Service Migration within the Cloud: Code Mobility in SP2A

Michele Amoretti, Maria Chiara Laghi, Fabio Tassoni, Francesco Zanichelli
Department of Information Engineering
University of Parma
Viale Usberti 181/a, Parma, Italy
michele.amoretti@unipr.it

ABSTRACT

Cloud Computing (CC) is characterized by dynamically scalable and often virtualized resources that are provided as a service over the Internet. To date, no generally agreed scheme for the internal organization of a cloud has emerged, and multiple approaches have been proposed, ranging from traditional N-tier to peer-to-peer ones.

In this paper we illustrate our CC approach based on service mobility, that allows systems to cope with highly dynamic environmental conditions. We illustrate a recent extension to our SP2A middleware, that enables peer-to-peer CC systems characterized by services that migrate on-demand. Moreover, we propose and discuss the results of the deployment of a real SP2A-based CC system in PlanetLab.

KEYWORDS: cloud computing, scalable computing, peer-to-peer.

1. INTRODUCTION

Cloud Computing (CC) can be considered a new computing paradigm that allows users to temporary utilize computing infrastructure over the network, supplied as a service by the cloud-provider at possibly one or more levels of abstraction [8].

After VMware announced its acquisition of SpringSource, it seemed like CC would emerge as ”clash of the titans” with billion-dollar players VMware, Amazon, Microsoft, etc. leaving little room for smaller companies. Fortunately, such analysis is superficial and incomplete. While the benefits of CC are enormous in terms of reducing costs, increasing utilization, and providing scalability, the early state of the technology and the lack of standards could provide an opportunity for other players to introduce an open cloud architecture that gains center stage.

In this context, the idea of highly dynamic clouds appears appealing, with large, stable service providers side by side with small to medium service providers that join and leave dynamically. Even clients may contribute with their resources, on a volunteer basis. In our opinion, such large-scale, complex clouds cannot be managed by a central authority. On the contrary, a peer-to-peer approach should be adopted and to this purpose we propose the SP2A [1], which is both a framework and a middleware for peer-to-peer service-oriented architectures.

An academic research project that supports the idea of dynamic clouds is Cloud@Home [4], that defines a democratic form of CC in which the resources of users accessing the Cloud can be shared in order to contribute to the computing infrastructure. The proposed solution allows users to overcome both hardware and software compatibility problems of Volunteer computing and, in a commercial context, it can establish an open computing-utility market where users can both buy and sell their services. With respect to Cloud@Home, our framework adopts distributed mechanisms for service sharing, and introduces novel mechanisms to dynamically deploy, un-deploy and migrate services, based on code mobility techniques, for dynamic load balancing and QoS preservation.

This paper focuses on service migration within SP2A-based clouds. In section we illustrate how service migration is enabled by SP2A. In section we show the results achieved from an experimental evaluation conducted on the PlanetLab [6] platform. Finally, in section we conclude the paper discussing achieved results and outlining future work.

2. SP2A-BASED CLOUDS

In the context of a Cloud where single users, as well as research groups, social communities, and public administrations make available their distributed computing resources, our framework enables peer-to-peer networks whose participants are allowed to search for services, to consume them but also to replicate them, thus increasing service availability.

The Service-oriented Peer-to-Peer Architecture (SP2A) [1] is a framework and a lightweight middleware based on the peer-to-peer paradigm, defining the basic modules for building service-oriented peers (SOPs) for efficient and robust Grid environments. SP2A allows designers to cope with the requirements of applications with a large number of users dynamically connecting to the system, and provides high levels of scalability, decentralization and interoperability.

The SP2A middleware is distributed as a set of Java interfaces and both J2SE and J2ME class implementations, which support state of the art technologies for peer-to-peer message routing, service description and deployment: Web Services, OWL-S and JXTA. These technologies complement each others: Web Services provide a framework for service description; OWL-S supplies service providers with a core set of markup language constructs for describing the properties and capabilities of their services in unambiguous, computer-interpretable form. JXTA protocols [7] operate at the lower level providing P2P functionalities.

Since its initial release, the SP2A middleware dictates that each resource must be provided as a service, thus matching one of the major concepts of Cloud Computing [1]. Indeed, to use a resource, a consumer peer must know whether the related Resource Provision Service (RPS) exists and is available; if it operates under a specified set of assumptions, constraints, policies; and if it can be invoked through a specified means, including inputs that the service requires and outputs that will form the response to the invocation. Each RPS is associated to a name, a WSDL description (and, optionally, a OWL-S one), and a uniquely identified owner, and it is packaged as a jar file for the purpose of service migration. In addition, they expose a WSDL description (and, optionally, a OWL-S description), which specifies how to access its functionalities. Service deployment is transparent to the user, which only has to invoke the `shareRPS()` method of the `RPSManager` singleton object. The latter encapsulates the creation of a JXTA service advertisement, and the related service instance in a new thread.

SP2A allows the sharing in the P2P network of both the descriptions of ”local” RPSs, *i.e.* services whose instances are intended to be running in separate threads, with respect to the SP2A-based peer application, and ”external” RPSs, *i.e.* services deployed in traditional containers (*e.g.* Axis servers) and addressed by simple URLs.

2.1. Mobile Service Problem and Pull Approach

According to the definition given by Carzaniga *et al.*, code mobility is the capability to reconfigure dynamically, at runtime, the binding between the software components of the application and their physical location within a computer network [3].

Mobile code technologies can be analyzed by considering the ability to transfer the state of an execution thread (or execution unit, to use a more general term). We recall that a technology supports *strong mobility* if it allows executing units to move their code and execution state (*e.g.*, the stack and instruction pointer of a thread) to a different site. When an executing unit must be transferred to a remote site, it is suspended, transmitted to the destination site, and resumed there. A technology supports *weak mobility* if it allows an executing unit in a site to be bound dynamically to code coming from a different site (*i.e.*, the code can be moved and executed automatically), but no execution state is transferred across the network. This means that even though some data state could be transferred, the executing unit would need to be restarted upon arrival [3].

The Mobile Service Problem we considered in the context of p2p-based Cloud Computing (illustrated in fig.1) is a *weak mobility* problem, because the execution unit that searches for the service is dynamically bound to the code coming from a different site, but no migration of execution state is involved. We created a *Code on Demand* architecture, where nodes discover remote services, obtain the related know-how (hosting requirements and packaged bytecode), and execute them within their own context.

Before downloading the service, the requesting peer must check that it has enough resources to execute it, to avoid network overload. The service provider exposes an XML file with minimal hardware (number of processors, memory) and software (OS type and architecture, Java version, external libraries) requirements for service execution. This file is converted to a string and appended to the *description* field of Resource Provision Service (RPS).

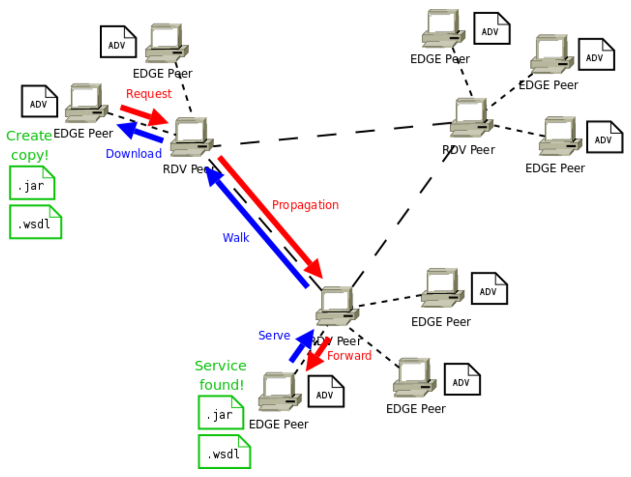

Figure 1. The Mobile Service Problem.

The Pull approach we devised, to cope with the Mobile Service Problem, involves the following steps:

- peer P_1 searches for service S
- service S is found on peer P_2
- P_1 checks the requirements for hosting service S
- if hosting requirements are met, P_1 obtains the $S.jar$
- P_1 locally activates S and becomes a new provider for S

2.2. Download Process

JXTA protocols provide three transport mechanisms for communication among peers: *Endpoint Service*, which is a point to point communication level, *Pipe Service*, which integrates the Endpoint Service with the abstraction of the virtual communication channel and *Jxta Socket*, at the higher level, that provides an interface similar to standard sockets' one. We used JXTA Sockets because they allow bidirectional secure communication; moreover they add an ACK NUMBER to the message that provides additional information to the payload to ensure reliability to the communication and to guarantee that data is received in the correct order. It is possible to configure the socket output buffer to reduce the number of packets and facilitate data reading operations.

The communication protocol for mobile Service Problem is described by the following algorithm (and illustrated in fig.2):

- the server, after deploying the service, waits for client connections through the *JxtaServerSocket* class
- a client connects to the server which provides the discovered service through the *JxtaSocket* class
- both client and server create an I/O stream for data exchange
- the client requests the download of the service
- the server communicates the .jar size and starts to send the file
- if the client receives the packet and is able to reconstruct the file it sends a notification message to the server
- the server closes the connection and waits for other requests
- the client shares the service for download

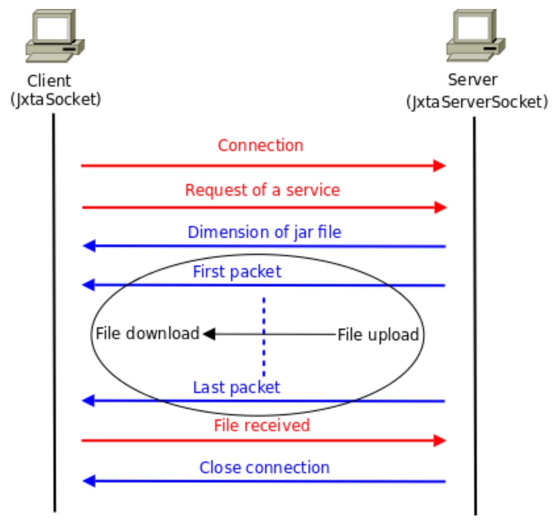

Figure 2. Communication Protocol for Service Download in SP2A.

During the discovery phase, results are stored in a vector, whose dimension gives information about the number of nodes found on a single discovery operation that are sharing the service for download: the greater the dimension is, the more the service is available in the network. The basic idea of our download protocol is to first try to download the service from the nodes listed in the actual vector and, if the vector is empty, or none of the peers are available at the moment, to invoke `findRPS()` method to start a new discovery in the JXTA peer-to-peer network.

Every time the `downloadRPS()` function is performed with the vector dimension as a parameter, a certain number of attempts to download the file from each provider is performed, with an increasing timeout. The number of attempts increases if the discovery results in the vector are scarce, and decreases if a lot of providers are found in order to have a greater probability to find an available node from which downloading the service.

In the implementation of our Pull solution for the Mobile Service Problem, we took advantage of the Java Reflection API to add in the runtime classpath of client the base class name of the downloaded service, and then to instantiate an object of the same class. Reflection mechanisms may be used also to implement security mechanisms, as explained in the following.

2.3. Security Issues

While adding flexibility in software design and bandwidth optimization to traditional distributed computing systems, mobile code systems increase vulnerability to malicious intrusions over the Internet. Possible vulnerabilities with mobile code belong to one of these categories: attacks performed by a mobile program against the remote host on which the program is executed and attacks due to the subversion of the mobile code and its data by the remote execution environment.

As described in [9], when protecting a host from potentially malicious code, code mobility imposes some security constraints:

- host and mobile code bear separate identities, therefore the mobile code's origin must be authenticated;
- mobile code is exposed through the network, hence the host must verify the integrity of the mobile code it just received;
- the host does not generate the mobile code, but another party does: consequently, the actions it performs must be limited through access control and/or checked through semantic verification.

Code signing is a useful process to protect code when transferred, both from passive (*i.e.* eavesdropping and traffic analysis) and active attacks (*i.e.* message modification, deletion, forging); code is digitally signed by the provider in order to assure strong authentication and integrity to the consumer. Another approach is to use the the Java Reflection API to inspect the downloaded package, and also to test it in a controlled environment, in order to check if its behavior is the expected one.

The problem of protection from a malicious host is intrinsically more challenging; a malicious host can block the code functionality, steal information transferred with the code, such as partial results or communication keys, and modify the code itself, redistributing a corrupted version in the network.

To address these issues, SP2A currently exploits SAFE (Service Advisors For E-business) [2], a consistent QoS-aware reputation management system for service-oriented P2P networks which aims at improving the quality of service provider selection within purely decentralized mobile environments. SAFE relies on a decentralized voting scheme with a DHT-based approach to globally share information about advisors, *i.e.* peers with direct experience on specific service providers. Each service transaction is followed by an evaluation expressed in terms of a set of application specific QoS parameters, *e.g.* timeliness and accurateness, in order to improve the objectivity of the assessment. Service provider selection, albeit not deterministic, is based on available reputation which is ultimately dependent on advisors' evaluations. To avoid that malicious advisor peers, namely those providing deceptive advices, might collect a very high number of transactions, SAFE's DHT stops storing the updates if an excessively fast growth of the number of interactions with the same consumers is observed. Data integrity of information exchanged among peers is guaranteed by means of digital signatures.

2.4. Application Example: P2P Video Streaming

The SP2A platform enables general-purpose nodes to provide resources in a peer-to-peer fashion, following a PaaS model in which services may be diffused according to users' requests. Moreover, if the number of requests for a service decreases, some providers may decide not to offer it any more, and to download the code of other (more popular) services becoming providers for them.

An application that may take advantage of such an architecture is *P2P Video Streaming* [5]. The traditional client/server model for video streaming is not very scalable with respect to an increasing number of users, and suffers from limited bandwidth and excessive workload at the server side. The peer-to-peer paradigm offers a good solution to the problem of load balancing; indeed, each peer shares its resources (bandwidth for communication among nodes and disk storage for video contents) with other peers that are requesting a service.

Two main kinds of video streaming are available: *live* and *on-demand*. In a live session the video is provided simultaneously (and possibly in real-time) to all interested users, and playback is synchronized; conversely, a *Video-on-Demand (VOD)* service allows to play a video from any point and at any time, on the basis of users' requests. We argue that the case of live video streaming is particularly suitable to evaluate the adaptiveness of the SP2A platform, with service diffusion or regression depending on the global workload. Indeed, live streaming channels have higher probability that peaks of requests occur (*e.g.* in correspondence of popular events, such as sport or music events).

In the following we summarize a possible scenario for the P2P Live Video Streaming (see also fig.3), that is independent from the particular algorithms used to deliver the stream:

- in an initial setting, M peers provide the video streaming service and K peers may host a video stream buffer, thus becoming provider when necessary.
- a node starts a discovery for a certain video and selects one of the M provider peers; the selected node may accept the request only if it has not yet reached the limit of L served peers, otherwise the requestor has to choose another provider from the list.
- the M provider nodes communicate with each other, and are able to estimate the global workload on the system, all the time. When the system appears not to be able to guarantee the required QoS, they request the K available nodes to download the service and start providing it (such kind of interaction can be interpreted as "solicited pull").
- if a new provider peer is idle for a certain period of time, it may decide to remove the service and stop providing it. Otherwise, if requests for a service continue to increase it may also offer the service code for download, thus contributing to the service diffusion in the P2P network.

3. EXPERIMENTAL EVALUATION

We evaluated the proposed architecture on the PlanetLab testbed, a global distributed system that supports the development of new network services. Since the beginning of 2003, more than one thousand researchers at top academic institutions and industrial research labs have used PlanetLab to develop new technologies for distributed storage, network mapping, peer-to-peer systems, distributed hash tables, and query processing. The PlanetLab Consortium

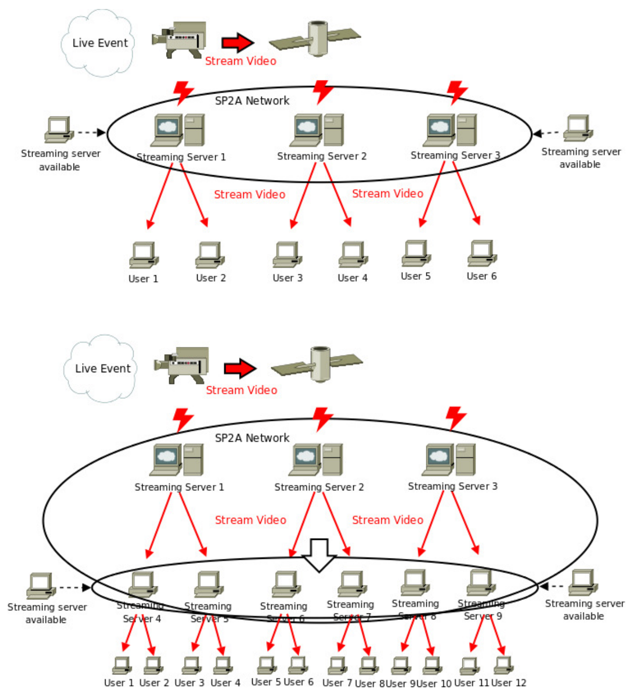

Figure 3. P2P Live Video Streaming Service in SP2A: in the Top, the Service Is Offered by a Limited Number of Nodes, Due to the Low Number of Requests; as the Number of Requests Increases (Bottom), the SP2A Architecture Enables Other Peers to Become Providers of the Service.

currently consists of 1056 nodes at 490 sites, and is managed by Princeton University, the University of California at Berkeley, and the University of Washington. Our University participates in PlanetLab Europe.

Given that all nodes are not always available in PlanetLab, the maximum number of 400 machines (at different sites) we used to host peers can be reasonably considered adequate for performance evaluation of the proposed system. In our experiments, we tested the worst case scenario of a service, initially provided by a few peers, that is simultaneously requested for download by a great number of users. Such a scenario represents a typical situation for the *P2P Live Video Streaming* example, for which the architecture must rapidly enable a certain number of nodes to become providers for the service. The most significant parameters we considered for evaluating the performance of our architecture are the time of service diffusion from a small to a large number of nodes, the discovery time and the workload distribution.

The test service was packaged as a *.jar* of 2MB size. Results presented refer to the average of 5 different tests we executed starting from the same initial conditions (with the penalization of variability on the state of nodes in PlanetLab network). All experiments have been repeated several times (up to ten), and their results have been averaged.

In a first experiment we considered the deployment of the service from a single provider to 100 downloaders, and we observed that the deployment speed is mainly influenced by the service discovery time (figure 4). The deployment time T_{deploy} is the sum of the discovery time $T_{discovery}$ (almost constant, see fig.4) and of the download time $T_{download}$ (linear, fig.5, resulting that the 100 nodes are served after about 700 seconds. The measured standard deviation shows that only a few nodes have a considerable delay. The workload distribution in fig.6 shows that an average of 20-25% of nodes that first obtain the service become provider, each one for 3-4 other requesting nodes.

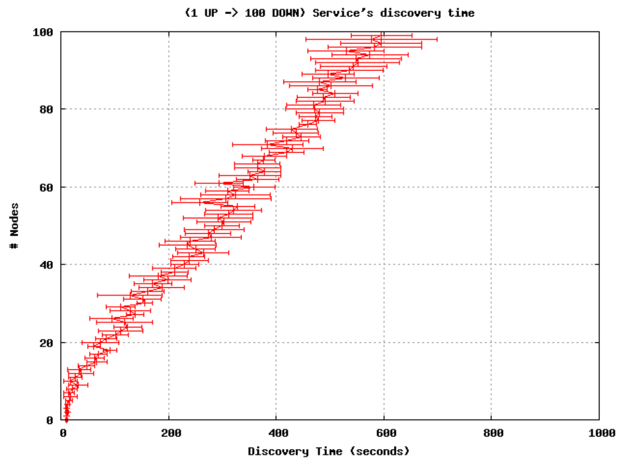

Figure 4. Discovery Time ($T_{discovery}$) for a Single Provider and 100 Downloader Peers.

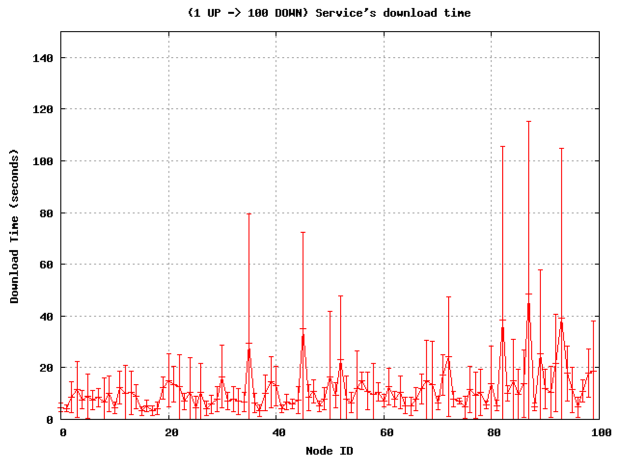

Figure 5. Download Time ($T_{download}$) for a Single Provider and 100 Downloader Peers.

A second experiment was setup by deploying the service

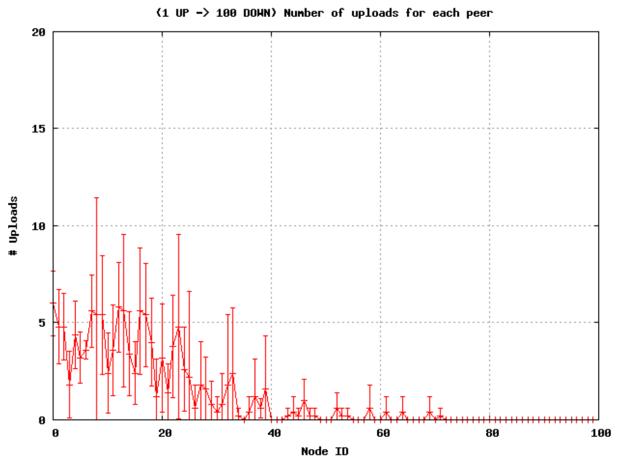

Figure 6. Service Upload Distribution, with 100 Nodes.

from a single uploader to 200 requesting nodes; as in the previous case $T_{discovery}$ influences the deployment of the service. T_{deploy} curve has a linear shape, but with two different slopes: the first, up to 50 served nodes, with a slow growth, because of the high workload of the system (a lot of nodes are requesting the service); in the second phase, a greater number of provider nodes is available, and the probability of finding a free peer to download from is higher. For space limitations we avoid to show these results, which are not very different to those obtained in the first experiment.

In a third setup we considered a single initial provider node and 400 downloaders; in the first phase, $T_{discovery}$ curve has an exponential shape, with up to the 80% of requesting nodes being served (fig.7). In the last part of graph, we observe a decrease in the slope of service discovery; this is probably caused by excessive workload on some of PlanetLab nodes, for which some peers are penalized in service discovery and download, with respect to the majority of the peers in the system. That was less evident for experiments with a lower number of requesting nodes. As for other cases, $T_{download}$ is almost constant (fig.8). In the workload distribution (fig.9) we notice that, as in the previous cases, an average of 20-25% of first served nodes immediately become provider for the requested service, and are able to serve 5 new requests each.

3. CONCLUSIONS

In this paper we presented a framework and a middleware that enable highly dynamic and adaptive clouds, characterized by peer providers and by services that can be replicated by means of code mobility mechanisms. We de-

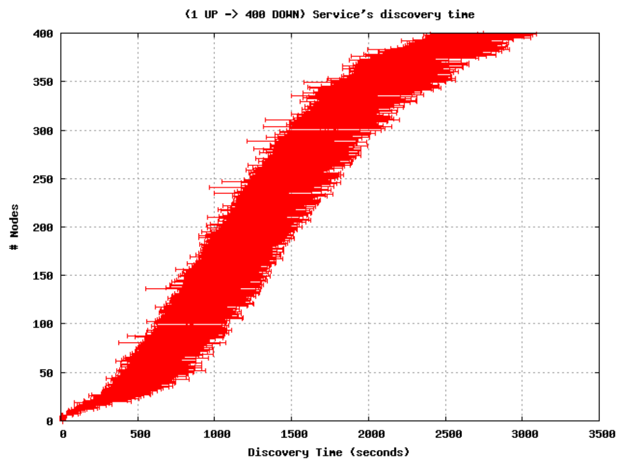

Figure 7. Discovery Time ($T_{discovery}$) for a Single Provider and 400 Downloader Peers.

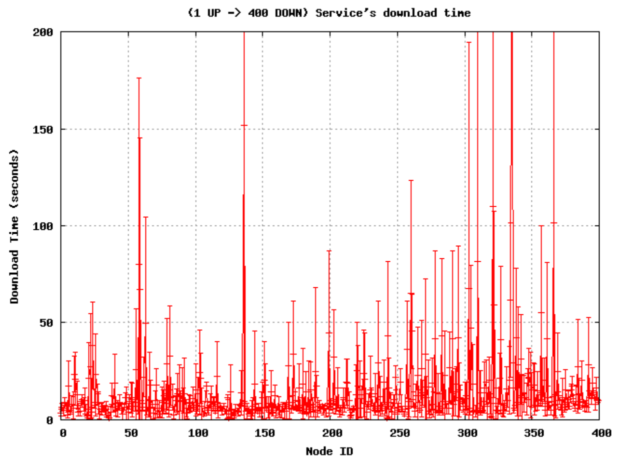

Figure 8. Download Time ($T_{download}$) for a Single Provider and 400 Downloader Peers.

scribed the Pull strategy for service migration, but also a Video streaming based example in which services are pushed to peers that are able to host them.

As future work we are going to evaluate the reliability and effectiveness of the security mechanisms we introduced. Using PlanetLab to run the prototypes allows us to setup realistic clouds, with heterogeneous hosts and dynamic load conditions.

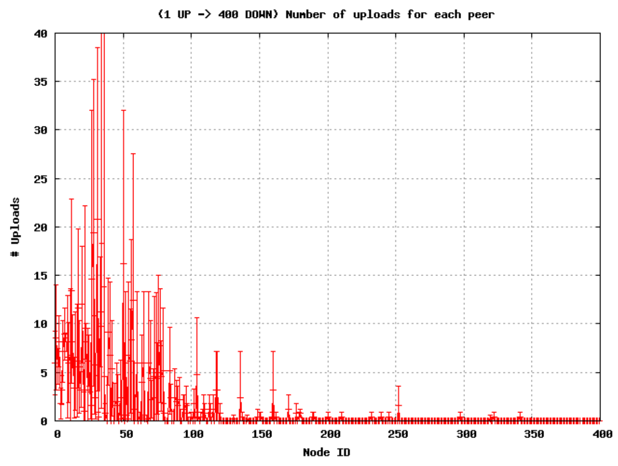

Figure 9. Service Upload Distribution, with 400 Nodes.

REFERENCES

[1] M. Amoretti, F. Zanichelli, G. Conte, "SP2A: a Service-orientd Framework for P2P-based Grids", Proc. of the 3rd international workshop on Middleware for Grid Computing, Grenoble, France, November 2005.

[2] M. Amoretti, M. C. Laghi, A. Carubelli, F. Zanichelli, G. Conte, "Reputation-based Service Selection in a Peer-to-Peer Mobile Environment", Proc. of the 9th IEEE Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM 2008), Newport Beach, CA, USA, June 2008.

[3] A. Carzaniga, G.P. Picco, G. Vigna, "Is Code Still Moving Around? Looking Back at a Decade of Code Mobility", Proc. of the International Conference on Software Engineering (ICSE), Minneapolis, MN, USA, May 2007.

[4] S. Distefano, A. Puliafito, M. Scarpa, "Volunteer Computing And Desktop Cloud: The Cloud@home Paradigm", Proc. of the 8th IEEE International Symposium on Network Computing and Applications (IEEE NCA09), MA, USA, July 2009.

[5] Y. Liu, Y. Guo, C. Liang, "A survey on peer-to-peer video streaming systems", *Peer-to-Peer Networking and Applications*, Vol.1, No.1, Springer, 2008.

[6] *PlanetLab project home page*, http://www.planet-lab.org

[7] B. Traversat, A. Arora, M. Abdelaziz, M. Duigou, C. Haywood, J.-C. Hugly, E. Poyoul, B. Yeager, "Project JXTA 2.0 Super-Peer Virtual Network", Technical Report, Sun Microsystems, 2003.

[8] L. Wang, J. Tao, M. Kunze, A.C. Castellanos, D. Kramer, W. Karl, "Scientific Cloud Computing: Early Definition and Experience", Proc. of the 10th IEEE International Conference on High Performance Computing and Communications HPCC'08, Dalian, China, September 2008.

[9] S Louriero, R Molva, Y Roudier, "Mobile Code Security", Proc. of ISYPAR 2000 (4me Ecole d'Informatique des Systmes Paralleles et Repartis), Code Mobile, Toulouse, France, February 2000