechanisms, Heuristics and Rules for P2P Systems

Ciprian Dobre, Florin Pop, and Valentin Cristea

Abstract Large-scale P2P distributed resources that aggregate and share resources over wide-area networks present major new challenges for scientists. This chapter focuses on the challenges to enable scalable, high-level simulation of applications, middleware, resources and networks to support scientific and systematic study of large scale distributed applications and environments, specifically designed for P2P systems. Its contributions are on the intelligent control of distributed P2P systems, where simulation is used in decision making as a way to predict future performance under some control law in question.

1 Introduction

Over the Internet today, computing and communications environments are more complex and chaotic than classical distributed systems, lacking any centralized organization or hierarchical control. Peer-to-Peer systems provide a good substrate for creating, for example, large-scale data sharing, content distribution and application-level multicast applications [23]. In the last years, extensive research was conducted towards overcoming scaling problems with unstructured P2P net-works, such as Gnutella, where data placement and overlay network construction are essentially random. Still, evaluation of such solutions is hard to achieve because of the size of the systems and the nondeterministic nature of computing and data resources involved [4].

Ciprian Dobre
University POLITEHNICA of Bucharest, Romania, e-mail: ciprian.dobre@cs.pub.ro

Florin Pop
University POLITEHNICA of Bucharest, Romania e-mail: florin.pop@cs.pub.ro

Valentin Cristea
University POLITEHNICA of Bucharest, Romania e-mail: valentin.cristea@cs.pub.ro

The field of modeling and simulation was long-time seen as a viable alternative to develop new algorithms and technologies and to enable the development of large-scale distributed systems (LSDS), where analytical validations are prohibited by the nature of the encountered problems. The use of discrete-event simulators in the design and development of LSDS is appealing due to their efficiency and scalability. Their core abstractions of process and event map neatly to the components and interactions of modern-day distributed systems and allow the design of realistic scenarios. Compared with the alternative of implementing the new technology directly in real-world to demonstrate its viability, the simulation of distributed systems is a far better alternative because it achieves faster validation results, minimizing the costs involved by the deployment process [9].

The focus of this chapter is on the modeling and simulation of large scale P2P systems as a support for *intelligent control of distributed P2P systems*. We describe alternatives to designing and implementing simulation instruments to be used in the validation of distributed P2P technologies. The chapter is based on the experience accumulated by the authors in developing a generic model for the simulation of distributed system technologies, integrating components and mechanisms to create realistic simulation experiments of large scale systems. The original model proposed in the MONARC simulator incorporates all the necessary components and characteristics that allow the complete and accurate design of realistic simulation experiments of complex distributed architectures, consisting of many resources and various technologies, ranging from data transferring to scheduling and data replication, with resources working together to provide a common set of characteristics [10].

The chapter presents the design characteristics of a simulation model that extends the original model of MONARC to allow evaluating P2P mechanisms and technologies. We present the characteristics of P2P systems that influenced the design process of the proposed model. We demonstrate that it includes the necessary components to describe various actual distributed P2P technologies, and provides the mechanisms to describe from network overlays, to the evaluation of different strategies in job scheduling procedures [2] and intelligent algorithms. As a case study, we demonstrate that the proposed simulation model contains the necessary components and characteristics to allow complete and accurate design of specific P2P simulation experiments. We present experiments designed to evaluate scheduling procedure. We also present a simulation study designed to evaluate the properties of several network overlays.

We also present a critical comparison study of the most important simulation projects involved in the modeling of P2P systems, based on existing research [29]. The analysis highlights their specific characteristics, from the types of simulation models or internal simulation design to the implementations of the respective instruments. We demonstrate that, although the use of a particular simulation instrument depends very much on the scope of the simulation being conducted and the skills of the user, they all cover important aspects of distributed systems, allowing exploration of different areas of parameter space. Among these projects, we demonstrate that the model proposed in MONARC is the most generic one, being capable to handle a wider range of simulation scenarios. It also offers the highest number of capabilities, being able to consider many parameters, capabilities and components that any other existing simulation instrument for P2P systems.

The chapter is organized as follows. Section 2 presents open issues for large scale P2P systems and Section 3 the mechanism for performance optimization in large scale P2P systems. Then Section 4 presents the simulation models and proposed tools for LSDS. Section 5 describes the general framework of MONARC and in Section 6 the MONARC extension for P2P is highlighted. Some experimental evaluations are presented in section 7. The conclusions are presented in the last section of this chapter.

2 P2P Issues and their Influence on the Simulation Model

This section presents specific characteristics of Large Scale P2P systems, and their successful integration into the proposed simulation model. Such particular aspects were previously identified by authors [13] as having a major impact on the effectiveness and deployment of P2P systems and applications:

- *Decentralization*.
- *Scalability*.
- *Anonymity*.
- *Self-organization*.
- *Cost of ownership*.
- *Ad-hoc connectivity*.
- *Performance*.
- *Security*.
- *Transparency*.
- *Usability*.
- *Fault-resilience*.
- *Interoperability*.

These characteristics of the P2P systems positively influenced the development process of the simulation model. Various studies also identified specific elements of the simulation model that enables the correct modeling of an LSDS environment. Authors of the study presented in [35] present the features that must be implemented by a simulation model in order to allow the correct modeling of a Grid environment. The identified set of features consists of: *multi-tasking IT resources*, *job decomposition*, *task parallelization*, *heterogeneous resources*, *resource scheduling*, and *resource provisioning*. We present the identified feature of a generic P2P environment simulator model.

Multi-tasking IT resources: Processors, database servers, network links, and data storage devices are all pre-emptive multi-tasking resources. Tasks submitted to such a resource go immediately to the processing queue containing all other tasks that are being processed by the resource. Each of these tasks get processed by the resource for a pre-defined time-slice and then put back to the processing queue (if not waiting for other resources). As a result of multi-tasking, the submission of a new task can change the completion time of existing tasks being processed by the resource. The simulation model incorporates processing units, database servers, network links, and data storage devices. The processing unit has a queue of all the tasks being concurrently processed by the resource. An interrupt mechanism ensures modeling of tasks concurrency. The time needed to complete a task takes into account the number of other competing tasks. In this way the multi-tasking requirement is one of the characteristics implemented in the simulation model. A detailed simulation of the task management within each resource is far too time-consuming when considering grid environments with hundreds of resources simulated for the duration of weeks. Instead, when a new task is submitted to a resource, the simulation framework performs a good approximation of the multi-tasking behavior by re-estimating the completion time of all tasks being processed by that resource.

Job decomposition: Each job from a workload may be composed of multiple resource requirements. The terminology of "task" is generally used to define a single resource requirement within a job. Each resource requirement may be for a specific configuration of a server or a database. For example, a job may be composed of three tasks: get data from a customer database, perform data mining on a compute server, and add the results to a sales database. Job decomposition is important to model because grid designs typically focus on providing a certain type of resource (e.g., compute servers) on a grid and overlook the impact on other resources (e.g., databases, network bandwidth) needed by the job. The simulated job is programmed to handle specific actions. A job can, for example, be programmed by the user to request data from a database server, to perform some computation using the obtained records and then send the results for further processing to another job. The tasks performed by a job can be correlated with the tasks performed by another one. The dependencies between the jobs can be specified in the form of DAG structures in the simulation model.

Task parallelization: Each task in a job (decomposed as described above) may be parallelizable. This is often the case with grid workloads. Parallelization could be of two types: "Embarrassingly parallel" tasks can simply be split up into as many chunks as available resources. The second category consists of parallel tasks that are more constrained and consist of a specific number of parallel paths, regardless of the number of available resources. A simulated job can start new simulated jobs, each one performing specific tasks. This, correlated with the job decomposition characteristic implemented using the DAG structures, can be used to handle both described cases of task parallelization.

Heterogeneous resources: Grid resources can be of various vendor platforms, models, and operating systems. Therefore, the processing time of a task on a resource is subject to its performance benchmarks. A resource may be associated with multiple performance benchmarks that are relevant for different types of computing tasks. The computing unit specified in computing power in the form of generic SI95 units, accommodating for a wide-range of performance benchmarks. The computation, data and network models all take into account resources with various amounts of resources to model the heterogeneity of resources in the simulation experiments.

Resource scheduling: The grid simulator must be able to model scheduling policies as used by various resource brokers to determine which resource should process an arriving task. For that, the simulation model provides a meta-scheduling functionality, allowing the execution the simulated jobs in a distributed manner. The meta-scheduler can incorporate a wide-range of user-defined scheduling algorithms. Local scheduling algorithms can also be added in any user-defined experiments.

Resource provisioning: The ability to provision resources for processing particular types of tasks is another feature to be simulated. These provisioning policies could be either calendar based (e.g., a department needs a server to be an email server during 9AM to 5PM every workday and can release it to be a data-mining processor on the grid at other times) or based on a more dynamic policy that monitors workload arrivals and resource usage and reacts accordingly. Simulation of such provisioning policies in conjunction with the resource scheduling policies would allow grid designers to determine whether the respective policies are aligned and consistent with each other. Also, it would provide the desired grid performance in terms of resource availability, workload throughput and processing times. This is why the simulation model allows the insertion of background jobs to handle the execution of specific resource provisioning tasks. Such jobs can be used in dependency with other simulated jobs. In addition, the database server can perform programmed actions, being modeled as a special task in the simulation model. It can simulate special operations such as data archiving on a calendar based designed policy. The automation of resource provision is particularly important to the simulation model because it can be used to experiment with various data replication algorithms and provide flexibility to the simulation scenarios [3]. In traditional client-server models the information is concentrated in centrally located servers and distributed to client computers and, for example, access rights and security are more easily managed. The centralized systems topology yields inefficiencies, bottlenecks, and wasted resources and even if hardware performance and cost have improved, centralized repositories are expensive to set up and hard to maintain.

One of the more important ideas of *decentralization* is the emphasis on the usersównership and control of data and resources. In a fully decentralized system, every peer is an equal participant making the implementation of the P2P models difficult because there is no centralized server with a global view (see Fig. 1). This is why many P2P file systems are built as hybrid approaches. This is the of Napster, where there is a centralized directory of the files but the nodes download files directly from their peers [11].

In fully decentralized file systems (Freenet, Gnutella), finding the network becomes difficult. In Gnutella, new nodes must know the address of another node or use a host list with known IP addresses of other peers. The node joins the network of peers by establishing a connection with at least one peer currently in the network. Then, it can begin discovering other peers and cache their IP addresses locally. One method to categorize the autonomy of a P2P system is through the “pure P2P” versus “hybrid P2P” distinction [28].

An immediate benefit of decentralization is improved *scalability*. Scalability is limited by factors such as the amount of performed centralized operations (e.g, *synchronization* and *coordination*), the amount of maintained state, the inherent parallelism and the programming model that is used to represent the computation. Achieving good scalability should not be at the expense of other features, such as determinism and performance guarantees. For instance, hybrid P2P systems, such as Napster, keep some amount of the operations and files centralized.

Searching may fail even when an object exists, making the behavior of the system nondeterministic. Systems such as CAN, Chord, Oceanstore, and PAST, dictate a consistent mapping between an object key and hosting node [33]. Thus, an object can always be retrieved as long as the hosting nodes can be reached. Nodes in these systems compose an overlay network, where each node maintains information about a small number of other nodes. This approach limits the amount of state that needs to be maintained. The logical topology of the overlay provides some guarantees on the lookup cost.

A goal is to allow people to use systems without concern for legal or other ramifications and to guarantee that censorship of digital content is not possible. There are three kinds of *anonymity* involved in communicating pair: sender anonymity, which hides the senderś identity; receiver anonymity, which hides a receiverś identity; and mutual anonymity, which hides the identities of the sender and receiver, are hidden from each other and other peers [18].

It is important to understand the degree of anonymity a certain technique can achieve. There is a spectrum of anonymity degrees that cover absolute privacy, be-

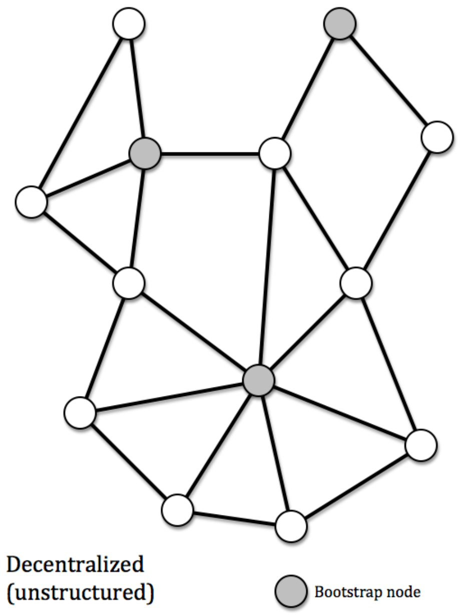

Fig. 1 Unstructured overlay: an overlay is represented as a graph that describes how the nodes are connected with each other.

yond suspicion, probable innocence and provably exposed. Beyond suspicion means that even though an attacker can see evidence of a sent message, the sender appears no more likely to be the originator of that message than any other potential sender in the system. There are six techniques for enforcing different kinds of ano-nymity with different kinds of constraints:

- *Multicasting* (or *broadcasting*) can be used to enforce receiver anonymity. An entity that is interested in obtaining a document subscribes to the multicast group and its identity is hidden for the sender and other members of the group. The party that possesses the document sends the document to the group. This technique can take advantage of the underlying network that supports multicast (e.g., Ethernet or token ring) [21].
- *Spoofing* the senderś address. For connectionless protocols such as UDP, the senderś anonymity can be enforced by spoofing the senderś IP address but this requires the protocol changing and this is not always feasible, because most ISPs now filer packets originating from invalid IP addresses.
- *Identity Spoofing*. Anonymity can be ensured by changing the identity of a communicating party. For example, in Freenet, a peer passing a file to a requestor can claim to be the owner of the content. The responder is possibly innocent, from an attackerś point view, because there is a nontrivial probability that the real responder is someone else ([34]).
- *Covert paths*. Instead of communicating directly, two parties communicate through some middle nodes. A party that wishes to hide its identity prepares a covert path with the other party as the end of the path. The covert paths can use store/forward or persistent connection. By varying the length of the covert paths and changing the selected paths with different frequency, different degrees of anonymity can be achieved.
- *Intractable aliases*. The client can open an account and be recognized upon returning to the opened account, while hiding the true identity of the client from the server. Techniques of this kind ensure sender anonymity and rely on a trusted proxy server. The degree of anonymity that can be achieved falls in between absolute privacy and beyond suspicion.
- *Non-voluntary placement*. A publisher forces a document onto a hosting node using, for example, consistent hashing and because the placement is non-voluntary, the host cannot be held accountable for owning the document.

In P2P systems, *self-organization* is needed because of scalability, fault resilience, intermittent connection of resources, and the ownership cost. Those systems scale unpredictably in terms of the number of systems or users, and the load, causing an increased probability of failures, requiring self-maintenance and self-repair (see Fig. 2 for different topologies properties) [25].

Similar reasoning applies to intermittent disconnection because it is hard for any predefined configuration to remain intact over a long period of time. Adaptation is required to handle the changes caused by peers connecting and disconnecting from the P2P systems. It is costly to have dedicated equipment for managing such a fluctuating environment, so the management is distributed among the peers.

Shared ownership reduces the cost of owning the systems and the content, and the maintaining cost. This is applicable to all classes of P2P systems but it is most obvious in distributed computing. For example, SETI@home is faster than the fastest supercomputer in the world, yet at only a fraction of its cost [12].

The ad-hoc nature of connectivity has a great impact and P2P systems and applications need to be able to handle systems joining and withdrawing from the pool of available systems. For example, in distributed computing, the parallelized applications cannot be executed on all systems all of the time because some of the systems will be available all of the time, some will be available part of the time, and some will be not be available at all.

In systems with higher guarantees, such as service-level agreements, the ad-hoc nature is reduced by redundant service providers. In collaborative P2P systems and applications, the ad-hoc nature of connectivity is evident because collaborative users are increasingly expected to use mobile devices, making them more connected to Internet and available for collaboration. To handle this situation, collaborative systems support transparent delay of communication to disconnected systems, having proxies delegated on networks to receive messages, or having other sorts of relays on the sending system or somewhere in the network that will temporarily hold communication for an unavailable system. P2P systems and applications need to be designed to tolerate sudden disconnection and ad-hoc additions to groups of peers. The P2P architectures taxonomy is presented in Fig. 3.

P2P systems aim to improve *performance* by aggregating distributed storage capacity and computing cycles of devices spread across a network. Performance is influenced by three types of resources: processing, storage, and networking. Networking delays can be significant in wide area networks. Bandwidth is a major factor when a large number of messages are propagated in the network and large amounts of files are being transferred among many peers. This limits the scalability of the system.

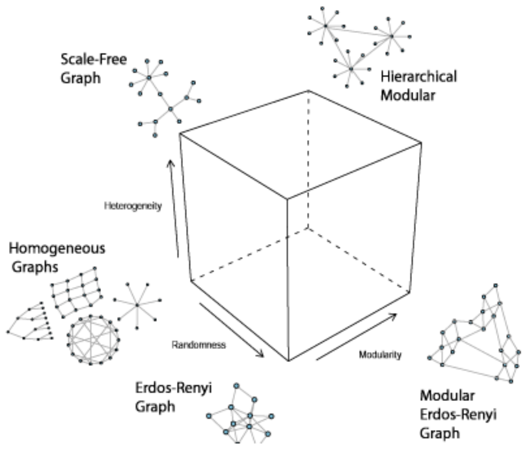

Fig. 2 The space of network topologies properties. (Inspired by [25]).

In centrally coordinated systems (e.g., Napster, Seti@Home) coordination between peers is controlled by a central server, although the peers also may later contact each other directly. These systems are vulnerable to the problems facing when using centralized servers. To overcome such limitations, different hybrid P2P architectures have been proposed to distribute the functionality of the coordinator in multiple indexing servers that cooperate with each other to satisfy user requests. DNS is an example of a hierarchical P2P system that improves performance by defining a tree of coordinators, with each coordinator responsible for a peer group. Communication between peers in different groups is achieved through a higher level coordinator.

In decentralized coordinated systems, such as Gnutella and Freenet, there is no central coordinator and communication is handled individually by each peer. They use message forwarding mechanisms search for information and data, but they end up sending a large number of messages over many hops from one peer to another. Each hop contributes to an increase in the bandwidth on the communication links and to the time required to get results for the queries. The bandwidth for a search query is proportional to the number of messages sent, which in turn is proportional to the number of peers that must process the request before finding the data.

3 Performance Optimization in Large Scale P2P Systems

There are three key approaches to optimize performance of large scale P2P systems: *replication*, *caching* and *intelligent routing* [17].

Replication puts copies of objects/files closer to the requesting peers, minimizing the connection distance between the peers requesting and providing the objects. Changes to data objects have to be propagated to all the object replicas. In combination with intelligent routing, replication minimizes the distance delay by sending requests to closely located peers. Replication also helps to cope with the disappearance of peers.

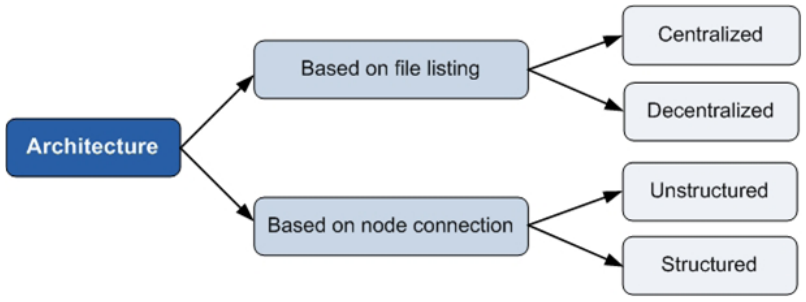

Fig. 3 P2P Architectures Taxonomy.

Caching reduces the path length required to fetch a file/object and therefore the number of messages exchanged between the peers. Reducing such transmissions is important because the communication latency between the peers is a serious performance bottleneck facing P2P systems. Efficient caching strategies can be used to cache large amounts of data infrequently. The goal of caching is to minimize peer access latencies, to maximize query throughput and to balance the workload in the system. The object replicas can be used for load balancing and latency reduction.

Intelligent routing and *network organization*. To fully realize the potential of P2P networks, it is important to understand and explore the social interactions between the peers. Adamic ([1]) explored the power-law distribution of the P2P networks, and introduced local search strategies that use high-degree nodes and have costs that scale sub-linearly with the size of the network. [22] determine "good" peers based on interests, and dynamically manipulate the connections between peers to guarantee that peers with a high degree of similar interests are connected closely. Establishing a good set of peers reduces the number of messages broadcast in the network and the number of peers that process a request before a result is found.

P2P systems came up with different naming and discovery schemes, but there is also a requirement for administration transparency. Users are typically non-experts and, thus, the P2P software should not require any significant set up or configuration. P2P systems should be network and device transparent. They should work on the Internet, intranets, and private networks, using high-speed or dial-up links. They should also be device transparent, which means they should work on a variety of devices, such as PDAs, desktops, and cell phones. Another form of transparency is related to security and mobility. Automatic and transparent authentication of users and delegation to user proxies can significantly simplify users' actions. Supporting mobile users and disconnection in particular, can enable users to work independently of whether and how they are connected to the Internet or intranets.

Some current P2P systems (e.g., Groove) provide special nodes, called relays, that store any updates or communication temporarily until the destination reappears on the network. Others (e.g., Magi) queue messages at the source, until the presence of the destination peer is detected. Another problem is non-availability of resources. This may occur either because the resource is unreachable because of a network failure or because the peer hosting the resource has crashed/gone offline. While the former may be resolved by routing around the failure and is already supported by the Internet, the latter requires more careful consideration. Replication of crucial resources helps alle-viate the problem. A challenging aspect of P2P systems is that the system maintenance responsibility is completely distributed and needs to be addressed by each peer to ensure availability, different from client-server systems, where availability is a server-side responsibility.

In the past, there were different ways to approach interoperability, such as standards IEEE, common specifications, and common source code, open-source and de facto standards. In the P2P space, some efforts have been made to improve interoperability, but interoperability is still not supported. The P2P Working Group is an attempt to gather the community of P2P developers together and establish common ground by writing reports and white papers that would enable common understanding among P2P developers.

The JXTA effort approaches interoperability as an open-source effort, by attempting to impose a de facto standard. A number of developers are invited to contribute to the common source tree with different pieces of functionality. Only a minimal underlying architecture is supported as a base, enabling other systems to contribute parts that may be compatible with their own implementations. A number of existing P2P systems have already been ported to the JXTA base [14].

4 Simulation Models and Tools for LSDS

This section analyses existing work in constructing methodologies for analyzing and comparing various simulation tools targeting LSDS.

The various properties of any simulator designed to handle LSDS related technologies fall in two main categories: the *simulation taxonomy* and the *design taxonomy*. The simulation taxonomy analysis the simulation tools according to the adopted simulation models, while the design taxonomy categorizes the simulation tools according to their design and implementation.

The *simulation taxonomy* comprises five properties, as presented in the Fig. 4. Motivation indicates the major target of a simulation tool. According to this taxonomy a modeling instrument can be used to study various scheduling algorithms, to study various replication and data movement optimizations, or to study a particular model of an LSDS system. This category considers only the upper most motivation. If, for example, we consider the case of a simulation tool designed specifically to study various scheduling algorithms, we can observe that for this case the simulator must also provide additional support, such as simulated underlying networks or processing nodes. If the underlying LSDS components are also simulated, then a scientist could also evaluate various other file replication algorithms (assuming the possibility to also simulate data warehouses for example). But generally such modifications require great amount of work, and, except for the cases when the original developers of a simulator redesigned it at some point to comprehend some different classes of Grid related problems, such developments are virtually non-existent. The majority of the Grid simulation projects were developed in the context of the validation of the LHC experiments and their proposed running conditions. For this reason we often see as possible motivation the categories identified by the authors of the taxonomy described in [36]: Data Transport, Data Replication or Scheduling issues.

The *behavior taxonomy* classifies the modeling instruments based on how the simulation proceeds. A deterministic simulation has no random events occurring, so repeating the same simulation will always return the same simulation results. In contrast, a probabilistic simulation has random events occurring, so repeating the same simulation often returns different simulation results [15, 8, 9].

The simulators for LSDS must be designed according with criteria grouped under the following headings:

- Simulator Architecture
- Usability
- Scalability
- Statistics
- Underlying Network Simulation
- System Limitations

Using the categories of the proposed presented taxonomy, in this section we present an analysis of the properties of most representative modelling instruments for LSDS. One goal of this study is to present the most relevant related work in the field of modelling and simulation of distributed systems. One other objective of the analysis is to evaluate on a real case scenario the capabilities of the presented taxonomy to correctly investigate the capabilities of various simulation instruments for LSDS systems.

DHTSim is a discrete event simulator for structured overlays, specifically DHTs. It is intended as a basis for teaching the implementation of DHT protocols, and as such it does not include much functionality for extracting statistics. It is implemented as discrete event based message passing within the JVM [26]. *P2PSim* is a discrete event packet level simulator that can simulate structured overlays only. It contains implementations of six candidate protocols: Chord, Accordion, Koorde, Kelips, Tapestry and Kademlia. *Overlay Weaver* is intended to be a toolkit for easy development and testing of P2P protocols. It provides functionality for simulating structured overlays only and does not provide any simulation of the underlying network [20]. *PlanetSim* is a discrete-event overlay network simulator written in Java. It supports both structured and unstructured overlays, and is packaged with Chord-SIGCOMM and Symphony implementations. *SimGrid* [16] is a simulation toolkit that provides core functionalities for the evaluation of scheduling algorithms in distributed applications in a heterogeneous, computational Grid environments mainly. It aims at providing the right model and level of abstraction for studying Grid-based scheduling algorithms and generates correct and accurate simulation results. *GridSim* [24] is a grid simulation toolkit developed to investigate effective resource allocation techniques based on computational economy. *OptorSim* [32] is a Data Grid simulator designed for evaluating optimization in data access technologies for Grid environments. It adopts a Grid structure based on a simplification of the architecture proposed by the EU DataGrid project. *ChicagoSim* [19] is a simulator designed to

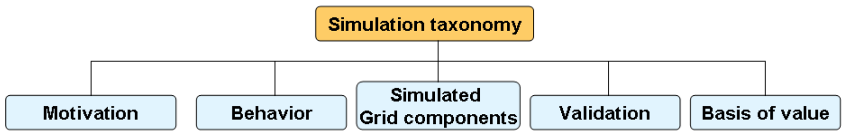

Fig. 4 The simulation taxonomy.

investigate scheduling strategies in conjunction with data location. It is designed to investigate scheduling strategies in conjunction with data location.

A summary report of presented simulation tools and others are provided in Table 1 (web references are also available for each simulator).

Table 1 Analysis of simulation tools.

Simulator	Support	Scalability (max nodes)	Reference
Peersim	Structured and unstructured overlays	10^6	http://peersim.sourceforge.net/
P2PSim	Structured overlay	3000	http://pdos.csail.mit.edu/p2psim/
Neurogrid	Structured and unstructured overlays	≤ 20	http://www.neurogrid.net/
Narses	Unstructured overlays	≥ 3	http://sourceforge.net/projects/narses/
Querycycle	Unstructured overlays	-	http://p2p.stanford.edu/
GPS	Structured and unstructured overlays	-	http://www.cs.binghamton.edu/ wyang/gps/
DHTSim	Structured overlay	-	http://www.swt.hs-mannheim.de/dhtsim/
Overlay Weaver	Structured overlay	4000	http://overlayweaver.sourceforge.net/
Gnutellasim	Unstructured overlays	600	http://www.cc.gatech.edu/computing/compass/gnutella/
OMNeT++	Structured and unstructured overlays	1000	http://www.omnetpp.org/
Planetsim	Structured overlays	100000	http://projects-deim.urv.cat/trac/planetsim/
OverSim	Structured and unstructured overlays	100000	http://www.oversim.org/
SimGrid	Structured overlays	≥ 3	http://simgrid.gforge.inria.fr/
GridSim	Structured overlays	≥ 3	http://www.buyya.com/gridsim/
OptorSim	Structured overlays	≥ 3	http://edg-wp2.web.cern.ch/edg-wp2/optimization/optorsim.html

None of these simulators present general solutions to modeling dependability technologies for LSDS. They focus on providing evaluation methods for the traditional research in this domain, which up until recently targeted the development of functional infrastructures. However, lately, the importance of dependable distributed systems was widely recognized and this is demonstrated by the large number of research projects initiated in this domain. Our proposed simulation model aims to provide the means to evaluate a wide-range of solutions for dependability in LSDS [31].

Security in particular has never been properly handled by any of these projects be-fore. The only currently existing simulator that offers the possibility to evaluate security solutions designed for distributed systems is G3S (Grid Security Services Simulator) [27]. It aims to support various authentication mechanisms such X.509

certificates and Kerberos tickets. It also includes mechanisms for disseminating security threats, for evaluating various access control policies, etc. The simulator uses the simulation model proposed in GridSim. Similar to this model, we too support all the mechanisms found in G3S and several others. In addition we offer the possibility of evaluating security in a more general context, considering the entire context of distributed systems, with its specific characteristics.

In addition, our proposed simulation model targets the generic evaluation of dependable distributed systems. The model is able to simulate, for example, various faults occurring in such systems, which allows researchers experiment with abnormal behavior of any of the systemś components. This can be coupled with various security enforcement or fault recovery solutions. We argue that a correct evaluation of dependability in distributed systems should provide a complete state of the entire distributed system. Because of the complexity of the LSDS, involving many resources and many jobs being concurrently executed in heterogeneous environments, there are not many simulation tools to address the general problem of LSDS computing. The simulation instruments tend to narrow the range of simulation scenarios to specific subjects, such as scheduling or data replication. The simulation model provided by MONARC is more generic that others, as demonstrated in [10, 8]. It is able to describe distributed system technologies, and provides the mechanisms to describe concurrent network traffic, to evaluate different strategies in data replication, and to analyze job scheduling procedures [5].

5 MONARC Simulation Engine

MONARC (MOdels of Networked Analysis at Regional Centers) is a simulator for realistic evaluation of large distributed computing systems [7]. The MONARC engine is structured on several layers. The basic one provides all the components of the system and their interactions. The largest one is the regional center, described by its location on a map (latitude and longitude), and which contains a farm of processing nodes (central processing units), database servers, mass storage units and local and wide area networks. Each network is described by speed and by a wide area network or router that connects it. Each CPU is described by power, memory capacity, and address on network, name of the network where it is connected and the maximum speed of communication in the network. Another component that models the behavior of the application is the “Activity” object. It is used to generate jobs based on different scenarios. A job is another basic component, scheduled for execution on a CPU unit by a “Job Scheduler” objects. Over this basic layer, it is possible to build a wide range of models, with different level of complexity (one or multiple regional centers, each with different hardware configuration, depending on the system that we want to simulate).

With this structure it is possible to build a wide range of models, from the very centralized to the distributed system models, with an almost arbitrary level of complexity (multiple regional centres, each having different hardware configuration and possibly different sets of replicated data). The analysis of the characteristics of various LSDS architectures was essential in the design process of the simulation model. It influenced the decision on the type of components and interactions required to completely and correctly model various Grid related experiments. Table 2 explains the meanings of some of the components used by the simulation framework.

Table 2 Simulation concepts for P2P Systems.

COMPONENT	SHORT DESCRIPTION
Data Container	Contains Objects of a single type in defined range.
Data Base	Contains sets of Containers. It is associated with an AMS Server.
Data Base Catalogue	Provide the mechanism to locate any Object and the AMS server which can retrieve this object.
Mass Storage Unit	High capacity unit, but slow, connected in LAN. Performs data access (W/R) in an OODB Model. Handles data storage on local disks or tape.
CPU Node	Typical processing Node, having a defined processing power, memory, and I/O channel. Allows concurrent execution of multiple jobs.
I/O Link (Link Port)	Describes the quality of I/O connection of each component on LAN. Allows simultaneously multiple transfers. It is the basic entity where any simulated transfer can start off.
LAN	Specifies how individual components are connected in LAN.
WAN	Specifies the connectivity for the wide area network. An internet like naming scheme is used.
Router	Specifies a router used to connect two or more WANs together in the simulation.
Protocol	Describe the modality in which the transfer is simulated.
Farm	A set of CPU Nodes, AMS servers, Mass Storage Unit and a Job Scheduler.
Job	Specifies typical tasks used in the simulation.
Active Job	Used by simulation system to perform a user defined job. It is dynamically allocated to a CPU Node when load constrains are satisfied.
Activity	A generic object used as a loadable module which defines a set of jobs and how they are submitted for execution.

For constructing the P2P overlay, we decided to use a single regional center with many CPU, on each of them running an application composed of two jobs. This application represents one peer in the system. We had to define an activity to generate these jobs. The activity reads the graph of the peers from a configuration file and then, based on this graph, generates the jobs, each of them knowing the topology or at least a part of it (its neighbors). In such a simulation all CPU units belong to the same local area network, and normally, if node x wants to communicate with node y, even if in the graph no edge connects them, they transfer data directly. By constructing the overlay, two nodes communicate directly, if an edge exists between them, or indirectly, using the path between them, if no edge connects them.

The most important classes of MONARC, grouped based on their appurtenance to a particular simulation model, are presented in Figure 5. In the figure the classes are grouped on several packages. The classes comprising the simulation engine (in light grey) were presented in the previous section. The classes comprising the job model (in red) together with the scheduling model (in light green), the ones included in the data model (in green), and the ones belonging to the network model (in orange) are all presented in this section.

In the diagram, the activity represents the users of the system being modeled. They generate data processing jobs based on different scenarios. Each regional center can have one or more (or none) such activities submitting jobs to it. The farm entity handles the CPU units (processing nodes) of the regional center. The farm is an active object by itself. Its role is to wait for job submissions coming from the activity objects and to call the appropriate job scheduler to handle the execution. The CPUUnit describes the behavior of a computing station belonging to the modeled system. It is characterized by the amount of available CPU power (measured using the SPEC benchmark) and the amount of memory. The memory occupation can make use of a paging mechanism, if required. This ensures the correct modeling of the type of behavior specific to several existing operating systems. The amount of CPU power to be used is allocated to the processing jobs according to an indicated priority. The user can specify which jobs must be processed with a higher priority than others. This translates into more amount of power being allocated for some prioritized processing jobs.

An extension to the CPU unit is represented by the CPUCluster. This object models a cluster of CPU units. Besides CPU power and memory, it is also characterized

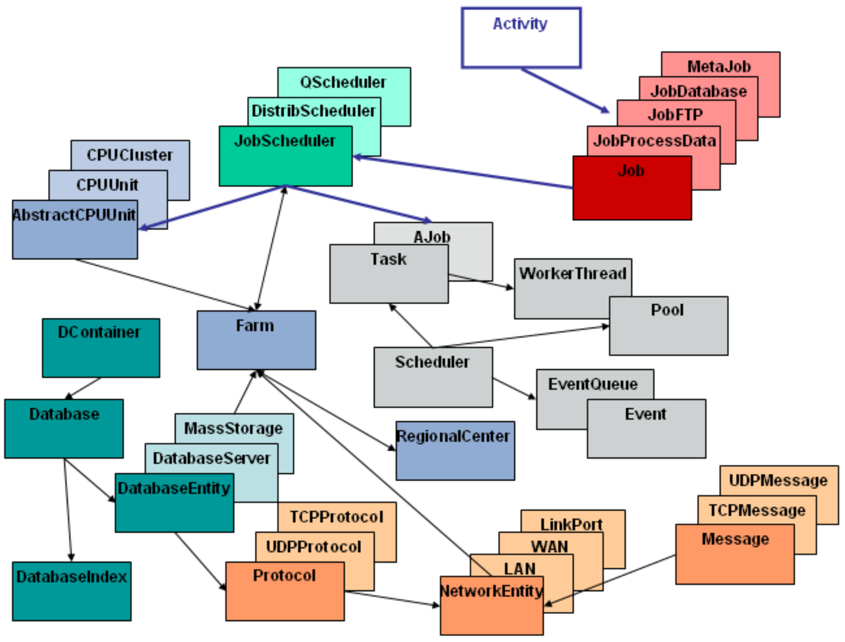

Fig. 5 The components of the MONARC simulation framework.

by the available disk space. This class can even simulate the failure of CPUs. The CPUCluster is well suited for modeling large LSDS environments, while the original CPUUnit object is more appropriate to model low scale distributed systems, consisting of only several nodes.

In MONARC, with every simulation step, the simulator executes the following operations in a loop described as follow:

```
1. Look at each simulation task and:
    a. If the task is in the created state, assign it to
    a worker thread from the pool and change the task’s
    state to ready
    b. If the task is in the ready state,  restart its
    execution by making a V() on the semaphore
    c. If the task is in the finished state, remove it
2. Wait until all the tasks that were running block again
or finish their execution
3. Process the events:
    a. Take from the future queue the event(s) with the
    minimum time stamp. The simulation time advances,
    becoming equal to that time stamp.
    b. For each event taken from the queue, look for the
    destination task. If it is waiting for an event (i.e.,
    it is in the waiting state), deliver the event to the
    task. Else, put the event into the deferred queue.
```

The distribution of jobs to the appropriate processing resources is being handled by a specialized JobScheduler. MONARC 2 provides two already defined algorithms for the scheduling of jobs, a centralized and a distributed one. We present these entities, as well as the details regarding the implementation of the data, network and task processing simulation models, in the next sections.

For instance, for a P2P overlay, in the next figure we illustrate a way of connecting five peers in a system:

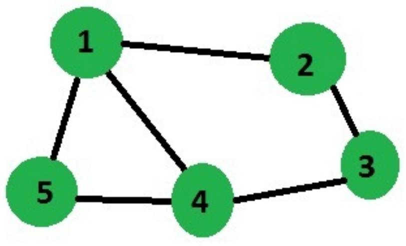

Fig. 6 Example of P2P overlay topology.

All nodes are connected in the same local area network. Normally, if *peer1* wants to communicate with *peer3*, it does it directly, because a link connects them. By constructing the overlay, we respect the given topology and make the information move only on the links associated with the edges in the graph. So, *peer1* transfers data to *peer3* through *peer2*, or through *peer4*, or through *peer5* and then *peer4*. The choice can be made depending on multiple factors, such as minimum number of hops, maximum link speed, minimum traffic (balanced traffic), and, generally, minimum cost associated with the link.

Each peer is defined by an application consisting of two jobs. First jobś purpose is to send and receive useful data, and also find a path based on minimum cost between two peers (e.g. peer1 and peer3 in the above example) and the second job's purpose is to send data between first kind of jobs (e.g. *peer2* for *peer1* and *peer3* in the above example). We will call first kind of jobs JobP2P and the second kind **JobRelay**. On CPU unit 1 runs, besides JobP2P 1, *JobRelay 1* which represents connecting node for other two peers (e.g. 2 and 5).

In the next section we discuss about two algorithms implemented for constructing this overlay. In the first algorithm, each JobP2P knows the entire topology of the system (the graph associated). It calculates locally the path to a remote peer where it wants to transfer data, based on a chosen criterion and then, sends the data through that path to the destination. *JobRelay* does not hold any information; its aim is to listen for messages from peers or from other relays and then send them to the next hop in the path. In the second algorithm each *JobPeer* knows only its neighbors. For finding a path to a destination, a peer broadcasts a message to the neighbors. If a neighbor does not know the route to the destination it broadcasts the request to all its neighbors. When a node holds information about the destination desired, it replies it unicast to the source that made the request. When the answer reaches the source peer, it can send the data to the remote peer selected. The second algorithm is appropriate to the idea of gossiping protocol as peers does not know the topology of all the system. Yet, both algorithms bring advantages and disadvantages and in the next section both topics will be discussed. We will choose the best algorithm in terms of performance, scalability and flexibility, over which the gossiping protocol will be implemented.

6 MONARC Extensions with an Overlay for P2P Systems

The discovery of the resources is simulated in MONARC by the resource catalogue, a hardware configuration file, which includes global parameters, such as the number of simultaneous threads and the name of the regional center, and detailed description of the center which includes its location, all the central processing units, the local, wide area network and the routers that it manages.

The descriptions of the resources are loaded at the beginning of the simulation. The configuration file also specifies the name of the Activity object used to generate data processing jobs based on our scenario. The Activity object loads the topology of the P2P system from a configuration file and overrides the *pushJobs* method according to the loaded simulation scenario. The file that specifies the graph contains the number of the nodes in the system and each of them is described by index, flag and color. Then, it follows edges definition, with the index of vertices connected by it, a color and a cost. The loader of topology has a method that returns the graph described in the file. The Activity object, in the overridden method *pushJobs*, generates, for each node in the graph, a *JobP2P* and a *JobRelay* objects. These objects receive, in the constructor, the node associated, and besides that, each *JobP2P* receives the topology. A P2P and a relay job associated with the same index of the node are running on the same CPU, each of them sending and receiving messages on different ports. Therefore, we will open four ports on a CPU, two of them for each different job running on it, one for sending and one for receiving. The *Activity* object also generates a *Job* responsible for closing the relays, as those jobs are waiting for messages and continue the transfer by forwarding them, in an infinite loop. In conclusion, when a P2P job finishes its processing and transfer, it sends a message to *JobCloseRelays* (which is running on a different CPU). After this job receives a number of messages equal to the number of nodes in the system, it sends a message with null data to each *JobRelay*, so they will take the decision to stop.

A *JobP2P* class has five methods: *computePath*, *sendData*, *receiveData*, *stopRelays* and *run*. It extends the basic *Job* class. The *computePath* method is used to find the path between the current node and a remote one, based on our topology. It receives as parameters the index of the destination and the name of the algorithm used to generate an optimal path. It returns the path according to the algorithm desired. The *sendData* method is used to simulate the sending of data to the next node in the path. It calls the *computePath* method to find the way to the destination. Then, a package is made, with the path and the data to transfer and is sent to the next hop. The relays unpack the data received, remove the current hop in the path and forward the message to the next node, until the message reaches its destination. The *receiveData* method is used to simulate the receive of data from a relay or a P2P neighbor. The *run* method overrides that one defined in the super class, and its responsibility is to send and receive data from remote P2P job. This method is overritten by the gossiping algorithm implemented in a class that extends *JobP2P*. Before the end of the *run* method, a message to the job responsible for closing the relay is send.

A *JobRelay* object receives a message, unpacks the transferred data, removes the first node from the containing path, and finally sends the new message to the next relay or P2P job (if the next hop in the way to the destination is the destination itself). It runs in a loop and stops only when a null data is received.

A *JobCloseRelay* waits for a number of messages equal to the number of nodes in the graph and then it sends a null data message to all relays in the system to stop running.

In Figure 7 we defined a topology for a P2P system. The rule of sending messages is node *x* to node *x+1* modulo the number of vertices. We will take for example two nodes which communicate directly, by being neighbors in graph, and two which communicate indirectly, with the help of relays met in the path to the destination. *JobP2P 1* (which runs on *CPU 1*, having the address *10.0.0.2*) sends the data directly to *JobP2P 2* (which runs on *CPU 2*, having the address *10.0.0.3*). The final sending makes use of the port calculated using the rule:

```
Port of P2P job, for sending = (index of job) + 3.
```

The receiving port for a P2P job is composed as:

```
Port of P2P job, for receiving = (index of job) + 1.
```

In this case *JobP2P 1* sends the data starting from *10.0.0.2*, port *4*, to *JobP2P 2*, on *10.0.0.3*, port *3*. *JobP2P 6* should send information to *JobP2P 1*, following the previously described rule. So, it locally calculates the optimal path to the destination, and considering the fact that each link costs the same value, the resulting way will be *(4, 2, 1)*. *JobP2P 6* will send a package consisting of the list of nodes to the destination, from which we removed the next hop *(4)* and the specific data. Therefore the package is *(2, 1) + data*. It will be send from *JobP2P 6* to *JobRelay 4*.

JobRelay listens for information on port:

```
Port of Relay job, for receiving = (index of job) + 2.
```

Therefore, *JobP2P 6* sends the data from *10.0.0.7*, port *9*, to *JobRelay 4*, on *10.0.0.5*, port *6*. *JobRelay 4* unpacks the information received in the message, sees that the next hop to the destination is *2*, removes it from the path, makes a new package with the remaining list and the useful data *((1) + data)* and transfers it to the *JobRelay 2*.

```
Port of Relay job, for sending = (index of job) + 4.
```

JobRelay 4 sends from *10.0.0.5*, port *8*, to *JobRelay 2*, on *10.0.0.3*, port *4*. *JobRelay 2* removes the remaining node from the list, checks it for emptiness and sends a new message with the data received to *JobP2P 1*, connected directly to node *2*.

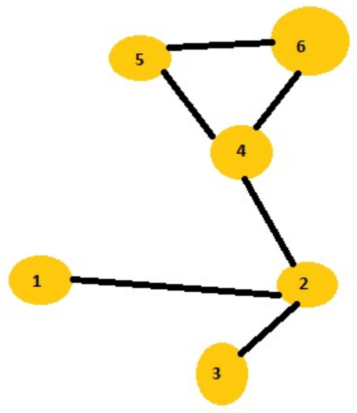

Fig. 7 Graph for the basic algorithm for overlay construction.

The disadvantage of this approach is the fact that each *JobP2P* object should keep in memory all the topology of the system, so the fact that the algorithm works or not depends strictly on the local memory capacity of each peer. Also, another drawback is the difference with how the gossiping protocol should be designed. In such protocols, a node cannot control and have knowledge about all the system from the beginning; the information is achieved in time. So, it is suitable for a node to keep in memory only the list of neighbors. These two disadvantages are removed with the algorithm described in the next section.

AODV Algorithm for Overlay Construction. The second algorithm is Ad hoc On-Demand Distance Vector (AODV) Routing, a routing protocol for mobile ad hoc networks (MANETs) and other wireless ad-hoc networks. It is a reactive routing protocol, meaning that it establishes a route to a destination only on demand. In contrast, the most common routing protocols of the Internet are proactive, meaning they find routing paths independently of the usage of the paths. AODV is, as the name indicates, a distance-vector routing protocol.

In AODV, the network is silent until a connection is needed. At that point the network node that needs a connection broadcasts a request for connection. Other AODV nodes forward this message, and record the node that they heard it from, creating an explosion of temporary routes back to the needy node. When a node receives such a message and already has a route to the desired node, it sends a message backwards through a temporary route to the requesting node. The needy node then begins using the route that came first to it.

Following the model described in the previous section, we also had to define an *Activity* for generate the jobs. In this case the relays have to receive information about direct neighbors, as they need to manipulate the routing table. When a *JobP2P* wants to transfer data to a remote node with which it is not connected, it makes a request to the *JobRelay* associated and asks for a route to the destination desired. If the relay does not have the answer, a request is broadcast to all the neighbors and the AODV algorithm is applied.

A *JobP2P* has the following methods: *sendRequest*, *sendData*, *receive*, *sto-pRelays*, and *run*. The *run* method overrides the parent method of the superclass *Job*, and describes the behavior of the peers. The *sendRequest*ś purpose is to send a request to the relay running on the same CPU unit for a specific destination. The *sendData* method sends data to a relay, which represent a temporary node in the way to the desired gossip partner, if no edge connects them, or otherwise directly to the peer. The *receive* method has two purposes: one is to receive a route requested from the associated relay and the second is to receive data from a partner. After the peers finish their job, they send a message to the task responsible with stopping the relays.

A *JobRelay* has to manage the routing table of each application running on a CPU unit, and also has to keep in the memory the requests that were made by a source node and could not yet been accomplished (pending requests looking for a route to a specific destination). This is necessary as, when an answer is find, it should be replied backwards, unicast to the source. So, for each request that we broadcast to all neighbors, we keep in memory the source and destination, to identify it, and also, the next temporary step to the initial requesting source. The relays also have to monitor the sequence number of each request to a specific target, to avoid loops. So, they maintain a vector with a number of positions equal to the number of vertices of the graph and on position *x* we keep the sequence number of the request to destination *x*. The class has three methods: *receive*, *sendData* and *run*. The behavior of the object is specified by the *run* method and it always should first receive a request and then send some data according to the situation. There are five requirements that are able to be managed and all are listed below:

- Listen for request from the complementary P2P job which asks for a route. If the relay knows a next hop to the destination required, it sends back the reply, otherwise it broadcasts the request to all neighbors. The *JobRelay* forwards the route to the P2P job, when it becomes known.
- Manage requests from other relays looking for a route. The situation is similar to the one above: if the destination can be reached from that point, a reply is send backwards; otherwise a request is broadcast to other temporary nodes directly connected.
- Forward useful data received to a specific destination. If the destination is the CPU unit where this relay runs, the data is sent to the partner *JobP2P*, and the transfer is accomplished.
- Transfer backwards a route reply to a specific requesting source. It is possible that many replies will be received, for the same target, but we know that the first one is the fastest, so we keep it in the routing table and remove it from the pending requests. Therefore, all the other answers corresponding to the same target will be dropped.
- Stop running when a null data is received. This means that all peers finished and sent a message to the *JobCloseRelays* (the job responsible for closing the relays).

The next section provides an example and it is explained how the algorithm works on a specific topology. We will explain how *peer6* can transfer data to *peer1*

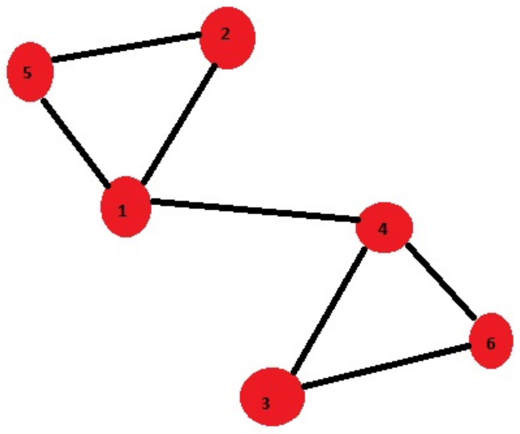

Fig. 8 Graph for the AODV algorithm for overlay construction.

and also, *peer2* to *peer3*. *Peer6* delegates its associated relay with the responsibility to find the next hop in the way to *1*. *Relay 6* checks its routing table and as no entry resides in it for the destination required, the request is forwarded to all neighbors. *Relay 4* has the route to *1*, as an edge directly connects them, so an answer is given back to *relay 6* in order to set the entrance in the routing table and then to *peer6*. After receiving the next hop for the target destination requested, *JobP2P 6* sends the data to the relay part of application *4*, which forwards the message to *peer1*.

Peer2 delegates the complementary relay with finding the route to *3*, the same way described above. Therefore, this one communicates the request to *5* and *1*. None of them has an entry in the routing table for the final remote index, so both broadcast the request to all neighbors except the source of the message. Having this said, *relay 1* forwards the call to relays *5* and *4*. First of them ignores it because it has just processed on its CPU the same request identified by a sequence number associated with the destination. Similarly, *relay 1* drops the requirement solicited by *5*. Before forwarding the message, *relay 1* has to keep in memory some fields for identifying the associated reply containing the route desired, for sending it unicast backwards. So, we will create a pending object with the requesting source and its destination and also, the next hop in the way to the source (source: *peer2*, destination: *peer3*, next hop to *peer 2*: *relay 2*). *Relay 4* knows how to reach the destination (as *3* is connected with *4* and some entrances in the routing table are set at the beginning of the job for all the neighbors), so it sends back to *relay 1* the reply with the route. After receiving it, *node 1* removes the pending object set for this request, fix in the routing table the next hop to the final target solicited, which is *4*, and forward the reply to the next temporary node in the way to the initial source. As this node is right the relay associated to the requesting peer (*node 2*), it sets properly the entrance in his routing table (next hop: *1*, destination: *3*) and sends the reply to its peer. Then *peer2* transfers data to *relay 1*, which forwards it to *relay 4*, and this one to *peer3*.

As in the first algorithm described, two peers connected by an edge in the graph, change information directly, avoiding relays.

This algorithm brings some advantages over the first one, as peers keep in memory only their neighbors. But, in spite of the simplicity brought by the first *JobRelay* designed, the second corresponding job does not only keep memory busy with different information related to the routing process, it also has to manage much more requests. Another drawback of the algorithm implemented is network flooding with messages. Besides that, the number of context switches increases, as for reaching a peer that is not connected to a certain node; we have to make a request to the associated relay. Moreover, when a node broadcasts data, for every message sent, a context switch appears. In the first algorithm the routing decisions were strictly made by the sending peer, so it is avoided any overhead brought by communication between peers and relays corresponding to the same application, and by multiple sending of route request.

7 Simulation experiments

In the simulation experiments, the regional center (P2P) is first described by latitude and longitude, parameters which do not present interest in our simulation as we only have a single regional center with multiple jobs. Then, the name of the local area network is specified, its speed, the name of a CPU and also of the activity object, both described below.

In the CPU section are specified 55 CPU, each with the same features: power, dimension of memory in megabytes, the address in the local area network (first CPU has the address 10.0.0.1, the second 10.0.0.2, etc) and link maximum speed. We use the first unit (0) for hosting the job responsible for closing the relays and the others for the peers and relays specified by the topology.

In the activity section is specified the name of the activity class that generates the jobs. We varied several parameters, such as maximum simultaneous threads, CPU power, memory dimension and maximum link speed, to analyze which of the two algorithms works better in terms of scalability and performance.

For the first series we chose a honeycomb topology with three chains and we calculated each peer's destination according to the formula:

```
ID (Destination) = (ID (Source) + 1) modulo (topology
dimension)
```

A configuration file for a simulation test is presented below:

```
queue\_type = vector
max\_simultaneous\_threads = 1000
regional0 = P2P
network\_fairness = true

[P2P]
latitude  = 45.0
longitude = 27.0
initial\_pool\_size = 0
lan0 = LAN
lan0\_max\_speed = 1000.0 \# Mbps
cpu\_unit0 = cpu0
activity0 = activity

[cpu0]
from = 0
to = 54
cpu\_power = 1.0  \# SI95
memory = 20.0  \# MB
page\_size = -1.0  \# doesn't matter
link\_node = 10.0.0.1        \# the address
link\_node\_max\_speed = 10.0  \# Mbps
```

```
link\_node\_connect = LAN

[activity]
class\_name = P2PAODVActivity
```

Table 3 Results.

				Basic algorithm		AODV algorithm	
max_threads	cpu_power	memory	link_node_max_speed	Execution Time	Simulation time	Execution Time	Simulation time
1000	1.0	20.0	10.0	2404 ms	88.83	36917 ms	247.78
1	1.0	20.0	10.0	2747 ms	88.83	43437 ms	242.74
1	4.0	20.0	10.0	2567 ms	88.83	43114 ms	243.50
1000	4.0	20.0	10.0	2470 ms	88.83	40674 ms	243.37
1000	4.0	10.0	10.0	2844 ms	88.83	43073 ms	242.81
1000	4.0	100.0	10.0	2330 ms	88.83	40225 ms	245.99
1000	4.0	100.0	100.0	4297 ms	9.09	113475 ms	38.01
1000	4.0	100.0	1000.0	3327 ms	2.25	484706 ms	33.78

As seen in Table 3, the simulation time remains constant, when the maximum link speed of a node does not vary. In the first six experiments, we varied the maximum simultaneous threads, the power of the CPU and the memory dimension and the only output parameter that has changed visibly is the execution time, which varied in an expected way (for example, when the maximum simultaneous number of threads decreases, the execution time increases, when the CPU power increases, the execution time decreases, when memory is smaller, the time is greater). We observe an interesting fluctuation of the simulation time when we vary the fourth parameter, as it decreases sharply, with a high percentage on every unit added to the input parameter.

Time variations appear the same way on both algorithms, but we can clearly observe that the real time on the basic algorithm is with at least one unit smaller than on the AODV algorithm, and the simulation time with three times less on the average in the first, than in the second algorithm. We continue to analyze the algorithms and provide a series of tests performed on the same topology (honeycomb with 54 nodes), but this time the destination ID is calculated using the formula:

```
ID (Destination) = (ID (Source) + 8) modulo (topology
dimension)
```

We expect both time parameters analyzed to grow, as in the first series of tests, the way of generation the destination id (*source id* + *1*) caused 15 direct communications (for example *peer 1* with *peer 2* and all the peers of the first chain, beside *peer 6*, which transfers data to *peer 7*, on the second chain, or peers (14, 15), (39, 40), etc. All these pairs transfer data directly, so they finish working - the sending part - immediately after they are scheduled for processing), 28 communications separated by one hop (pairs (12, 13), (27, 28), etc), 10 by two hops (pairs (7, 8), (32, 33), etc) and one by four hops (pair (54, 1)). Thus, reaching the destination desired is much easier and faster this way. In the second series of tests, we have two direct communications ((8, 16), (9, 17)), two communications separated by one hop (10, 18), five by two hops (1, 9), five by three hops (11, 19), two by four hops (50, 4), six by five hops (14, 22), ten by six hops (18, 26), four by seven hops (26, 34), three by eight hops (30, 38), nine by nine hops (33, 41) and six by ten hops (45, 53). We can easily observe that the communication will be slower in this case, as in the basic algorithm, for reaching a destination, a package has to pass through multiple relays compared to the first case, and in the AODV algorithm, in addition to this reason, the reply to a route request will come later, as a node who knows the path to a specific remote index is far from the source.

The conclusions of this set of experiments are similar to the ones above, as before changing the maximum speed of a node link, the simulation time remains at constant values, on both algorithms, and, even though the difference is not significant, it is smaller on the basic algorithm. Moreover, when the link speed increases, the simulation time decreases. We can also state the major difference between the real times measured, in the AODV algorithm being with at least one unit greater.

The next section provides some graphics for a better view of how the jobs evolve, of the time when peers start ending and when the job responsible for closing the relays starts executing the commands, of how a CPU hosting a peer and a relay works with I/O and of how these jobs are completed. The tests are made on the first case of the second set of tests (*max_simultaneous_threads* = *1000*, *cpu_power* = *1.0*, *memory* = *20.0*, *link_node_max_speed* = *10.0* and *destination_id* = *source_id* + *8*).

The first pair of figures (Fig. 9 and Fig. 10) represents the graphics of the jobs simulated. Initially, in the system 109 jobs are generated (54 peers, 54 relays and one job responsible for relays ending). Given the fact that the numbers of pairs which communicate through a certain number of hopes are proportional distributed, on the basic algorithm (Fig 9) we observe an almost linear decrease of the jobs (peers) until

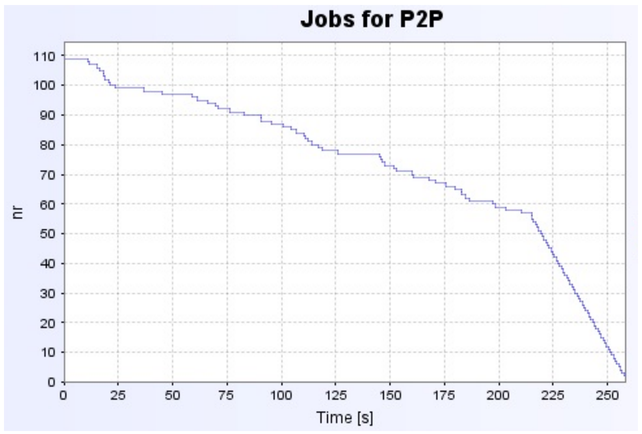

Fig. 9 Graph of running jobs for the basic algorithm for overlay construction, 54 nodes.

a straight line appears, which represents relays closure. All the peers calculate the path to the destination locally, so the ending of a peer depends on how fast its information reach the destination (and this depends on how many hops it has to cross) and when it receive data from a source peer. For example, *peer 16* receives information from *peer 8*, which is directly connected, and sends data to *peer 24* which is five nodes away. Normally, this peer finishes among the first, this fact depending also on the way the jobs are scheduled for processing and on network characteristics. In contradiction, we take, for example, *peer 40* which communicates with *peer 32* (eight hops away) and *peer 48* (nine hops away). This job stops processing later. After the simulated time 220 sec, the relays are closed by the job responsible, each of them being commanded at a fixed time variation (the time between a sending made by the *JobCloseRelays* and a receiving made by the destination relay). This explains the straight line after that time. The difference brought by the AODV algorithm is the fact that jobs start finishing later, as for finding the path to a certain destination, a request should be broadcasted in the system and a reply should be sent back. This happens twice for each *job X*, one search being made for a remote peer desired by *X* and one by a source node seeks the path to *X*. This explains straight horizontal line at the beginning of the simulation and the late start of peers ending.

Figures 11 and 12 are complementary to the graphics above, as the first two show how many jobs are in the system at every second of the simulated time and the second illustrate how many are closed and also, submitted in the system. The straight red line emphasizes the fact that all jobs are generated when simulation starts.

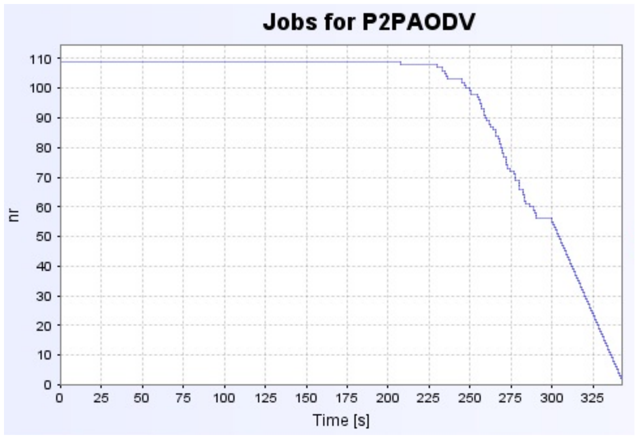

Fig. 10 Graph of running jobs for the AODV algorithm for overlay construction, 54 nodes.

8 Conclusions

The purpose of chapter was to present the general characteristic for P2P simulation, a short analysis of simulation model and existing simulation tools, and to present the design of an algorithm to monitor the resources in a P2P system. Distributed systems have evolved exponentially last decade, so a centralized system to manage them would be difficult and expensive to maintain. Thus, we needed a fully decentralized mechanism.

In this we described an simulation model designed for realistic evaluation of large distributed system, specifically adapted for P2P systems. The model extends the one presented by the MONARC simulator and extensively used in the literature to evaluate LSDS components related to networking, scheduling, data provisioning, security, fault tolerance, and even large architectures. The choice of using the MONARC was also based on the ability to easily extend it with the possibility to model overlays specific for P2P systems, as the simulator already offered a basic layer which contained all the components of the system and their interactions and which was easy extensible according to our specific needs. For a P2P network, constructing the overlay means to respect a given topology of a system when communication is realized. The topology is given by a graph of connections between nodes. We chose to allocate each central processing unit corresponding to a node to the same local area network. Normally, two nodes, which are not connected in the graph, exchange data directly, being in the same LAN. By respecting the topology, they communicate through the path that connects them.

Two algorithms were designed in order to extend this simulation tool. The first one assumes that each peer keeps locally a copy of the entire topology. It uses this topology to calculate the path towards a desired target when sending the package. Accordingly, the message will follow the path computed from the beginning. In the

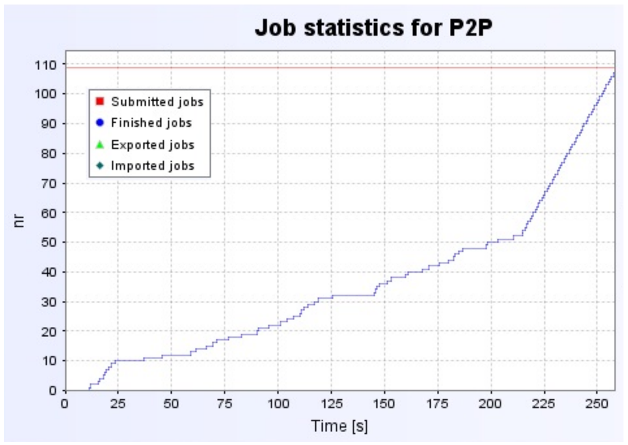

Fig. 11 Job statistics for the basic the basic algorithm for overlay construction, 54 nodes.

second algorithm peers hold only the list of direct connected neighbors. For finding a path to a destination, they broadcast a message in the system, and the peers having knowledge about a path to that destination send backwards corresponding replies.

The simulation model adopted by MONARC follows the design of many actual distributed system technologies. The model incorporates all the necessary components to conduct various simulation studies, ranging from scheduling to data replication or data transfer technologies. In extensive evaluation experiments we demonstrated that the simulation model is perfectly adapted to be used to design realistic simulations of a wide-range of distributed systems technologies, particularly P2P, with respect to their specific components and characteristics.

Acknowledgements The research presented in this paper is supported by national project "DEPSYS - Models and Techniques for ensuring reliability, safety, availability and security of Large Scale Distributes Systems", Project CNCSIS-IDEI ID: 1710. The work has been co-funded by the Sectoral Operational Programme Human Resources Development 2007-2013 of the Romanian Ministry of Labour, Family and Social Protection through the Financial Agreement POSDRU/89/1.5/S/62557.

References

1. Adamic, L.A., Lukose, R.M., Puniyani, A.R. Huberman, B.A.: Search in power-law networks. In: Physical Review E. v046135 i64 (2004).
2. Olteanu, A., Pop, F., Dobre, F., Cristea, F.: An adaptive scheduling approach in distributed systems, In: 2010 IEEE International Conference on Intelligent Computer Communication and Processing (ICCP 2010), Cluj-Napoca, Romania, pp. 435-442 (2010).
3. Eremia, B., Dobre, C., Pop, F., Costan, A., and Cristea, V.: Simulation model and instrument to evaluate replication techniques. In: International Conference on, P2P, Paralel, Grid, Cloud and Internet Computing (3PGCIC 2010), Fukuoka, Japan, pp. 541-547 (2010).

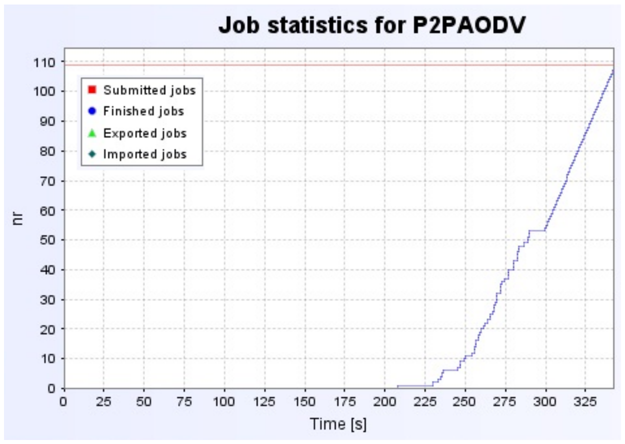

Fig. 12 Job statistics for the basic the AODV algorithm for overlay construction, 54 nodes.

4. Xie, C., Chen, G., Vandenberg, A., Pan, Y: Analysis of hybrid P2P overlay network topology. In: Comput. Commun., 31, 2 pp. 190-200 (2008).
5. Dobre, C.: A General Framework for the Modeling and Simulation of Grid and P2P Systems. In: Handbook of Research on P2P and Grid Systems for Service-Oriented Computing: Models, Methodologies, and Applications, N. Antonoupoulos, G. Exarchakos, M. Li, A. Liotta (Eds.), Ed. Information Science Reference (IGI Global), ISBN: 978-1615206865, pp. 657-686 (February 2010).
6. Dobre, C., Stratan, C.: MONARC 2 - distributed systems simulation. In: 14th International Conference on Control Systems and Computer Science, Ed. Politehnica Press, Bucharest, Romania, pp. 145-149 (2003).
7. Dobre, C., Pop, F., Cristea, V.: A fault-tolerant approach to storing objects in distributed systems. In: International Conference on, P2P, Paralel, Grid, Cloud and Internet Computing (3PGCIC 2010), Fukuoka, Japan, pp: 1-8 (2010).
8. Dobre, F., Pop, V., Cristea, V.: New Trends in Large Scale Distributed Systems Simulation. In: Proc. of the 2009 International Conference on Parallel Processing Workshops (ICPPW '09). IEEE Computer Society, Washington, DC, USA, pp. 182-189 (2009).
9. Dobre, C., Pop, V., Cristea, V.: Simulation Framework for the Evaluation of Dependable Distributed Systems. In: Scalable Computing: Practice and Experience, Scientific International Journal for Parallel and Distributed Computing (SCPE), (ISSN: 1097-2803), Vol. 10, No. 1/2009, pp. 13-23 (2009).
10. Dobre, C., Cristea, V.: A Simulation Model for Large Scale Distributed Systems. In: Proc. of the 4th International Conference on Innovations in Information Technology (Innovations'07), Dubai, United Arab Emirates (November 2007).
11. Stutzbach, D., Rejaie, R., Sen, S.: Characterizing unstructured overlay topologies in modern P2P file-sharing systems. In: IEEE/ACM Trans. Netw., 16 (2), pp. 267-280 (April 2008).
12. Anderson, D.P., Cobb, J., Korpela, E., Lebofsky, M., Werthimer, D.: SETI@home: an experiment in public-resource computing. In: Commun. ACM 45, 11, pp. 56-61 (2002).
13. Milojicic, D.S., Kalogeraki, V., Lukose, R., Nagaraja, K., Pruyne, J., Richard, B., Rollins, S., Xu, Z.: Peer-to-peer computing. In: Technical report (2002).
14. Antoniu, G., Bouge, L., Jan, M., Monnet, S.: Large-Scale Deployment in P2P Experiments Using the JXTA Distributed Framework. In: Euro-Par 2004 Parallel Processing, Lecture Notes in Computer Science, Ed. Springer Berlin / Heidelberg, pp. 1038-1047, vol. 3149 (2004).
15. Pop, F., Dobre, C., Godza, G., Cristea, V.: A Simulation Model for Grid Scheduling Analysis and Optimization. In: Proc. of PARELEC Conference, Bialzstok, Poland, pp. 133-138 (September 2006).
16. Casanova, H., Legrand, A., Quinson, M.: SimGrid: a Generic Framework for Large-Scale Distributed Experimentations. In: Proc. of the 10th IEEE International Conference on Computer Modelling and Simulation (UKSIM/EUROSIM'08), Cambridge, UK, pp. 126 - 131 (2008).
17. Wang, H., Takizawa, H., Kobayashi, H.: A dependable Peer-to-Peer computing platform. In: Future Gener. Comput. Syst., 23 (8), pp. 939-955 (November 2007).
18. Zhang, J., Duan, H., Liu, W., Wu, J: Anonymity analysis of P2P anonymous communication systems. In: Comput. Commun., 34(3) pp. 358-366 (March 2011).
19. Ranganathan, K., Foster, I.: Decoupling Computation and Data Scheduling in Distributed Data-Intensive Applications. In: Int. Symposium of High Performance Distributed Computing, Edinburgh, Scotland (2002).
20. Shudo, K., Tanaka, Y., Sekiguchi, S.: Overlay Weaver: An overlay construction toolkit. In: Comput. Commun., 31(2), pp. 402-412 (February 2008).
21. Walkowiak, K., Przewoniczek, M.: Modeling and optimization of survivable P2P multicasting. In: Comput. Commun., 34(12), pp. 1410-1424 (August 2011).
22. Ramanathan, N., Kohler, E., Estrin, D.: Towards a debugging system for sensor networks. In: Int. J. Netw. Manag., 15(4), pp. 223-234 (July 2005).
23. Rodriguez, P., Tan, S.-M., Gkantsidis, C.: On the feasibility of commercial, legal P2P content distribution. In: SIGCOMM Comput. Commun. Rev., 36 (1), pp. 75-78 (January 2006).

24. Buyya, R., Murshed, M.: GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing. In: Journal of Concurrency and Computation: Practice and Experience (CCPE), Volume 14 (2002).
25. Sole, R.V., Ferrer-Cancho, R., Montoya, J.M., Valverde, S.: Selection, tinkering, and emergence in complex networks. In: Complex. 8, 1, pp. 20-33 (September 2002).
26. Naicken, S., Livingston, B., Basu, A., Rodhetbhai, S., Wakeman, I., Chalmers. D.: The state of peer-to-peer simulators and simulations. In: SIGCOMM Comput. Commun. Rev., 37(2), pp. 95-98 (March 2007).
27. Naqvi, S., Riguidel, M.: Grid Security Services Simulator G3S) - A Simulation Tool for the Design and Analysis of Grid Security Solutions. In: Proc. of the First International Conference on e-Science and Grid Computing, Melbourne, Australia (2005).
28. Lui, S.M., Kwok, S.H.: Interoperability of peer-to-peer file sharing protocols. In: SIGecom Exch., 3(3), pp. 25-33, (June 2002).
29. Dinh, T.T.A., Theodoropoulos, G., Minson, R.: Evaluating Large Scale Distributed Simulation of P2P Networks. In: Proc. of the 2008 12th IEEE/ACM International Symposium on Distributed Simulation and Real-Time Applications (DS-RT '08), IEEE Computer Society, Washington, DC, USA, pp. 51-58 (2008).
30. Cristea, V., Dobre, C., Stratan, C., Pop, F., Costan, A.: Large-scale Distributed Computing and Applications: Models and Trends. Ed. Information Science Publishing, ISBN : 978-1615207039, 390 pg, 2010.
31. Cristea, V., Dobre, C., Pop, F., Stratan, C., Costan, A., Leordeanu, C.: Models and Techniques for Ensuring Reliability, Safety, Availability and Security of Large Scale Distributed Systems. In: 3rd International Workshop on High Performance Grid Middleware, 17th Intern. Conference on Control Systems and Computer Science, Bucharest, Romania, pp. 401-406 (May 2009).
32. Venters, W., et al: Studying the usability of Grids, ethongraphic research of the UK particle physics community: In: UK e-Science All Hands Conference, Nottingham (2007).
33. Joung, Y.-J., Wang, J.-C.: Chord2: A two-layer Chord for reducing maintenance overhead via heterogeneity. In: Comput. Netw., 51(3), pp. 712-731 (February 2007).
34. Takano, Y., Isozaki, N., Shinoda, Y.: Multipath Key Exchange on P2P Networks. In: Proc. of the First International Conference on Availability, Reliability and Security (ARES '06), IEEE Computer Society, Washington, DC, USA, pp. 748-755 (2006).
35. Bagchi, S.: Simulation of grid computing infrastructure: challenges and solutions. In: Proc. of the 37th conference on Winter simulation (WSC '05). Winter Simulation Conference, pp. 1773-1780 (2005).
36. Venugopal, S., Buyya, R., Ramamohanarao. K.: A taxonomy of Data Grids for distributed data sharing, management, and processing. In: ACM Comput. Surv. 38, 1, Article 3 (June 2006).