Peer4Peer: e-Science Community for Network Overlay and Grid Computing Research

Luís Veiga, João Nuno Silva, João Coelho Garcia

INESC ID Lisboa / Technical University of Lisbon

Rua Alves Redol 9, 1000-029 Lisboa, Portugal

Abstract This chapter describes a novel approach to Grid and overlay network research that leverages distributed infrastructures and multi-core machines enabling increased simulation complexity and speed. We present its motivation, background, current shortcomings and the core architectural concepts of the novel research proposed. This is an on-going effort to further our peer-to-peer cycle-sharing platform by providing a scalable, efficient and reliable simulation substrate for the Grid and overlay topologies developed by the research community. Thus, Grid and overlay simulations are improved due to: 1) increased scalability of simulation tools with a novel parallel, distributed and decentralized architecture; 2) harnessing the power of idle CPU cycles spread around the Internet as a desktop Grid (over a peer-to-peer overlay); and 3) a framework for topology definition, dissemination, evaluation and reuse which eases Grid and overlay research. The infrastructure, simulation engine, topology modeling language, management services, and portal comprise a cloud-like platform for overlay research.

Keywords: cycle-sharing, peer-to-peer, overlay network simulation, e-Science, Grid computing, cloud computing, public computing

1. Introduction

The last decade has witnessed the emergence of e-Science in some fields, where the leading scientific research can no longer be carried out resorting exclusively to laboratory equipment. Instead, e-Science research entails the acquisition, storage and intensive processing of vast amounts of data. This requires access to high-performance and/or large-scale computing infrastructures. Examples of e-Science research themes include not only simulations (social, particle physics, systems, and networks), but also drug research and molecular modeling, earth sciences, and bio-informatics.

Researchers working in each specific e-Science field tend to aggregate around informal communities sharing tools, data sets, experiment results, data processing and/or simulation code, and sometimes, computing resources, in a distributed fashion being subjected to different levels of (de-)centralization. However, most e-scientists have limited computing skills, as they are neither computer scientists, engineers or programmers. Hence, the paramount importance of increasing the simplicity and transparency of the tools, middleware, models and algorithms that support e-Science activities.

More pointedly, regarding this work, peer-to-peer overlay networks and Grid infrastructures are areas of very active research within the distributed systems, middleware and network communities. The type of research we are addressing in this work is a part of computer science (developing protocols for overlay networks and Grid middleware is computer science). If this can be carried out with the very help of supporting infrastructures, middleware and applications deployed on a distributed system (reducing coding as much as possible), it is just another case of e-Science (even though it is not addressing fundamental sciences such as physics or chemistry; in fact, e-Science is also carried out on the other side of the sciences' *spectrum*, e.g., with statistical analysis employed in social sciences).

Given the large number of elements in such topologies (overlays and grids), an important fraction of the current research in this field is performed not on real systems but instead resorting to simulation tools (e.g., Simgrid [CAS01], PeerSim [JELA10], OverSim [BAU07]), hence reducing machine cost and administration issues.

Simulation of these topologies amounts to storing all the relevant information of the participating nodes in the simulated topology (either being simulated peers or simulated nodes integrated in a simulated Grid) and executing the protocol-described behavior of all participating nodes. This is very resource intensive as the data and behavior of a large number of nodes is being stored and simulated in a single machine.

Moreover, resorting to simulations enables researchers to carry out experiments with more elements (e.g., participating sites, applications) and more sophisticated behaviour than would be possible by resorting exclusively to real, manually deployed and user-driven applications. The results of these stress-test experiments enable researchers to argue and defend more realistic claims about their proposed overlay protocols, Grid middleware and schedulers. In such competitive research areas, new ideas without strong results to back them up simply will not deserve full credit, regardless of their intrinsic value. As the Internet grows bigger, thus must simulations grow in size and, consequently, also in complexity. Often, simulation code and simulated results are the sole information sources available to perform repeated research, which is still infrequent in computer science but an adamant requirement in other fields, such as medical and biological sciences.

Currently, research in these fields is carried out using two alternatives which are sometimes combined. Protocols and algorithms are sketched and coded resorting to application programming interfaces provided by simulation tools (sometimes, protocol and algorithm code are directly inserted as extensions to the simulation tool source code). This allows the study of the algorithm and protocol properties, coordination, soundness, and behaviour in very large scale populations (usually, around thousands or millions of simulated computing nodes). The simulators (NS2 [IH09], GridSim [BM02], Simgrid [CAS01], PeerSim [JELA10], OverSim [BAU07]) are programs that normally process an event-queue serially, which contains protocol messages sent by nodes, and delivers them to addressed nodes by executing predefined methods on those objects. These tools are limited to the computing power and available memory of a single machine. Therefore, details of such simulations must be restricted to a minimum in order to simulate such vast populations within acceptable timeframes. The simulations do not execute full application and operating system code, and do not monitor communication links with full accuracy.

Therefore, in order to demonstrate the feasibility of the actual deployment of a protocol and algorithm in a realistic test field, researchers must resort to an alternative approach. They make use of distributed test-beds where a limited number (up to a few hundreds) of dedicated physical machines (or shared via virtualization technology) execute the actual complete code stacks (operating system, middleware, protocol, and application code), where the performance of the execution and communication is evaluated by employing real machines and real network links among the test-bed nodes.

Unfortunately, current overlay and Grid simulation is encumbered by architecture, scale and performance limitations that cannot be solved simply by stacking more powerful computers. The performance of simulation tools is hindered because network and topology simulation code is mostly serial and manipulates a large global state, which is assumed to be consistent. Simulations are run in a centralized and sequential way therefore not drawing many of the advantages of increasingly prevalent multi-core machines or computing clusters. Thus, although the increased power of aggregated computers may be used to execute more Grid and overlay simulations simultaneously, it is not possible to run each individual simulation faster or to leverage more CPUs to run more complex simulations (larger number of elements) within a given time frame. Due to these limitations and memory demands, most simulations are today limited to just tens of thousand nodes which is not realistic given today’s existing widespread usage of peer-to-peer systems.

Conversely, although existing utility and cloud computing infrastructures can manage large numbers of (virtual) machines, and can thus execute all the nodes of an experiment concurrently (or in parallel), it is currently unfeasible, both practically and financially, to allocate millions of machines in a dedicated test-bed. An alternative source of computing power must be found and drawn from. Large scale peer-to-peer infrastructures, usually dedicated to content-sharing, may be leveraged as processor-sharing overlays (usually called cycle-sharing platforms).

In this document we described the motivation, vision, architecture, and current implementation and performance results of Peer4Peer, an ongoing project addressing the challenges described above. Peer4Peer prescribes the usage of a world-scale free and voluntary cycle-sharing peer-to-peer infrastructure to perform simulations of actual peer-to-peer protocols, algorithms and Grid middleware, of virtually unbounded size and with increased performance, breaking the serial nature of current simulation tools. Furthermore, the peer-to-peer infrastructure can also be leveraged for content-sharing and store programs, scripts, configuration of simulated network topologies, experimental results, and schedules for experiment/simulation execution. This will facilitate the dissemination and reuse of research in areas of peer-to-peer protocols and Grid middleware. In summary, Peer4Peer is aiming at providing researchers with a platform to support e-Science, in the specific field of overlay and Grid simulation.

The remainder of this chapter is organized as follows. In the next section, we present a comprehensive portrait of how e-Science is carried out nowadays. In section 2, we present the case for a novel approach to overlay and Grid simulation. Sections 3 and 4 presents the main aspects of the broader long-term vision associated with the Peer4Peer project. In sections 5, we present the architecture of the current development efforts to deploy Peer4Peer, while section 6 describes the main components and implementation details of the current deliverables. In section 7, we offer evaluation and performance measurements of the implementation already carried out. Section 8 is dedicated to put into perspective the relevant related research in the main themes connected with our work, in a broader view. The document closes with some conclusions and future work.

2. Current Approaches to E-Science

The 1990's saw the birth of the cluster, a group of co-located general purpose computers connected by a dedicated network, as a mean to execute parallel scientific computations. PVM [S90] provided a set of message passing primitives, with implementations available on multiprocessors and LAN connected workstations, while, from the distributed systems side, NOW (Network of Workstations) [ACP+95] provided an abstraction layer, allowing the transparent access to network scattered resources (storage and CPU). From that point on, the architecture of a typical parallel system has been moving away from massive multiprocessors (connected with a proprietary bus) to clusters of computer nodes (some of them multiprocessors themselves) connected with commodity network equipment (Gigabit Ethernet or Mirinet).

This evolution of execution environments affected the available programming systems as well as resource management and scheduling middleware. Although clusters allow avoiding the cost of acquiring a mainframe or supercomputer, they have some drawbacks: competition between local (usually interactive) applications and cluster-wide applications, configuration and performance difficulties when the cluster is heterogeneous, and, lastly, reduced bandwidth if the cluster network interconnect is used for multiple purposes.

At the same time, the development of the internet allowed an easier access to remote resources. One of those resources was computing cycles. SETI@home [ACK+02] was one of the first systems to successfully use remote and scattered idle computer to process data and solve a scientific problem. Following these trends, today most parallel computations either run on clusters of computer nodes [TOP500] or over the internet [CKB+07, SCH07], and most fit in one of two categories: message passing or parameter sweep.

In message passing applications, data is split and distributed among several nodes. Periodically messages are transmitted between nodes to transmit the relevant information, mostly changes on frontier data. To perform such communication, the most used API is MPI [F95]. In parameter sweep application, a master process goes over a range of possible parameters by starting several concurrent slave processes which, while running the same program, are executed using different parameters. Once these slave processes conclude, results are transmitted to the master process and aggregated there.

In clusters of small dimension, the management of the nodes can be had-hoc and without the need of specialized software. For any medium sized installation with several concurrent users it is necessary to have a software layer that manages the available resources, the submitted jobs and the authorized users. Resorting to nodes scattered over the internet to create a sort of ad-hoc cluster in order to perform a computation, amplifies both the benefits and problems of traditional clusters: nodes are even more varied and unreliable, and networks have less bandwidth and are less secure.

Examples of cluster resource managers are Condor [LLM88], Portable Batch System [HEN95], or Sun Grid Engine [GEN01]. Besides this software layer, clusters still need a distributed file system (such as AFS[HOW88], CIFS [LEA96], GPFS [SH02] or GlusterFS [GLU10]) in order to guarantee a uniform file access on every node. These resource managers handle the submitted jobs queues and schedule those jobs to the available resources. This process takes into account the requirements (memory, and number of processors), the available resources and other users privileges, and uses different scheduling policies [OSU82, FR92, KA99, JN99].

With Ian Foster's vision [FK99, FOS02] of a new computing infrastructure for research, the so-called Grid, the classical cluster disappeared from the center of the environment, becoming just a service for the creation of globally disperse communities. This vision encompasses two distinct development directions: the infrastructure to connect the various scattered resources (clusters) and the tools to ease the access to data, processing resources, and results.

One of the first initiatives for building a Grid infrastructure was Globus [F06]. Besides the creation of an infrastructure to connect geographical disperse resources, Globus also helped define a series of standards related to authentication, resource accounting, and interfaces to access remote resources. This new vision of a global computing environment changed the way authentication should be performed. While on a local cluster system a simple user/password authentication scheme was simple enough, on a globally distributed environment it is not. As users from different organizations (with some local common interests but possibly with global conflicting ones) will interact and share resources, sophisticated authentication schemes and resource access policies are fundamental.

Classical parallel programming paradigms can also be deployed on the Grid: users either develop a parallel application using a message passing API or define a parameter sweep job. In addition to these models, grids frequently support the definition and execution of workflow patterns, where a set of resources are accessed and used according to a graph description where each node represents a job and each vertex represents a dependency and/or data transmission between jobs.

In order to allow the parallel execution of programs on grids, meta-schedulers (Globus GRAM [FFM07], GridWay [DSA10]) have been developed. This new software layer aggregates different clusters and provides a uniform view of the various resources available. As in a cluster, when submitting a job, the user defines how many processors are required and submits the execution code, the input files and parameters. Based on this information, the meta-scheduler finds the best remote cluster to execute it.

Another vector of Grid research and development was centered on the creation of communities around research subjects. The fundamental aggregation software for such communities are research portals. Generic Grid portals (GridSphere [NRW04], NINF [SNS+02], Nimrod/G [AGK00]) allow for the interaction with the Grid using a simple user interface. With these portals users can authenticate themselves on the Grid, and define parallel jobs without resorting to the command line or editing configuration files. Although Ganga [HLT+03] is a generic Grid framework for job definition and management that allows for the development of specific plug-in to be used by specific communities. At the moment there are plug-ins to execute simulations in the context of CERN's LHCb and ATLAS experiments.

Other Grid portals are specific to a research community. Through a web based graphical interface users may access experimental and simulation data, reuse previously developed simulation code and start jobs on computing clusters. NanoHub [KMB+08] is an example of such a portal. Besides the access to community web resources it also allows for the execution of simulation codes using available Grid backends.

Another relevant Grid platform is LEAD [PLA+06], a project in which a service-oriented architecture (based on web services) is used to leverage and extend a weather simulation Grid. The high level of service composition in LEAD provides users with powerful tools for simulation data generation (Weather Research and Forecast model), storage (myLEAD), search (Algorithm Development and Mining, aDaM) and visualization (Integrated Data Viewer). This architecture not only provides resource discovery to leverage computing power (mostly from supercomputing centers, albeit these are not usually available in research developed in other fields), but also eases application development (resorting to a workflow system, analogous to DAGMan in Condor), and data management and search (by means of a metadata catalogue).

Triana [TWS+05] is a Grid middleware framework that also aims at easing e-Science by, besides offering resource discovery and scheduling, providing an alternative application development model for Grid applications, coding-free for most researchers. This model is based on the graphical interactive composition of reusable components from an extendable component library (more than 500 developed covering, e.g., statistical analysis, signal, audio and video processing). Components are inserted into task-graphs akin to workflows and can be scheduled for execution in distributed manner. Although initially a stand-alone approach exclusively based on Java and RMI, Triana is now interoperable with Grid middleware, component frameworks and web services. Thus, Triana becomes more of an application development architecture and model than strictly a Grid middleware.

While the fully institutional parallel computing has been growing with several Grid initiatives, a more lax approach to parallel computing has also grown. With the growth of the Internet, idle computing cycles on desktop computers became a resource to be used and aggregated. One of the first projects to successfully use this new resource was SETI@home [ACK+02]. Attaining a performance of about 27 TeraFLOPS it proved that embarrassingly parallel applications could be executed outside the classical computing centers. Several projects tried to develop efficient architectures for the execution of such problems on the Internet, but the only successful surviving system is BOINC [AF06, SCH07]. Although its architecture is simple, it is difficult to set up a BOINC server and gather donors, and this architecture is limited to the execution of parameter sweep simulations, not allowing the execution of regular message passing parallel applications.

The cloud, in the form of utility computing infrastructures [AMA10,OPL10, EI10], is considered the next step in large-scale computing infrastructures allowing for the integration of grids (with computing clusters distributed across virtual organizations) and utility computing farms (Amazon Elastic Clouds [AMA10]). Such infrastructures revolve around the instantiation of system-level virtual machines (e.g. Xen), of preset configurations, charged by the whole number of processing hours or calls to web applications. When applied to the parallel execution of simulations this new infrastructures can be a viable and affordable source of computing power [EH08].

Although easily accessible, the resources provided by a computing cloud can only be used as a classical cluster, either to execute message passing applications, web applications or embarrassingly parallel problems.

The Archer platform [FBF+08] goes one step further from the classical Grid portal systems. It gathers around the same platform the users with similar computational requirements (simulation of computer architectures), but in the back end its approach diverges from the traditional Grid. Besides the usual Grid resources, Archer aggregates donated resources from the edge of the internet (as any other distributed computing infrastructure) and uses cloud and utility computing infrastructures as a source of computing power.

We have seen that the actual source for computing cycles ranges from dedicated clusters to the edge of the Internet at users' homes. Utility computing infrastructures are also becoming a viable source of computing power to solve simulation problems. Existing software has also been adapting to the new requirements: it is possible to execute parallel MPI based applications on clusters (dedicated or created on utility computing infrastructures), execute embarrassingly parallel application on the Internet (using distributed computing infrastructures) or execute workflows on the Grid (using for each step the best available resource).

To help use scattered and distributed resources, user communities have been gathering around Grid portals. These systems not only enable the sharing of knowledge (scientific results, educational material) but also allow an efficient access to distributed computing infrastructures. These portals only provide access to the Grid: set of institutional infrastructures where resources are lent or rented for a particular objective.

3. The Need for Next Generation Overlay and Grid Simulation

Clearly, the rich plethora of existing approaches to distributed computing and distributed resource and content sharing, described in the previous section, have enabled e-Science in various domains. Armed with such vast amounts of computing, storage and communication resources, researchers in various fields (e.g., physics, chemistry, materials science or biology) have achieved discoveries that could otherwise simply not have been possible in our lifetime due to limited computing power and resources.

Nevertheless, in several specific domains, simply amassing distributed resources is not enough. Simulation of distributed systems, namely peer-to-peer overlays and Grid middleware, is such a domain. Although the extra available resources can offer improved overall throughput (i.e., ability to perform more simulations simultaneously), by themselves they cannot guarantee improved speed (i.e., the elapsed time of each individual simulation).

In this chapter, we are addressing one of these specific domains where e-Science has offered only limited support for researcher interaction, content sharing, and some resource sharing but none for increasing the speed of experimentation. This is especially relevant in an era where multicore and multi-CPU machines are becoming prevalent, and cloud computing is promising elastic allocation of cheaper computing resources. These aspects should be entirely leveraged and taken advantage of in order to improve the performance of the tasks comprising peer-to-peer and Grid simulation.

Currently, many researchers neither have access to clusters (nor grids), nor have an adequate budget for accessing commercial cloud computing platforms such as Amazon EC2. Furthermore, they face a conundrum when testing algorithms and middleware: they either use higher-level Grid/overlay simulators managing thousands of nodes but run serial, and not distributed or parallel simulations; or they use parallel and distributed lower-level system test-beds but run complex OS/application code requiring computing power several orders of magnitude higher for experiments of comparable complexity.

Existing free large-scale distributed research test-beds (e.g., PlanetLab [CCR+03], Emulab [WLS+02]) allow running experiments of complete distributed systems executing on real test-bed nodes or virtual machines, using complete system images (operating system, protocol/middleware and application code) of each of the simulated participating sites in a distributed computation. However, the inherent overhead is prohibitive and simulating a small to medium size system (e.g., hundreds of nodes) would require roughly the same order of magnitude of test-bed nodes to be available simultaneously which is impracticable. Some interfaces for network (NS2) and overlay (OpenDHT) simulators are also offered but they only reduce the burden of having to ship entire virtual appliances to be executed at the test-bed nodes. They do not allow cooperation among several simulators to parallelize simulations or simulate larger, more complex topologies.

When simulations are designed, the models and representations used to describe topologies are often of low semantic level, ad-hoc and mostly programmatic. This forces researchers to actually write tedious code to instantiate nodes in the topology and intricate, tool-dependent code to implement basic data structures and behavior (routing, neighbor tables). Topology reuse, analysis and both scholarly and teaching dissemination are greatly limited.

In summary, there are three important challenges that correspond to present shortcomings causing drag in these areas, regarding not only research but also even teaching:

- the serial nature of simulation tools, commonly involving centralized globally shared data and event queue manipulations, prevents the scale-up that could be provided by the increasing democratization of access to multi-core machines (e.g., dual-core laptops and quad-core desktops);

- the impossibility of executing simulations in distributed manner thereby preventing the scale-out that could be reached by engaging platforms for voluntary cycle-sharing, utility computing and Grid infrastructures (in testbeds as PlanetLab full execution imposes an overhead that greatly limits simulation size); and

- the dominance of programmatic descriptions of researched and tested topologies, locked in specific programming languages and simulator API idioms.

The aforementioned shortcomings cause a number of problems that hinder the expansion of current overlay and Grid research, both in depth, breadth and community impact. Examples of such restrictions include: limited simulation size and complexity; limited scope of results; inefficient employment of resources involved in experiments; lack of architecture/simulator-agnostic descriptions of protocols, middleware, and schedulers; very restricted ability to perform independent repeated research [NLB+07]; lack of a uniform repository for reusability of such research; global absence of accessible teaching platforms.

In the next sections we will describe in greater detail the ongoing work to achieve this with: i) a global overview of the Peer4Peer proposal, ii) the architecture for its deployment, iii) details of the current implementation, and iv) some performance and scalability results.

4. The Peer4Peer Vision: Peer-to-Peer Cycle-sharing, Parallelized Simulation, Data and Protocol Language

Peer4Peer's aim is to provide researchers with a cloud-like platform for network overlay research. This entails contributing to laying foundations that will contribute to step up research in peer-to-peer overlay and Grid topologies to a new level. In order to achieve this, we need to tackle existing limitations regarding scale, efficiency and expressiveness by providing a new platform for overlay simulation research, as described in this section.

Therefore, in order to advance further, overlay and Grid simulation face new requirements:

- increasing scalability of simulation tools with a novel parallel, distributed and decentralized architecture;
- harnessing the power of idle CPU cycles over the Internet by incorporating them in a desktop Grid or a public computing infrastructure (on a peer-to-peer overlay), avoiding the need for proprietary clusters, access to institutional grids or utility computing budgets; and
- further Grid and overlay research with a framework for integrated topology definition, dissemination, evaluation and reuse.

To address the problems identified, the Peer4Peer approach is to attack them both in breadth and depth. This implies contributing with novel scientific solutions to specific problems (e.g., resource discovery on peer-to-peer systems), as well as providing the research community with a useful and usable simulation infrastructure integrated with a portal for easy access, configuration and sharing.

The primary substrate for Peer4Peer is an extendable peer-to-peer overlay platform (the mesh) comprising possibly asymmetrical participant nodes, aggregating individual desktop peers as well as efficiently leveraging, if available, computing time on server clusters, Grid sites and utility computing facilities.

Simulations of peer-to-peer overlay and Grid topologies are to be parallelized and distributed. What is now mostly a single main task in current simulation tools must be decoupled in order to allow concurrent progress of (partial) simulations of different regions of the simulated topologies. This results in higher speed-ups due to parallel execution and increases scalability by enabling the mesh to host simulations of potentially unbounded size and interaction complexity.

To overcome the lack of expressiveness of the simulation specification mechanisms in current tools, there is a need for a domain-specific language that allies greater expressiveness with the ability of reuse for teaching, study, and repeated research. Such topology modeling language (TML) will allow the specification of simulation requirements, flat, layered, multidimensional, hierarchical and novel recursive topologies to simulate arbitrarily large systems. A TML must encompass, at least, rules to specify: entry, exit, routing and recovery protocols; neighbor tables; indexing and data stored at nodes and how/where to retrieve it.

Finally, available open-source groupware and content management systems may be extended in order to create a portal for groups and communities to interface with Peer4Peer and interactively share topologies, simulations and results. An XML-RPC or REST-based interface enables automatic deployment and retrieval of results promoting embedding of Peer4Peer in web pages and mash-ups.

Simulated topologies need not be limited to content sharing. They could themselves be simulated cycle-sharing topologies as it is common in Grid infrastructures. Therefore, the use of a TML may allow the definition of policies regarding resource discovery, reservation, scheduling, accounting, recycling, redundancy, versioning, etc. to be enforced within each simulation.

Therefore, at its core, Peer4Peer is a recursive overlay topology comprising its substrate and all the simulated topologies. TML and its templates will be the empowering means to allow simulated topologies to define and explore new structures, behaviours, strategies and their easy widespread sharing.

5. Current Peer4Peer Architecture

In this section, we describe the main aspects of the current Peer4Peer architecture, which is depicted in Figure 1: the mesh of simulator nodes running the overlay simulator, the simulation calculating the evolution of the state of a simulated network overlay, the services supporting the establishment and management of the simulation network, the topology model that formally specifies the simulated overlay, and, finally, the portal providing access to the simulator and mechanisms for sharing all involved information (topologies, simulation scenarios, results, etc,...)

A. Mesh: Cycle-sharing overlay

The mesh is a peer-to-peer overlay able to harness idle computing cycles from asymmetrical participant nodes. Thus, it incorporates specific mechanisms and interfaces to engage peers, both individual desktop, server clusters, Grid sites and utility computing time slices. Each type of peer has different capabilities and a dedicated module is able to access them (invoking corresponding middleware) and exploit them (representing them as higher capacity nodes) by giving them higher simulation loads.

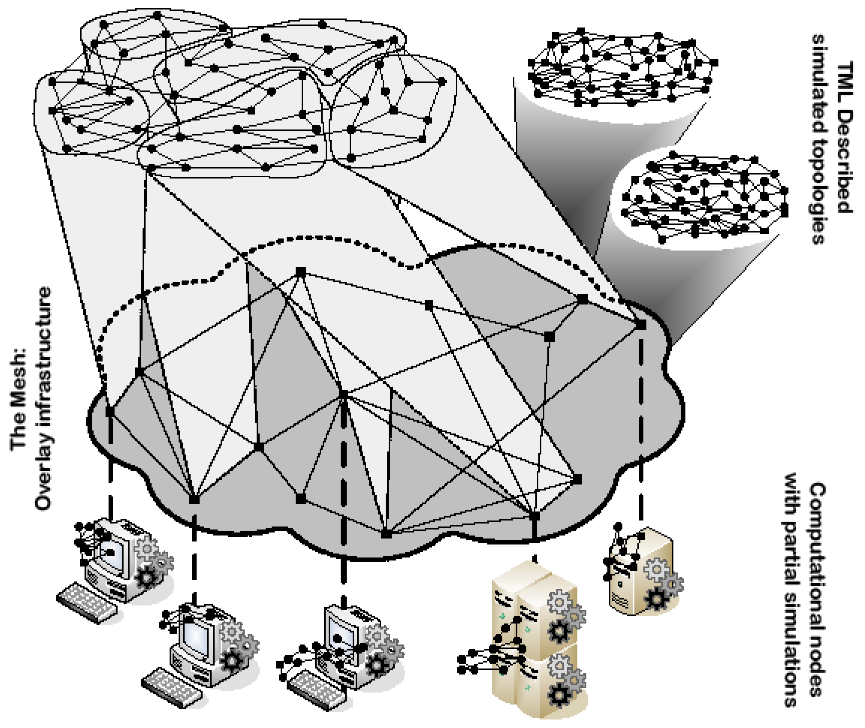

Figure 1: Peer4Peer overall vision

The Peer4Peer mesh is extendable. It provides the fundamental support for simulation deployment, result caching, data storage, and resource discovery and management for cycle-sharing and distributed scheduling. On top of these fundamental mechanisms, more high-level services can be deployed such as retry-based fault-tolerance, replication for reliability, checkpointing and migration to ensure progress in long simulations, or more global evaluation aspects such resource recycling, accounting and quality-of-service. Though challenging, it must be noted that such high-level services can be implemented at the overlay/simulator-level. Therefore, the associated software development effort is much lower than implementing similar mechanisms within an operating system or virtual machine runtime.

The mesh follows a hybrid and hierarchical structure. In this sense, the mesh behaves as a recursive overlay topology. Regarding content and routing, the mesh is a structured overlay. Each top-level data item stored in the mesh is in itself either a topology description, a description of resource discovery and management policies, simulation setting, or results of simulation execution. Topology descriptions encompass node population, routing and communication protocol, and content placement rules. Resource and management policies to be enforced include: description of resource discovery, reservation, scheduling, accounting, recycling, redundancy, and versioning.

Simulation settings encapsulate topology, resource and management policies. Simulation results include the outcome of simulations and aggregate properties such as average and total messages sent, etc. Simulated topologies are represented and managed across Peer4Peer as recursive topologies, i.e., overlays where data items represent segments of the very Grid and overlay topologies whose simulation may span tenths or hundreds of computers.

Regarding base-level services such as resource discovery (mainly w.r.t. CPU, memory and bandwidth) and scheduling of (partial) simulations, the mesh will employ automatically designated super-peers to act as information hubs about resources of groups comprising their neighboring nodes. Super-peers will aggregate into a structured overlay only to exchange resources when there are not enough within each group.

B. Simulator: Parallel and Distributed Topology Simulation

At every node in the mesh, a number of concurrent simulator threads is executed. Each one is in charge of executing a number of simulation steps to simulate a specific region of the simulated topology, i.e., it will perform a partial simulation. This way, events localized in the vicinity of a simulated node cannot interfere with other simulated nodes at a distance preventing them from being influenced by it (i.e., out of the event – message – horizon for a given number of hops) until some time has passed. Partial simulations may run in the same or different nodes.

Each node of the mesh will execute a component of the distributed simulation engine. Communication among nodes running partial simulations of one topology simulation interact using a standard API that either manages shared memory or exchange messages via the mesh. The simulation engine component may have different versions in accordance to the type of underlying infrastructure providing computing cycles. All versions of the simulation component perform parallel (i.e., multi-threaded) execution targeting multi-core machines as even desktop computers can benefit from it. All versions can be deployable as virtual appliances.

Simulation components are to be executed primarily on common cycle-sharing desktop nodes. Nonetheless, executing simulations on server clusters, Grid sites and utility computing infrastructures will leverage widespread middleware, libraries and virtual machine support (e.g., shared-memory, STM, Terracotta, MPI, Globus, and Xen).

Despite the simulations being distributed, consistency among simulation nodes must ultimately be enforced to ensure simulation accuracy and correctness, although a high degree of non-determinism is already present in this type of systems in real deployments. In order to achieve this, we partition the global simulation state and investigate the adaptation of optimistic consistency algorithms, such as divergence bounding and more recent vector-field consistency [SVF07] employed. This leverages notions of locality-awareness employed in most massive multiplayer games that, while guaranteeing global consistency, employ region-based approaches to speed up most of the game interaction. The amount of time, or number of rounds, that mandate synchronization among regions, to ensure safety when interactions cross region boundaries, can be configured and tuned. Hard synchronization needs only to occur when events interfere with two or more of such simulated regions.

C. High-level Services

Since the mesh is mostly comprised of voluntary desktop machines, node churn (entry, exit or failure) will be high in most peer-to-peer overlays. Therefore, to ensure fairness and balance in resources used by running simulations and to prevent wasted efforts on node failure, a number of higher-level services need to be deployed on top of the basic mesh. These will be included in the basic client as plug-ins to the mesh protocol and simulator engine.

To ensure fairness, these modules will provide migration of partial simulations to perform load-balancing. Migration is also useful to attempt at co-locating sibling partial simulations in the same mesh node or neighboring nodes. This will prevent network communication among simulation engines and increase speed.

To prevent wasted work, the simulation modules provide fundamental fault-tolerance and reliability support. Each partial simulation may be scheduled several times to different nodes in the mesh, its results gathered and compared in order to confirm their correctness. To ensure that very long running simulations make any progress, partial simulations will be checkpointed periodically and their state stored in the mesh as regular simulation result data.

Therefore, over time, execution of simulations over the mesh, together with the expectable fluctuations in the mesh membership, will trigger higher-level services such as migration in order to restore load-balancing, and to improve performance by aggregating simulated regions with more intensive intercommunication within neighboring mesh nodes, or even within same node if sufficient capability is available there.

As a bootstrapping configuration mechanism, to maintain proper balance with respect to scheduling across the mesh, in spite of the different capabilities of the nodes, we make use special template simulations in order to benchmark average simulation performance on each platform.

D. Topology Modeling

A key aspect to performing network simulation is being able to formally describe of the topology of the simulated network or overlay.
In order to allow faster and easier topology prototyping, development, deployment, testing, evaluation and redistribution, we are designing a new domain-specific language (a topology modeling language - TML) to express everything in Peer4Peer: data, structure, behavior and policies. TML prescribes a simple mostly declarative syntax, inspired in Java, allowing the

definition of data structures, events and rules (predicates) to trigger actions (e.g., sending messages).

TML syntax includes features for the overall simulation description and topology structure (e.g., flat, layered, multidimensional, hierarchical and possible novel recursive topologies). This way, with reuse, TML allows easy definition of arbitrarily large systems in comparison with today's cumbersome programmatic approach. Regarding data structures inside simulated nodes, TML syntax encompasses rules to specify neighbor and routing tables. Concerning behavior, TML is used to express protocols (e.g., entry, exit, routing, and recovery). TML also allows the definition of pre-existing data content placed at simulated nodes prior to simulation start.

TML consists actually as a front-end pre-processor that generates source code targeting simulators' APIs (currently, only PeerSim). Nonetheless, if allows independence of simulator API and details, and can be made to target other simulators. We also intend to explore approaches based on byte-codes to speed up simulation execution. Such an approach will ensure compatibility across platforms hosting the mesh and the topology simulator.

At a higher level, TML aims the enrichment of simulations by separating topology structure and routing from higher-level policies such as resource discovery, reservation, scheduling, accounting, recycling, redundancy, versioning. Once defined, the description of a policy should be reusable in other simulations. This way, definitions of topology-related structure, data, behavior and policy can be mixed-and-matched resulting in increasing reusability, productivity and observability of studied properties.

E. Integration and Portal Support

To ease the deployment of Peer4Peer, its clients are distributed by using a number of alternatives all of them incorporating the same message and simulation protocol to ensure portability and integration. However clients can be either stand-alone applications to be run on desktops, jobs on a Grid (for maximum performance), or several space-efficient virtual appliances; thereby easing deployment and taking advantage of concurrent execution of partial simulations that can take place in multi-core and cluster participating nodes.

A web portal supports content sharing of topologies, simulations, policies and/or results among users, groups, and simulation-related communities. The main part of the development effort focus on allowing web interactive and automatic (scripted) interface with Peer4Peer. This is achieved by resorting to XML-RPC or REST-based interfaces. This also has the interesting outcome of allowing embedding of Peer4Peer in web pages and mash-ups, and automatic deployment and retrieval of results. This fosters the creation of a truly topology simulation community portal providing integrated and universal access to content and resources to perform simulations.

Finally, the portal, instead of a standalone infrastructure, can be integrated with available open-source groupware and content management systems to manage groups, communities and social networks to interface with Peer4Peer and interactively share topologies, simulations, results.

6. Implementation

The foundation of the current Peer4Peer's implementation is a platform aimed at the execution, with improved scalability and performance, of the peer-to-peer simulator, PeerSim, on top of a cycle-sharing infrastructure. This infrastructure is able to leverage the resources available at nodes included both in local clusters or LANs, Grid infrastructures, and a custom peer-to-peer network of voluntary cycle-sharing nodes. PeerSim execution is parallelized and being made distributed.

The Peer4Peer platform implementation is depicted in Figure 2 with its main software components and data flows. Peer4Peer implementation is based on the following components, following a top-down approach:

- job management and portal services using nuBOINC [SVF08], to store descriptions of job requests and primary replicas of topology descriptions, simulated protocols, experiment results, etc.;

- resource requirement description and resource discovery coordination using STARC [SVF10];

- peer-to-peer infrastructure enabled with cycle-sharing capabilities, called Ginger [VRF07], with overlay membership management, decentralized resource discovery, distribution of computation and decentralized data caching;

- modifed version of PeerSim (P4PSim) which runs protocol simulations in parallel, to take advantage of multicore hosts for increased performance, and with support for distribution to allow for larger and more complex simulations.

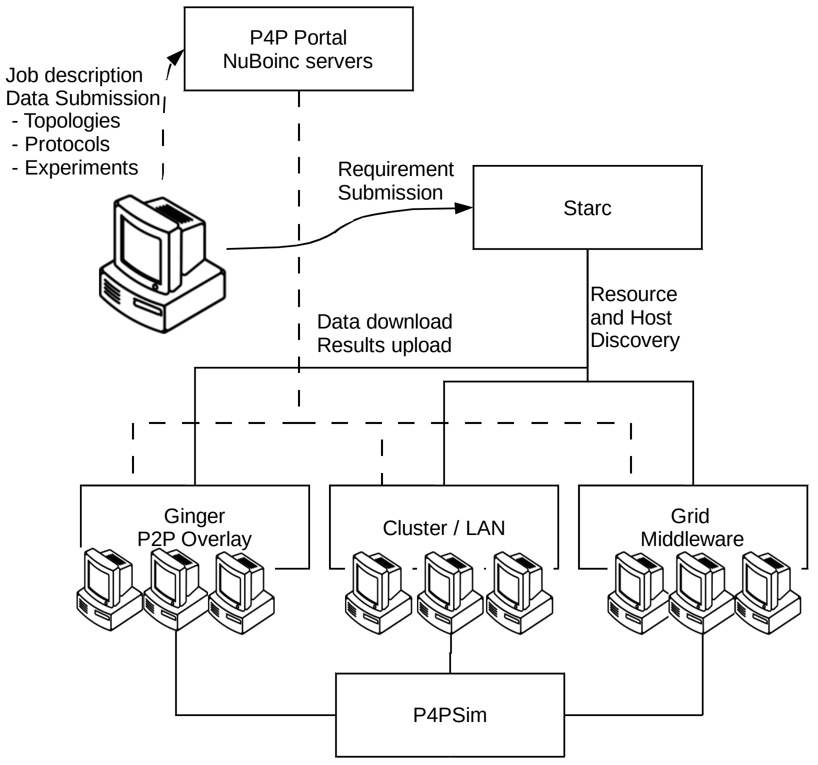

Figure 2: Peer4Peer main software components and data flows

Broadly, at the top, users submit job descriptions and data (topologies, protocols, experiment results, etc.) they want to make globally accessible to the Peer4Peer portal hosted by a number of nuBOINC servers. Job descriptions input data and simulation results may be cached elsewhere (namely, entirely on clusters or in chunks across a P2P overlay). At the bottom, every participating and donor node runs Ginger embedded with P4PSim to perform actual simulations.

Users may submit requirements (e.g., CPU cores and speed, memory, disk, network bandwidth) to STARC that will interface with three types of existing underlying network architectures: LAN and cluster managers, Grid middleware, and a P2P overlay (enhanced with cycle-sharing). This enables STARC to discover hosts able and willing to execute job tasks. Tasks are converted in gridlets (a work unit) that can be appended to resource requirement descriptions fed to STARC; they can be created either at the nuBOINC servers or previously by submitting nodes (by the Gridlet Manager in Ginger). The discovered hosts can retrieve job descriptions (optionally also embedded in gridlets) and data from nuBOINC servers or from the Ginger P2P overlay. When nodes do not belong to the Ginger overlay (relevant case but not the most interesting in the context of this work), the cluster coordinator or Grid middleware spawns a process with a Peer4Peer virtual appliance with: P4PSim, Ginger middleware (for gridlet management and optional access to Ginger P2P overlay) and a nuBOINC client to access the portals.

A. nuBOINC - Peer4Peer Portal

nuBOINC [SVF08], Peer4Peer's job management module, is an extended version of the Berkeley BOINC infrastructure. The main improvements provided by nuBOINC are the ability to submit jobs for legacy commodity applications or execution environments, PeerSim in our particular case, and the possibility to express a large set of jobs by a set of different input files or parameters for the same application, i.e. different peer-to-peer simulations in PeerSim. Every participating (and donor) node runs a client-version of nuBOINC for information submission and retrieval from nuBOINC servers at Peer4Peer portal.

Regarding data submitted by users, topologies and protocols employ an early approach to TML (topology modelling language). Node topologies are represented as pairs of IDs of simulated nodes (representing links to neighbors, that can be reciprocal or not). Data structures can be represented by any self-contained Java class (i.e., containing as fields only primitive types and arrays of primitive types). Behavior is represented as individual methods that manipulate a set of macro variables, replaced by the pre-processor. Method execution is triggered by condition rules (encoded as Java expressions). Although still in progress, this approach already offers relative independence of simulator API and details, and can be made to target any simulator (although the recommended simulator would be P4PSim for performance).

B. STARC – Resource requirement descriptions

The discovery of processing nodes for use in nuBOINC is based on STARC (see Figure 3). STARC [SVF10] is a middleware platform to make resource discovery more adaptive via extensibility (ability to incorporate new resources) and increased flexibility with better expressiveness in requirement description by employing XML files with a resource description algebra.

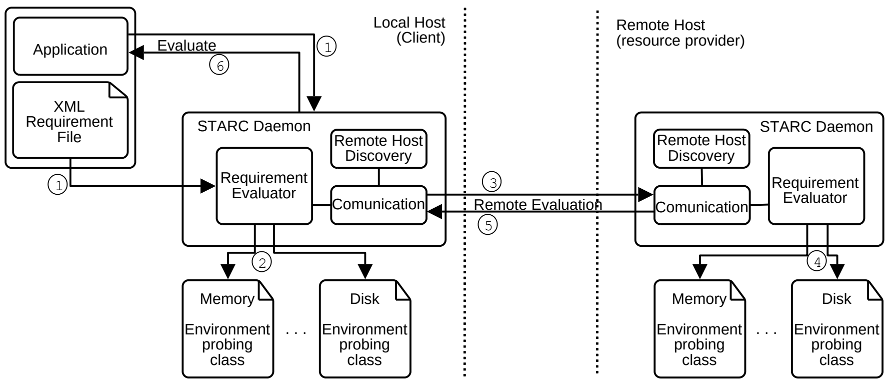

Figure 3: STARC Architecture

The STARC middleware is an assemblage of components, named STARC daemons that execute both in clients and in resource providers or donors. STARC uses XML requirement files stating application requirements. XML files state individual requirements (e.g., CPU), and allow for logical expressions combining requirements and their priority. Moreover, they described the assigned utility of resources and their depreciation partial fulfillment, while employing fuzzy-logic to combine multiple requirements.

XML requirement files are fed to a locally running STARC daemon. This daemon is responsible for the discovery of remote hosts and the evaluation of the requirements on those computers. The requester will receive a list identifying the available hosts and indicating how capable they are of fulfilling the requirements. In general, these requirements can range from available memory or processor speed to certain libraries of helping applications.

In more detail, the local STARC daemon receives a XML requirement file that is interactively created when the user submits jobs to nuBOINC servers. From this file (Step 1 in Figure 3) the STARC daemon reads the requirement and executes the relevant *Environment Probing Classes* (Step 2) in order to know how the resources fulfill the requirements. The requirements are evaluated using the values returned by the *Environment Probing Classes*. The evaluation of each resource is performed by a specific *Environment Probing Class*. The name of these classes is defined in the XML elements. This enables dynamic class loading. After local resource evaluation, if the request was originated from the local user, the STARC daemon contacts the remote daemons (Steps 3) by means of the *Communication* component. Each contacted Daemon evaluates the requirements (Step 4) and returns the resulting value (Step 5). The *Remote Host Discovery* module assembles and sorts the returned hosts, storing them in a list.

In cluster and LAN scenarios, the *Remote Host Discovery* module finds remote computers in the same sub-network. Each host evaluates the requirements against its resources and returns the resulting utility values. These are combined with host identification in a bounded ordered list by the local STARC daemon. In most situations, STARC needs only pick the top result or iterate over the list for parallel scheduling. In Grid infrastructures, we are currently implementing a STARC module within the framework of MDS4 [SPM+06] with a set of resource providers (for monitoring node resources and availability).

In the case of peer-to-peer cycle-sharing, the set of hosts made available to STARC to evaluate must be reduced and not the entire overlay population as this hinders scalability. In this scenario, STARC operates exclusively on two sets of hosts returned by the resource discovery mechanism in Ginger (described next in the Section): direct neighbors in routing tables, and those returned by a lower-level resource discovery mechanism seeking CPU, memory and bandwidth.

C. Ginger – Peer-to-peer discovery cycle-sharing, and work distribution

The Peer4Peer component that manages the distribution of computation is Ginger. Ginger is a middleware platform on a structured peer-to-peer overlay network (using the Pastry protocol [RD01]) that bases its operation around the concept of a gridlet (see Figure 4). A gridlet is a fragment of data, capable of describing all aspects of a work task, as well as the necessary changes for processing the data. When a job is submitted by an application for processing (e.g. to the Peer4Peer portal or directly to the overlay), it is partitioned into small tasks that are used to generate gridlets, which will be submitted to the overlay where they will be processed by other nodes. When the computation is complete, the results can be sent, in the form of gridlet-results directly to the sender node, becoming available in the overlay as cached results, or sent to nuBOINC servers.

The Ginger overlay, based on Pastry [RD01], is a structured overlay that offers scalable and efficient peer-to-peer overlay routing, and leverages peer-to-peer properties such as self-organization of nodes, completely decentralization and fault-tolerance. The Pastry protocol provides Ginger with a neighbour set for each node created with a heuristic proximity that includes a limited number of the geographically nearest nodes. Ginger, itself, can be tested on a simulator that can also connect to and monitor, as a participant, an actual peer-to-peer overlay.

In the case of Peer4Peer, only a specific type of application is targeted: the actual simulation tools, in particular, P4PSim. Therefore, gridlets correspond to simulations included in the jobs submitted to nuBOINC. They carry the necessary information for donor nodes to retrieve topology, protocol and experiment data (if they are smaller than 256KB, they are embedded in the actual gridlet), as well as its estimated cost as requirements (expressed with STARC).

In the case of Peer4Peer, *Gridlet Management* amounts to create gridlets corresponding to the individual simulations requested in a job (e.g., by varying parameters given to the simulation, being topology, protocol, overlay size,...) and reassembling the results under the same job ID, confirming all gridlets have been executed (if not they can be retried/resubmitted), and storing all the results in a compressed folder. The results can be maintained cached in the overlay, as in PAST [RD01a], fostering load-balancing, flexibility, and availability.

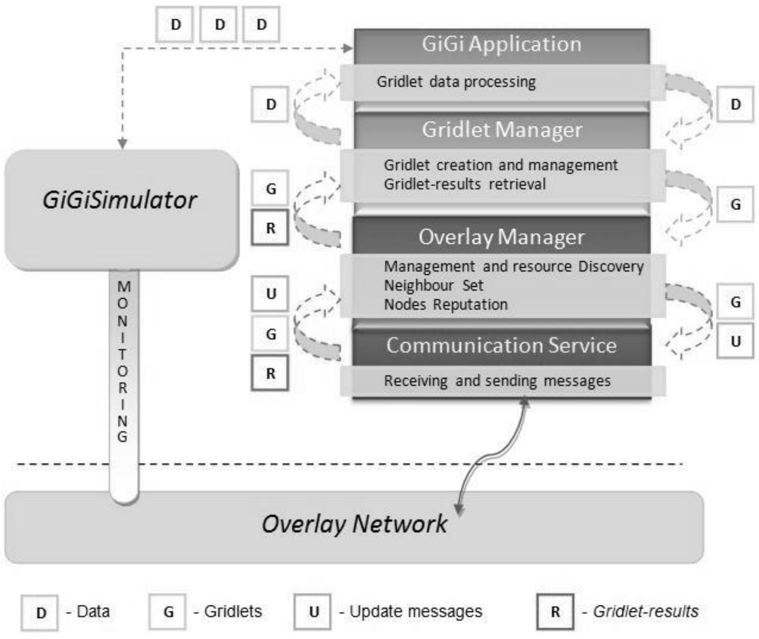

Figure 4: Ginger Architecture

Besides basic routing and overlay maintenance, the *Overlay Manager* is responsible for making the resources shared by donor nodes connected to the overlay available and engaged efficiently. This is achieved by employing a tailored resource discovery mechanism. The importance of network topology stems from the fact that resource discovery mechanisms follow the links formed by the peer-to-peer overlay network topology. By sending update-type messages, each node will announce its resources, only to those nodes that belong to the node's neighbour set. When a request is submitted by a node, it checks the information provided by its neighbours and forwards the gridlet to the node that seems more capable of process the gridlet. The main goal in Ginger is the correct routing of requests having into account the associated computation cost and various performance criteria that define the best choice, as the memory available, or bandwidth of the connection or other available resources in the node.

When a gridlet is received from the *Gridlet Manager*, a node with sufficient available resources to address this gridlet is selected as target for its routing. The statistics results of choosing that node for routing a request are stored in a table of reputation regarding that node. The *Overlay Manager* uses this table when, on the whole neighbour set there are no availability that meet the cost of a gridlet. In this case, the selection of a node to route the gridlet is based on the reputation table that as information regarding previous statistics results, as cases of failure and who had less delay in processing the applications. Globally, the definition of a node with better availability is one that has higher availability according to a weighted measure of the defined metrics (proximity, CPU, memory and bandwidth). Each metrics used to define the available resources contribute, in general, with similar weight in the weighted calculation of a node's availability.

D. P4PSim – scalable and efficient peer-to-peer simulation

P4PSim is an extension (and partial reimplementation) of PeerSim, able to make use of multiple concurrent execution threads to perform topology simulations, instead of the serial version that is currently available. It partitions the topology in a configurable number of partitions, predefined with the number of available cores in a machine (an aggregate number of cores may be used across a group or cluster of machines). Each simulated node is assigned to a partition and marked with a immutable "color". Nodes can be randomly assigned to partitions or assigned taking the simulated topology connectivity in account, favoring the placement of neighboring nodes in the same partition. In the latter case, boundary nodes may have more than one color.

Each partition is assigned a thread or a core. All nodes belonging to the same partition are simulated by the same thread, avoiding the need for blocking synchronization. This way, all cores are most of the time fully used simulating the nodes in their partitions. When interactions among nodes in different partitions occur (i.e., sending a message), a synchronized queue for each partition is used.

P4PSim instances can be run in distributed manner in two ways: i) in clusters, with Terracotta middleware [Ter10] that provides mostly transparent distributed shared memory for Java applications, ii) over wide area by using Java RMI. In this latter approach, currently, each group of P4PSim instances synchronizes at the end of each simulated round. In this case, there is increased latency, but that is compensated for the higher available memory for much larger simulations and, because of their increased size, the multiple CPUs available for parallel simulation in each simulation round.

Regarding communication between simulated nodes, it is performed using two distinct communication mechanisms: i) when nodes are simulated on the same P4PSim instance, or over Terracotta, a shared memory abstraction is used, while ii) if nodes are being handled by different P4PSim instances over wide area network (mostly for nodes on partition boundaries), simulated batches are batched in RMI invocations.

E. Other issues: security, heterogeneity

The Peer4Peer platform aims at portability and leverages heterogeneous platforms by relying mostly on the Java and Python language, with users and donors being able to manage the privileges associated with the VMs running on their machines.
In the specific case of resource monitoring performed by STARC, it is also executed at host nodes within the boundaries of a virtual machine. This way, access to local resources (e.g., file system, communication, etc.) can be limited and configured. This extends to the vast majority of

Environment Probing Classes. If one needs to access the native system directly, it can be subject to individual configuration and will not be executed without user's authorization. Finally, in the context of this work, STARC only schedules tasks associated with the execution of P4PSim instances that relies solely on Java.

7. Evaluation

The evaluation of the current Peer4Peer implementation is designed to show its feasibility regarding two main aspects: i) ability to leverage available resources in donor nodes across peer-to-peer cycle-sharing overlays (the most decentralized and scalable scenario), and ii) the ability to perform simulations faster, by efficiently employing multicore CPUs that are becoming prevalent in today's machines, even at laptop and desktop home machines.

A. Resource discovery and gridlet routing in Ginger

To measure and evaluate the performance of resource discovery (i.e., gridlet routing to donor nodes) in the Ginger P2P cycle-sharing overlay, we resorted to tests consisting of simulating the flow of messages throughout the network on two distinct scenarios. For the purposes of these tests, simulation was performed resorting to unmodified serial simulation.

There are 1000 nodes in the simulated system, divided into 2 groups of 500 nodes each: i) a group of host nodes, which provides its processing cycles for performing work from others, and ii) another group of client nodes, who will submit requests to take advantage of idle cycles available in the overlay. Two aspects that influence forwarding selection were tested: a) measurement of the information about the neighbour's availability; b) the ability that nodes have to learn about their neighbours by keeping historical records of statistics results (a simple measure of reputation).

In the first test (Test A), the information about the availability of a node is obtained from a weighted measure on the node's resources and its proximity to the local node. For effective measurements, tests should cover scenarios where the there are enough resources, where the availability in the whole network meets the demand (point of saturation); and situations of excessive demand that the network can not immediately overcome. Thus, the number of gridlets submitted to the overlay ranges over 300, 500 (1st point of saturation), 700, 900, 1000 (2nd point of saturation) and 1200 gridlets. The major goal of this test is to evaluate the performance and the efficiency of resource discovery approaches, namely regarding the number of messages needed to find a node able to execute a gridlet.

Three resource discovery variations were employed, regarding the metrics that decide routing: i) CPU, memory and bandwidth, ii) proximity in the node selection calculation, and iii) both combined. The first calculation only evaluates the availability in terms of resources, distributing the weight equally for the metrics: 33% for the CPU, 33% for memory and 33% for bandwidth. The second calculation only evaluates the proximity of the node. And finally, a last calculation weights the two measures, favouring resources: 40% to proximity and 60% to resources shared equally for each metric in 20%.

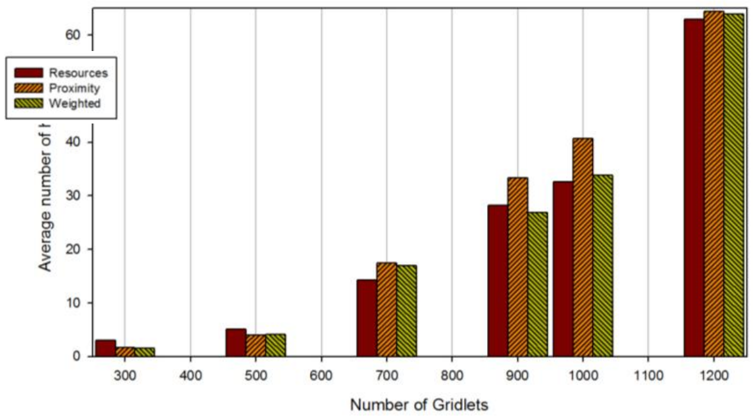

Figure 5: Average number of hops to serve a gridlet in Test A

From Figure 5, it is possible to observe the resource discovery quality obtained by the three calculations. It shows that for a number of gridlets less than or equal to 500 the calculation based only in resources obtained the worst results, but above the 500 requests it has improved, compared with the performance of the other calculation variations. Note that until the point of saturation (500) there is no lack of resources in the network, but from that point on, the lack of resources is a constant. Therefore, we can infer that the calculation based on resources is favourable for situations of solicitation in scarcity in resources. The calculation based on the nodes proximity, has good quality efficiency as long as there are many resources available on the network and very low quality in situations of immediate lack of resources, since it is from 700 gridlets submitted on, that results start to decline.

The second test (Test B), addresses the reputation maintained in each node, where they acquire information about the statistics results of their neighbours in the past, and learn the best ways as they send more requests to the network. The gains of using this mechanism can be obtained from the difference between the performance of the system with and without the execution of the reputation mechanism. In this test, the simulation is executed always with the same overlay network nodes and the tasks are all submitted from the same node. The number of tasks is always the same, 700 gridlets in order to test the behaviour of the system when there is a lack of resources on the network, and whether it can operate in a relevant scenario and influence the results.

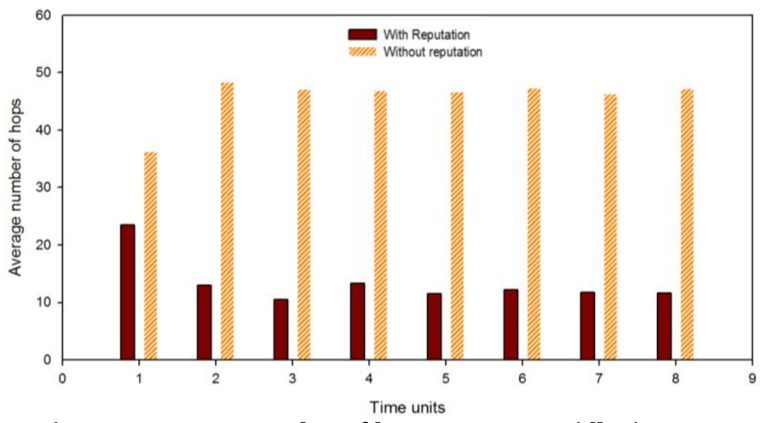

Figure 6: Average number of hops to serve a gridlet in Test B

According to Figure 6, it is possible to observe the routing quality obtained using the reputation mechanism. The smallest difference occurs during the first iteration since the reputation system has not yet acquired the information about their neighbours. In the second iteration, a large reduction in the number of retransmissions made is already visible, remaining at that level from that point on. Therefore, we can say that this mechanism converges very quickly to best performance.

B. Paralell simulation in P4PSim

P4PSim allows each execution thread to run the simulation of a subset of the simulated peer-to-peer network. The efficient parallelization of any application involves minimizing contention among concurrent tasks. In the case of P4PSim, it is essential to make sure that the simulated nodes assigned to an execution thread are as contiguous as possible so as to reduce locking and data transfer. As already mentioned, in order to minimize communication and contention between simulation threads (that may even run on different machines), the nodes of the simulated overlay can be assigned using an affinity metric, usually the number of hops between nodes, thereby resulting in the clustering of close simulated nodes in the same execution node.

We aimed to demonstrate the performance benefits when simulating the behaviour of very large network overlays. Figure 7 depicts the evolution of execution times as the number of nodes increases for two different operating modes of P4PSim (excluding partitioning time). The partitioning process is the mechanism of assigning simulated nodes to execution nodes. In the random version, simulated nodes are randomly assigned to execution nodes, whereas in the partitioned version, the set of simulated nodes is partitioned based on proximity. The algorithm that is being simulated is the periodical exchange of a monotonically increasing value stored by the nodes of a Pastry [RD01] network.

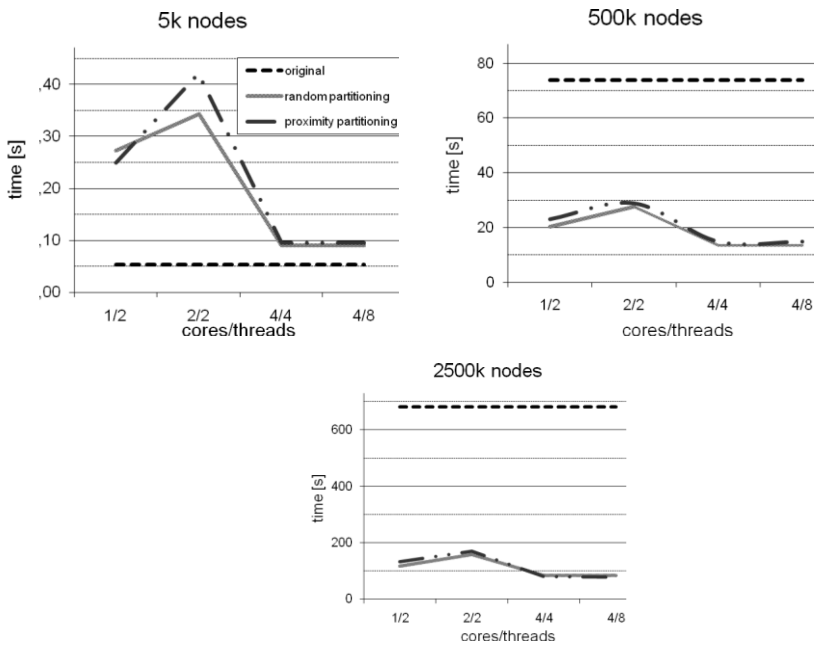

Figure 7: P4PSim parallel simulation execution times

The partitioning algorithms and simulation distribution are still being perfected but the benefits of executing the simulator concurrently are already amply apparent. As depicted in Figure 7, for small simulations (5k nodes), the overhead of synchronizing the access to shared data erodes the benefits of parallelization and therefore the execution times for parallel executions are between twice and eight times higher. However as the number of simulated nodes grows parallel execution becomes more efficient: for 2.5 million nodes, parallel execution is 4 to 8.6 times faster.

For the small number of threads and processors we experimented with, we observe that adding threads has a greater impact than adding cores. However, this effect will also be attenuated as the number of simulation execution threads increases.

8. Related Work

This section covers relevant work on the areas related with the scope of this work and its presiding vision: A) Simulation tools/test-beds, B) Parallel/distributed computing, C) Cycle-providing infrastructures, D) Peer-to-peer overlays, and E) Grid middleware.

A. Simulation tools and test-beds

Simulation facilitates design and improves productivity, reliability, avoiding costs in deployment. Tools offer different semantics/abstraction levels in simulation configuration and execution. Lower-level is used in electronics design [FFL05], computer architecture (Archer [FBF+08]) and network simulation (NS2 [IH09]). It allows fine-grained detail to define structure and behavior often with domain-specific languages. It provides precision in measurements and monitoring, with physical world phenomena (voltage, transmission delays).

Higher-level semantics offers lower overhead and addresses sophisticated behavior described with more abstract rules, yet defined programmatically, e.g., neighbor-sets and routing of peer-to-peer overlay topologies (Peersim, Oversim [BHK07]), distributed computing running Grid middleware (Simgrid [CLQ08], GridSim [BM02]), and general distributed systems over TCP [T09]. Yet, they execute centrally managing event queues limiting simulation size and complexity. Test-beds (PlanetLab [CCR+03], Emulab [WLS+02], SatteliteLab [DHB+08]) do allow parallel execution on geographically distributed nodes. Nonetheless, overhead is orders of magnitude higher (few rules vs. millions of instructions) in practice limiting simulation size and complexity [T09].

Our project differs from these by combining higher-level semantics with parallel simulation distributed over actual cycle-sharing overlay infrastructures. More specifically, it tackles the apparent incompatible tradeoff between the advantages of simulation tools and testbeds (i.e., simulations size vs. parallelism). Although testbeds (e.g., in the specific case of PlanetLab) can execute participating nodes in parallel and distributed fashion, their overhead is too great (usually requiring a complete virtual or real testbed machine running OS, Java VM and P2P protocol for each participating node). This clearly limits simulations to at most hundreds of simulated nodes. In Peer4Peer, simulation sizes can be much larger as each real machine may be simulating tenths of thousands of participating nodes, also in parallel fashion.

In the case of overlay and Grid simulation tools, they are centralized, i.e., for a given topology being simulated, the simulation tool can only leverage the available resources of a single machine. This clearly limits scalability as desktop/server computers' memory is bounded (even if larger than 8 or 16 MB), restricting the number of simulated nodes to tens of thousands (which is not realistic today as stated previously). Moreover, in the specific case of P2P simulations, usually larger sized, the simulation tools are actually of serial execution, which makes simulations of larger or more complex topologies very slow. In Peer4Peer, simulations are performed by P4PSim resorting to multiple threads (and processes), overcoming memory limitations and taking advantage of today's multicore machines.

B. Parallel and Distributed Computing

Concurrent activities are implemented as processes/threads/tasks cooperating with data sharing and coordination. Coordination ranges from stricter shared-memory (and DSM), looser message-passing (MPI [F95]) to rarefied Bags-of-Tasks (Ourgrid [BAV+07]). Activities may share data globally (DSM), partially in communication (MPI, MapReduce) or never in BoTs. Sharing eases programming avoiding explicit communication. Still, it requires consistency enforcement models.

Pessimistic approaches with locks/semaphores and DSM protocols [ZIS+97] have poor performance and little scalability. Optimistic approaches allowing temporary inconsistencies are essential for performance in large scale. In Software Transactional Memory [HLM+03], consistency is enforced at commit. Eventual consistency [TTP+95] relies on ulterior reconciliation.

A number of recent middle-ground approaches do allow data divergence limited in time and scope [YV06], use locality-awareness [CWD+05], or both (vector-field consistency [SVF07]). While cluster-based multiplayer gaming is a key current scenario, overlay simulation is still unaddressed.

C. Cycle-providing infrastructures

Early supercomputers linked CPUs by internal bus. Today, multi-cores equip cheaper, more compact multiprocessor computers. Previous increase of LAN speed fostered aggregating (even heterogeneous) computers into cluster systems (NOW [ACP95], PVM [S90]). Beyond access to remote clusters or supercomputers, users access computational grids [FK99] and shared/donated cycles (BOINC [AF06]).

Grid access implies belonging to virtual organizations of institutions exchanging computational resources. Underlying middleware (Globus [F06]) handles all authentication, accounting and resource allocation. In cycle-sharing, idle CPU time of PCs over the Internet is used for data processing, e.g., biology, astronomy. Cycle-sharing is integrated with peer-to-peer techniques to federate clusters or build desktop grids (CCOF [LZZ+04]), Ourgrid, Ginger [VRF07]).

Utility computing (Amazon EC2) employs virtualization techniques for on-demand execution of several VMs in a single real computer. Users launch VMs and create virtual clusters. Currently, however, no infrastructure enables simulation tools for faster or more complex simulations.

D. Peer-to-peer overlays

Peer-to-peer is effective in file-sharing, audio/video streaming, content/update dissemination, anonymity and cycle-sharing [ATS04]. peer-to-peer overlays are split in two dimensions: structure and centralization, with hybrids in both. Structured (Chord,Pastry,CAN,Viceroy) map content to node identification with locality gains, load-balancing, ensured and range-based lookup queries.

Unstructured (Gnutella,FreeNet) allow unrestricted content placing and arguably tolerate node entry/exit churn (partially debunked in [CCR05]). Most peer-to-peer are mostly or fully decentralized. Resource discovery [TTP+07] in peer-to-peer cycle-sharing finds computational resources: fixed (capabilities, applications) and transient (CPU/memory availability).

Structured approaches adapt topologies to map resource descriptions/locations as searchable content either flat, layered or N-dimensional [CFC+03]. Unstructured also centralize in super-peers, trees to speed search and minimize message flood, gossip.

Peer4Peer resorts to Ginger, a overlay hybrid of: structured for storage, result caching, capability matching; and unstructured for CPU availability, task forwarding in vicinities; super-peers for reputation and accounting. Cycle-sharing has not been harnessed before for larger scale overlay and Grid simulation.

E. Grid middleware

Grid initiatives not only mediate access to remote computing resources but also try to integrate into portals the tools needed by researchers. The LHC computing Grid allows access to scientific data, processing power and also provides interfaces to simulation and analysis tools used by the physics community.

A relevant area is the development of user interfaces (Ganga [HLT+03]) to interact with the available tools. Other initiatives (Archer, NanoHub [KMB+08]) develop simulation tools and portals for easy access to them for configuration, execution results and public data. Usefulness for the scientific community (integrated tools, data, common interfaces) fosters a large educational potential. A corresponding initiative will have similar impact in research reproducibility in overlay [NLB+07] and Grid communities.

9. Conclusion

This chapter presents the case for the use of a shared computing platform as the base for an e-Science community: the peer-to-peer overlay (P2P) and Grid middleware research community. The specific scientific problems tackled by this community, and how they are treated, are a great motivator for the establishment of an e-Science community: research work is based on simulation, most simulations are executed on a number of varied, yet well established tools (e.g., PeerSim, Simgrid), the simulations test the performance of diverse network protocols in multiple scenarios, and results are usually compared with previous competitors. All these characteristics make this community fit well on mixed environment of grids, clusters, and cycle-sharing overlays.

Unfortunately, the current implementations of peer-to-peer simulators and the management of institutional clusters reduce the available computing resources (and their effective utilization) for research in this field. Furthermore, resources scattered on the Internet (accessed by means of a BOINC-like distributed computing infrastructure) allow the execution of more simulations, but without users being able to increase the complexity (number of nodes and their interactions) of the topology being simulated.

To tackle these architectural problems we presented a platform, Peer4Peer that provides the main infrastructure for efficient peer-to-peer simulation on the Internet. This efficient simulation is addressed in three axes: i) concurrent execution of independent simulations, ii) parallelization of single simulations, and iii) use of distributed computers (on the Internet, on clusters, or on utility computing infrastructures) to speed up simulations and to allow the simulation of more complex scenarios.

The architecture of Peer4Peer is composed by: i) a job management and portal services using nuBOINC to submit job descriptions, data, and simulation topologies and protocols using a topology modeling language (TML), ii) expressive resource requirement description and resource discovery coordination using STARC, iii) peer-to-peer cycle-sharing infrastructure, Ginger, offering decentralized resource discovery, distribution of computation and decentralized data caching; and iv) P4PSim to run protocol simulations parallelized, taking advantage of multicore machines, with support for distribution to allow for larger and more complex simulations.

Current performance results are already very encouraging as we are able to achieve faster parallel simulations, and through the usage of cooperating P4PSim instances. Peer4Peer is able to handle larger simulations than on a single dedicated machine. Thus, we successfully speedup and scale overlay simulations with Peer4Peer.

References

[ACK+02] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, D. Werthimer. SETI@home: An Experiment in Public-resource Computing. Commun. ACM 45, 11. November 2002.

[ACP95] T. E. Anderson, D. E. Culler, D. A. Patterson A Case for Networks of Workstations: NOW. IEEE Micro vol. 15-1, pp. 54-64. 1995.

[AF06] D. P. Anderson and G. Fedak. The computational and storage potential of volunteer computing. In IEEE/ACM Intl. Symposium on Cluster Computing and the Grid. May 2006.

[AGK00] D. Abramson, J. Giddy, L. Kotler. High Performance Parametric Modeling with Nimrod/G: Killer Application for the Global Grid?. Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS 2000). USA. 2000.

[AMA10] Amazon.com, Inc.: Amazon elastic compute cloud, http://aws.amazon.com/ec2. 2010.

[ATS04] S. Androutsellis-Theotokis, D. Spinellis A Survey of Peer-to-Peer Content Distribution Technologies. ACM Computing Surveys, vol. 36-4, p.371. 2004.

[BAU07] I. Baumgart, B. Heep, S. Krause. OverSim: A Flexible Overlay Network Simulation Framework. In Proc. Of the IEEE Global Internet Symposium, pp.79-84. 2007.

[BAV+07] F. Brasileiro, E. Araujo, W. Voorsluys, M. Oliveira, F. Figueiredo. Bridging the High Performance Computing Gap: the Ourgrid Experience. CCGRID 2007 Seventh IEEE International Symposium on Cluster Computing and the Grid. Rio de Janeiro, Brazil. 2007.

[BHK07] Ingmar Baumgart, Bernhard Heep, Stephan Krause. OverSim: A Flexible Overlay Network Simulation Framework. Proceedings of 10th IEEE Global Internet Symposium (GI '07) in conjunction with IEEE INFOCOM 2007. 2007.

[BM02] R. Buyya, M. Murshed. GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing. Concurrency and Computation: Practice and Experience (CCPE), Volume 14, Issue 13-15. Wiley Press 2002.

[CAS01] H. Casanova. Simgrid: A Toolkit for the Simulation of Application Scheduling. First IEEE/ACM International Symposium on Cluster Computing and the Grid. 2001.

[CCR+03] Brent N. Chun, David E. Culler, Timothy Roscoe, Andy C. Bavier, Larry L. Peterson, Mike Wawrzoniak, Mic Bowman: PlanetLab: an overlay test-bed for broad-coverage services. Computer Communication Review 33(3): 3-12. 2003.

[CCR05] M. Castro, M. Costa, A. Rowstron. Debunking some Myths about Structured and Unstructured Overlays. Symposium on Networked Systems Design & Implementation. Volume 2, p.98. 2005.

[CFC+03] M. Cai, M. Frank, J. Chen, P. Szekely. MAAN: A Multi-attribute Addressable Network for Grid Information Services. Proc. 4th Int. Workshop on Grid Computing, GRID 2003. 2003.

[CKB+07] S. Choi, H. Kim, E. Byun, M. Baik, S. Kim, C. Park, C. Hwang. Characterizing and Classifying Desktop Grid. Proceedings of the Seventh IEEE international Symposium on Cluster Computing and the Grid (May 14 - 17, 2007). 2007.

[CLQ08] H. Casanova, A. Legrand, M. Quinson. Simgrid: a Generic Framework for Large-Scale Distributed Experiments. 10th IEEE International Conference on Computer Modeling and Simulation. 2008.

[CWD+05] J. Chen, B. Wu, M. Delap, B. Knutsson, H. Lu, C. Amza. Locality Aware Dynamic Load Management for Massively Multiplayer Games. Proceedings of the tenth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. 2005.

[DHB+08] Marcel Dischinger, Andreas Haeberlen, Ivan Beschastnikh, P. Krishna Gummadi, Stefan Saroiu. Satellitelab: Adding Heterogeneity to Planetary-scale Network Test-beds. SIGCOMM 2008: 315-326. 2008.

[DSA10] Distributed Systems Architecture Group, Universidad Complutense de Madrid. GridWay Metascheduler http://www.gridway.org/. 2010.

[EH08] C. Evangelinos, C. N. Hill.. Cloud Computing for Parallel Scientific HPC Applications: Feasibility of Running Coupled Atmosphere-Ocean Climate Models on Amazon's EC2. Proceedings of Cloud Computing and its Applications. http://www.cca08.org. 2008.

[EI10] Enomaly Inc. Enomalism : Elastic computing platform - virtual server management. http://enomalism.com.

[F06] 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. 2006.

[F95] I. Foster. Designing and Building Parallel Programs (chapter 8 Message Passing Interface). ISBN 0201575949 Addison-Wesley. 1995.

[FBF+08] R. J. O. Figueiredo, P. O. Boykin, J. A. B. Fortes, T. Li, J. Peir, D. Wolinsky, L. K. John, D. R. Kaeli, D. J. Lilja, S. A. McKee, G. Memik, A. Roy, G. S. Tyson. Archer: A Community Distributed Computing Infrastructure for Computer Architecture Research and Education. 2008.

[FFL05] J. A. B. Fortes, R. J. Figueiredo, M. S. Lundstrom. Virtual Computing Infrastructures for Nanoelectronics Simulation Proceedings of the IEEE. 2005.

[FFM07] M. Feller, I. Foster, and S. Martin. GT4 GRAM: A Functionality and Performance Study. 2007.

[FK99] Foster; C. Kesselman. The Grid: Blueprint for a New Computing Infrastructure. Morgan-Kaufman. 1999.

[FOS02] I. Foster. The Grid: A New Infrastructure for 21st Century Science. Physics Today, 55(2):42-47. 2002.

[FR92] D. G. Feitelson, L. Rudolph. Gang Scheduling Performance Benefits for Fine-Grain Synchronization. Journal of Parallel and Distributed Computing, volume 16, pages 306—318. 1992.

[GEN01] W. Gentzsch. Sun Grid Engine: Towards Creating a Compute Power Grid. In Proceedings of the 1st international Symposium on Cluster Computing and the Grid. May 15 – 18. 2001.

[GLU10] Gluster Inc. GlusterFS. http://www.gluster.org. 2010.

[HEN95] R. L. Henderson.. Job Scheduling Under the Portable Batch System. In Proceedings of the Workshop on Job Scheduling Strategies For Parallel Processing. 1995.

[HLM+03] Maurice Herlihy, Victor Luchangco, Mark Moir, and William N. Scherer III. Software Transactional Memory for Dynamic-Sized Data Structures. Proceedings of the Twenty-Second Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing 2003. 2003.

[HLT+03] K. Harrison, W. T. L. P. Lavrijsen, C. E. Tull, P. Mato, A. Soroko, C. L. Tan, N. Brook, R. W. L. Jones. GANGA: a User-Grid Interface for Atlas and LHCb in Proceedings of Computing in High Energy and Nuclear Physics. La Jolla, 2003.

[HOW88] Howard, J.H., Kazar, M.L., Nichols, S.G., Nichols, D.A., Satyanarayanan, M., Sidebotham, R.N., & West, M.J.. Scale and Performance in a Distributed File System. *ACM Transactions on Computer Systems* **6** (1): 51–81. February 1988.

[IH09] T. Issariyakul, E.Hossain Introduction to Network Simulator NS2. ISBN: 978-0-387-71759-3 Springer Books 2009.

[JELA10] M. Jelasity, A. Montresor, Gian-Paolo Jesi, Spyros Voulgaris. The Peersim Simulator. http://peersim.sf.net.

[JN99] J.P. Jones, B. Nitzberg. Scheduling for Parallel Supercomputing: A Historical Perspective of Achievable Utilization. Job Scheduling Strategies for Parallel Processing (JSSPP), pp. 1-16, 1999.

[KA99] Y. K. Kwok and I. Ahmad. Static Scheduling Algorithms for Allocating Directed Task Graphs to Multiprocessors. ACM Comput. Surv., 31(4):406–471, 1999.

[KMB+08] G. Klimeck, M. McLennan, S. P. Brophy, G. B. Adams III, M. S. Lundstrom. nanoHUB.org: Advancing Education and Research in Nanotechnology. Computing Science and Engg. 10, 5 (Sep. 2008), 17-23. 2008.

[LLM88] M. Litzkow, M. Livny, M. Mutka, Condor: a Hunter of Idle Workstations. In Proceedings of the Eighth International Conference of Distributed Computing Systems, pp. 104-111. San Jose, CA, USA. 1988.

[LEA96] Leach, P. and Perry, D.. Cifs: A common internet file system. Microsoft Interactive Developer. 1996.

[LZZ+04] V. Lo, D. Zappala, D. Zhou, Y. Liu, and S. Zhao. Cluster Computing on the Fly: P2P Scheduling of Idle Cycles in the Internet. In The 3rd International Workshop on Peer-to-Peer Systems (IPTPS'04) 2004.

[NLB+07] S. Naicken, B. Livingston, A. Basu, S. Rodhetbhai, Ian Wakeman, Dan Chalmers. The State of Peer-to-Peer Simulators and Simulations. Computer Communication Review, 2007.

[NRW04] J. Novotny, M. Russell, O. Wehrens, 2004. GridSphere: a Portal Framework for Building Collaborations: Research Articles. Concurr. Comput. : Pract. Exper. 16, 5: 503-513, Apr. 2004.

[OPL10] OpenNebula Project Leads. OpenNebula, The Open Source Toolkit for Cloud Computing, http://www.opennebula.org/start . 2010.

[OSU82] J. K. Ousterhout. Scheduling Techniques for Concurrent Systems. Proceedings of Third International Conference on Distributed Computing Systems, 22—30. 1982.

[PLA+06] Plale, B., D. Gannon, J. Brotzge, K. Droegemeier, J. Kurose, D. McLaughlin, R. Wilhelmson, S. Graves, M. Ramamurthy, R.D. Clark, S. Yalda, D.A. Reed, E. Joseph, V. Chandrasekar, CASA and LEAD: Adaptive Cyberinfrastructure for Real-Time Multiscale Weather Forecasting, Computer (Special issue on System-Level Science), IEEE Computer Science Press, Vol. 39, No. 11, pp. 56-63, Nov. 2006.

[RD01] A. Rowstron, P. Druschel. Pastry: Scalable, Distributed Object Location and Routing for Large-scale Peer-to-Peer Systems. IFIP/ACM International Conference on Distributed Systems Platforms (Middleware).Vol. 11, pp. 329-350. 2001.

[RD01a] A. Rowstron e P. Druschel, "Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility," ACM SIGOPS Operating Systems Review, vol. 35, 2001, pp. 188-201.

[S90] V. S. Sunderam. PVM: a Framework for Parallel Distributed Computing. Concurrency: Practice and Experience, Volume 2, Issue 4, pp. 315-339. 1990.

[SCH07] B. Schmidt. A Survey of Desktop Grid Applications for E-science. Int. J. Web Grid Serv. 3, 3 (Aug. 2007), 354-368. 2007.

[SH02] F. Schmuck, R. Haskin. GPFS: A Shared-Disk File System for Large Computing Clusters. In Proceedings of the 1st USENIX Conference on File and Storage Technologies, January 28 - 30, 2002.

[SNS+02] T. Suzumura, H. Nakada, M. Saito, S. Matsuoka, Y. Tanaka, S. Sekiguchi, The Ninf portal: an automatic generation tool for grid portals. In Proceedings of the 2002 Joint ACM-ISCOPE Conference on Java Grande (Seattle, Washington, USA, November 03 - 05, 2002). JGI '02. ACM, New York, NY, . 2002.

[SPM+06] J. Schopf, L. Pearlman, N. Miller, C. Kesselman, I. Foster, M. DArcy, and A. Chervenak, “Monitoring the grid with the Globus Toolkit MDS4,” in Journal of Physics: Conference Series, vol. 46, no. 1. Institute of Physics Publishing, 2006, pp. 521–525.

[SVF08] J. Silva, L. Veiga, P. Ferreira. nuBOINC: BOINC Extensions for Community Cycle Sharing. Workshop on Decentralized Self Management for grids, P2P, and User Communities - SELFMAN (in in conjunction with SASo 2008). Venice, Italy. October 20-24, 2008.

[SVF10] J. N. Silva, L. Veiga, P. Ferreira. Service and Resource Discovery in Cycle Sharing Environments with an Utility Algebra 24th IEEE International Parallel & Distributed Processing Symposium, Atlanta (Georgia) USA. April 19-23, 2010.

[T09] Chunqiang Tang. DSF: A Common Platform for Distributed Systems Research and Development. In Proc. of Middleware 2009, pp. 414-436. 2009.

[Ter10] Terracotta, http://www.terracotta.org/, 2010

[TOP500] Top 500 Supercomputers. http://www.top500.org . 2010.

[TTP+07] 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, archive Volume 23. 2007.

[TTP+95] Terry, D.B., Theimer, M.M., Petersen, K., Demers, A.J., Spreitzer, M.J., Hauser, C.. Managing Update Conflicts in Bayou, a Weakly Connected Replicated Storage System. Proceedings of the fifteenth ACM Symposium on Operating Systems Principles. 1995.

[TWS+05] Ian J. Taylor, Ian Wang, Matthew S. Shields, Shalil Majithia: Distributed computing with Triana on the Grid. Concurrency and Computation: Practice and Experience 17(9): 1197-1214, 2005.

[VRF07] L. Veiga, R. Rodrigues, P. Ferreira. GiGi: An Ocean of Gridlets on a "Grid-for-the-Masses”. Seventh IEEE International Symposium on Cluster Computing and the Grid, CCGRID 2007. Rio de Janeiro, Brazil. May 14-17, 2007.

[WLS+02] Brian White, Jay Lepreau, Leigh Stoller, Robert Ricci, Shashi Guruprasad, Mac Newbold, Mike Hibler, Chad Barb, Abhijeet Joglekar. An Integrated Experimental Environment for Distributed Systems and Networks. OSDI 2002.

[YV06] H. Yu, A. Vahdat. The Costs and Limits of Availability for Replicated Services. ACM Transactions on Computer Systems. 2006.

[ZIS+97] Yuanyuan Zhou, Liviu Iftode, Jaswinder Pal Singh, Kai Li, Brian R. Toonen, Ioannis Schoinas, Mark D. Hill, David A. Wood. Relaxed Consistency and Coherence Granularity in DSM Systems: A Performance Evaluation. Sixth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. 1997.

Acknowledgments: The authors wish to thank students Filipe Paredes and João Neto for their implementation efforts. This work was supported by FCT (INESC-ID multiannual funding) through the PIDDAC Program funds, and projects PTDC/EIA-EIA/102250/2008, PEst-OE/EEI/LA0021/2011.