Learning about Distributed Systems

Francisco Javier de Hoyos
Advisor: Joan Manuel Marquès
javier@dehoyos.es, marquesp@uoc.edu
Master in Free Software Studies
Open University of Catalonia (UOC), Barcelona, Spain

ABSTRACT

This paper presents the "state of the art" about distributed systems and applications and it's focused on teaching about these systems. It presents different platforms where to run distributed applications and describes some development toolkits whose can be used to develop prototypes, practices and distributed applications. It also presents some existing distributed algorithms useful for class practices, and some tools to help managing distributed environments. Finally, the paper presents some teaching experiences with different approaches on how to teach about distributed systems.

Keywords: distributed systems, education, Internet

1. INTRODUCTION

Distributed systems are more and more important every day. Future Internet [1] [2] [3] initiatives propose to use distributed services deployed at planetary scale to improve scalability and availability, focusing on robustness and security. Many universities and educational institutions include studies about distributed systems and many researchers around the world are investigating about this matter.

When a person wants to learn about distributed systems there are many resources already available on Internet, but sometimes is complicated to know where to start. This paper presents and compares many resources related to distributed systems, from platforms where to run the applications to distributed algorithms, including some other aspects interesting to teach and learn about distributed systems. It presents a "state of the art" of all matters related to distributed systems, focused on engineering education.

PLATFORM section presents different platforms where to run distributed applications: simulators, emulators and real platforms. First categories for these platforms are presented, with their advantages and disadvantages, and later platforms are described and classified in to these categories.

If a person wants to test a new distributed algorithm, he can use an existing development toolkit to avoid programming all application details and focus on implementing his algorithm. DEVELOPMENT TOOLKITS section presents some of these toolkits with their programming languages and licenses.

When teaching or learning about distributed systems, it’s very interesting to study existing distributed algorithms. They can be also implemented by students in practices. The section DISTRIBUTED ALGORITHMS presents some existing distributed algorithms and their implementations, related with Distributed Hash Table (DHT) and Virtual Coordinates areas.

Another important thing when talking about distributed systems is tools to manage and monitoring distributed environment. TOOLS section describes some tools to deploy applications, execute and monitor applications in parallel, change configuration and manage a large amount of nodes in a distributed environment.

This paper is focused on teaching and learning about distributes systems, so we want to present too some teaching experiences about this matter. One proposes to use games to learn about complex distributed systems, another one proposes using cloud computing instead of a dedicated cluster, and the last proposal is to use PlanetLab nodes to interact with student practices.

2. PLATFORMS

There are different ways to execute and test a distributed system. It can be executed in a software simulated environment, like simulators and emulators, or real environments like testbeds, cloud computing, clusters and grids. **Table 1** describes these environments and shows their advantages and disadvantages.

Table 1. Categories of environments for executing distributed systems

	Category	Description	
Software environments	Simulators	This kind of applications allows testing distributed systems under a simulated and simplified network model. The application must be adapted to the simulation framework.	
		Advantages: • Test conditions can be repeated. • Large number of simulated nodes.	Disadvantages: • Application must be adapted to the framework. • Simplified network model. No real network conditions.
	Emulators	Like simulators, these systems allow testing distributed systems under a simulated and simplified network model, but in this case the application doesn´t need to be adapted.	
		Advantages: • Test conditions can be repeated. • Large number of emulated nodes. • Application don´t need to be adapted.	Disadvantages: • Simplified network model. No real network conditions.
	Testbeds	Testbed platforms allow executing distributed applications in a set of hosts geographically distributed and connected to a real network. They are shared platforms, and uses virtualization to create different nodes on each host. In this way each application can have its own dedicated nodes and share hosts with other applications.	

Real environments		Advantages: • It uses a real network. • Application don´t need to be adapted. • Dedicated and fully configurable nodes on shared hosts. • Any kind of distributed application can be deployed.	Disadvantages: • Test conditions can´t be repeated, it´s a real network. • It´s a shared platform. Applications can impact on other applications running in the same hosts.
	Cloud Computing	Cloud computing platforms provide on-demand and dynamically scalable resources. They often use virtualization to share resources and can offer SLAs (service level agreements) to meet client quality of service (QoS) requirements. Infrastructure details are hidden and the “Cloud” is shown to the user as a single point of access to the infrastructure.	
		Advantages: • It uses a real network. • Application don´t need to be adapted. • Dynamically scalable. This allows adapt platform to usage peaks. • Pay per usage model. • User can sign SLAs to ensure quality requirements	Disadvantages: • Test conditions can´t be repeated, it´s a real network. • Some platforms allow fully configuration of nodes, but other ones offer limited configuration capabilities.
	Clusters	Clusters are a group of linked computers working together closely so that in many aspects they form a single computer. They are often, but not always, homogeneous and are connected to each other through fast local area networks. They are deployed to improve performance and/or availability over a single computer. Cluster platforms are designed to execute one distributed application at the same time.	
		Advantages: • It uses a real network. • Fast communication among nodes (often). • Fully dedicated platform. • Deployed “middleware” can help to develop distributed applications.	Disadvantages: • Test conditions can´t be repeated, it´s a real network. • Only applications designed to work with the deployed “middleware” can be executed. • It´s not a shared platform. Often only one distributed application can be executed at the same time.
	Grid Computing	Grids are usually computer clusters, but more focused on distributed computing. Grid computing is optimized for workloads which consist of many independent jobs or packets of work, which do not have to share data between the jobs during the computation process. Often, grids will incorporate heterogeneous collections of computers, possibly distributed geographically, sometimes administered by unrelated organizations.	
		Advantages: • It uses a real network. • Fully dedicated platform. • Deployed “middleware” can help to develop grid computing applications.	Disadvantages: • Test conditions can´t be repeated, it´s a real network. • Only applications designed for deployed grid computing “middleware” can be executed. • It´s not a shared platform. Often only one distributed application can be executed at the same time.
	Volunteer Computing	Volunteer computing is a type of network in which participant nodes donate their computing resources when the computer is idle. Some client software consists of a single program that combine scientific computation and the distributed computing infrastructure, but this approach is fixed and can´t be updated. Some middleware systems have been developed to provide a distributed computing infrastructure independently of the scientific computation and other middleware has been designed to create a testbed where to test distributed applications.	
		Advantages: • It uses a real network. • It uses idle computing resources. • Heterogeneous systems. • Hardware and maintenance cost for nodes are managed by computer owners (volunteers).	Disadvantages: • Test conditions can´t be repeated, it´s a real network. • Only applications designed to work with the deployed “middleware” can be executed. • Nodes are not owned. Availability can´t be ensured. • Computation could be repeated in different nodes to ensure precision.

Simulators and ***emulators*** allow us to test new network protocols, services or distributed applications simulating hundreds or thousands of nodes. Two great advantages are the possibility of simulate a large number of nodes with a single computer and the possibility of repeat the test under the exactly the same network conditions. On the other hand, they use a simplified network model so some issues found in a real network can't be modeled. In the case of simulators the application must be adapted to the simulation framework, but emulators can execute real distributed applications.

NS-2 [4] is an event based network simulator used for networking research. It provides support for simulation of TCP, routing, and multicast protocols over wired and wireless networks. It has a semi-free license, which provides source code and is free for non-commercial use. It has been written in C++ and Object TCL (OTcl).

OPNET [5] [6] is a commercial network simulator, but it´s provided free to universities for academic research and teaching. It allows creating a detailed network topology, making a simulation of network traffic generated by the application and analyzing the network response.

ProtoPeer [7] [8] is a development framework for Java described at section 3. It includes a simulator, and the same distributed application can be executed in a real platform or in a simulated environment, only changing the configuration file.

ModelNet [9] [10] is an emulation environment developed at University of California San Diego. Distributed application instances are executed in virtual nodes, and these nodes are interconnected using a network core element, which introduces a configurable set of network characteristics into the communication. Virtual nodes and network core are distributed among one or more computers in a local-area network.

Testbeds are environments designed for researching about new distributed applications and services. A testbed consists on a set of hosts connected to a real network and often geographically distributed. Different applications and services can share the platform, so it uses virtualization to show to the application a dedicated environment while working in a shared platform. The user must select in which nodes the experiment must be executed. Testbeds like Emulab are able to simulate or emulate the network, use real network connections or integrate all of them in the same experiment. Other platforms like Planetlab are designed mainly to test distributed services and applications at planetary scale, so their hosts are widely distributed across a lot of countries.

Emulab [11] [12] is a testbed software designed at the University of Utah. It allows different network configurations (simulated, emulated or real connections) and uses virtual nodes where the user can install and configure necessary software and runs the experiments. It’s possible to install different operating systems on each node, including FreeBSD, Windows XP and some Linux distributions. The most important Emulab platform is installed at University of Utah, but there are more Emulab platforms around the world.

PlanetLab [13] [14] [15] is a testbed composed of computers distributed over the globe, more of them hosted by research institutions. All PlanetLab computers run a common software package called MyPLC that includes a Linux-based operating system supporting virtualization and tools to monitor node health and system activity and to manage user accounts. A research group can request a PlanetLab slice to experiment with planetary-scale services, distributed applications and network protocols.

SatelliteLab [16] is a network testbed that adds heterogeneity to other testbed platforms like PlanetLab or Emulab. SatelliteLab architecture has two types of nodes: planets and satellites. Planets are nodes in charge of run the core experiment application, and they must be hosted in other testbed platform like PlanetLab. Satellites are lightweight nodes connected to the planets which only forward network traffic. SatelliteLab project provides the software for that satellite nodes and sample software for the network experiment application. SatelliteLab platform contains only the satellite nodes which can be integrated into experiments on other testbeds to add more heterogeneity.

Cloud computing [17] platforms provide on-demand and dynamically scalable resources. Infrastructure details are hidden to the user and they follow a pay-per-use model. It’s easy to scale the system when demand increases, or in the case of huge distributed systems experiments it’s possible to create nodes only to run the experiment. Commercial cloud computing platforms offer to the client SLAs (service level agreements) to meet client quality of service (QoS) requirements. Platforms as Google App Engine follow the model “Platform as a Service (PaaS)”, where they offer a development environment for applications with APIs, libraries and services, and the applications are executed in a sandbox. Other environments like Amazon EC2 follow the model “Infrastructure as a Service (IaaS)”, a more flexible approach that uses virtualization to allow the user fully control over the distributed system.

Amazon Elastic Cloud Computing (EC2) [18] [19] is part of Amazon Web Services cloud computing platform. Users can prepare their own Amazon Machine Image (AMI) files or select available generic AMI files containing the desired software, and they can run or stop virtual machine instances using these AMI files. Users have full control over installed software, and different operating systems are available, including Unix, Linux distributions and Windows Server. Users have control over geographical location of instances. Users pay for the time servers are running, so it's easy to adapt production platforms to peaks of demand. There are other Amazon Web Services available, like Amazon Simple Storage Service (S3) and Amazon Relational Database Service (RDS).

Google App Engine [20] [21] is a cloud computing platform for developing and hosting web applications. It's free up to a certain level of used resources. Currently supported languages are Python and Java. Developers don't need to install or configure any operating system, they only need to develop their application in Java or Python, and Google platform handles deploying code to a cluster and launching application instances as necessary. This approach simplifies deploying applications, but only applications developed for this platform can be executed. Applications are executed inside a sandbox and many restrictions apply to them. The platform offers a Datastore service to store information in a non-relational database.

Windows Azure [22] [23] [24] is the Microsoft approach to cloud computing. It provides a cloud operating system and a set of services to develop and host applications. Windows Azure is a cloud layer on top of a cluster of computers with Windows Server 2008 and a customized Hyper-V to provide virtualization of services. The platform currently supports .NET applications, ASP.NET, PHP, Java and Ruby.

Science Clouds [25] are a group of small clouds voluntarily made available by some institutions. It's based on **Nimbus** [26] software, an open source toolkit to create an Infrastructure-as-a-Service (IaaS) cloud. The objectives of this platform are two: to allow scientific people experiment with IaaS cloud software, and to provide projects developing software for these clouds with user comments, requirements and suggestions.

A ***cluster*** is a group of computers working together closely as if they were a single computer. It's deployed to improve performance and/or availability over a single computer. Computers in a cluster often are homogeneous and connected through a fast local area network. Clusters run an application at the same time, but some cluster applications can distribute different jobs over cluster nodes. ***Grid computing*** platforms [27] [28] are computer clusters optimized for distributed computing, in which the computation process can be divided into independent jobs which do not share data among them. Many organizations and institutions have private clusters and grids.

TeraGrid [29] [30] is a grid computing platform which integrate high-end computational facilities from 11 organizations and institutions through ultra-fast network links. TeraGrid unifies software, policies and procedures across these facilities to create a unified system to support advanced science and engineering research. TeraGrid uses middleware like Globus Toolkit and Condor-G. TeraGrid platform is coordinated through the University of Chicago.

Grid'5000 [31] is a scientific experimental platform to research about large scale parallel and distributed computing systems. It's initially composed of resources located at 9 sites in France connected with a dedicated network link of 10 Gb/s, but now it includes also a site at Porto Alegre, Brazil, and soon it will include a site at Luxemburg. It is designed for scientific research in the grid domain.

Volunteer computing is a type of network in which participant nodes donate their computing resources when the computer is idle (called also CPU scavenging). Some volunteer computing platforms are dedicated to computing applications and are a type of grids, like Boinc and Condor, and other platforms create a testbed where to test distributed applications, like Seattle which is specialized on network experiments in a heterogeneous platform, including portable devices. Condor software can manage together volunteer computing nodes and dedicated grids and computers, in a mixed environment, and it can be integrated with Globus Toolkit, a grid middleware software.

BOINC (Berkeley Open Infrastructure for Network Computing) [32] [33] is an open source platform for public-resource distributed computing. It has been designed to run scientific projects using resources donated by volunteers. Computer owners can participate in multiple BOINC projects and can choose these projects. BOINC features include redundant computing, support for user-configurable application graphics and a participant credit accounting (a numeric measure of how much computation participants have contributed). BOINC applications are developed in C, C++ or Fortran and use the BOINC programming API [34], a set of C++ functions. BOINC is based at University of California, Berkeley.

Seattle [35] [36] is an educational platform which depends on resources donated by users of the software. Applications for Seattle platform are developed using a subset of Python language and a programming API, called Repy, providing some communications and synchronization functionality. This API can be used from Seattle programs (restricted programs designed to run inside a sandbox in Seattle nodes) and Seattle applications (unrestricted applications which must be installed by the end-user). The same program runs across a wide range of operating systems and architectures, including Windows, Mac OSX, Linux, FreeBSD and portable devices. Seattle is based at University of Washington.

Condor [37] [38] is a specialized batch system for managing compute-intensive jobs. Users submit their compute jobs to Condor, and it puts jobs in a queue, runs them and sends the results to the user. Condor can manage a cluster of dedicated computers, idle time from desktop workstations or resources managed by Globus (using Condor-G). Condor supports several runtime environments, called universes, designed to run different kind of applications. C applications relinked with condor_compile can be executed in Standard universe and they can benefit from system calls, transparent checkpoint and migration of jobs if the computer must be returned to its owner. Non-interactive programs, Java applications or parallel MPI jobs can be executed in other universes. Condor is based at University of Wisconsin.

Table 2 includes all platforms described in this section and some characteristics about them to allow compare the platforms.

Table 2. Platforms

	Category							Language support	Operating System	License	Number of computers in platform
	Simulator	Emulator	Testbed	Cloud Computing	Cluster	Grid	Volunteer Computing				
NS-2	●							C++, OTcl	Unix / Linux / Windows	Free for non-commercial use	Not apply
OPNET	●							C/C++	Unix / Linux / Windows	Commercial. Free for universities	Not apply
ProtoPeer	●							Java	Java-enabled OS	GPL	Not apply
ModelNet		●						Any application	FreeBSD / Linux	GPL v2	Not apply
Emulab	●	●	●					Any application	FreeBSD / Linux/ Windows XP	AGPL v3	> 350
PlanetLab			●					Any application	CentOS and Fedora	BSD	> 1,000
SatelliteLab			●					Network experiments.	Windows, Linux, MacOS, Java	Copyleft	
Amazon EC2				●				Any application	Unix / Linux / Windows Server	Private	> 40,000
Windows Azure				●				.NET, PHP, Java, Ruby	Windows Azure	Private	> 3,500,000
Google App Engine				●				Python, Java		Private	
Science Clouds				●				Any	Linux	Apache v2	16
TeraGrid					●	●			Linux	Private	
Grid'5000					●	●		REST (HTTP)	Linux	Private	> 5,000
Boinc					●	●	●	C, C++, Fortran	Linux, Windows, MacOS	LGPL v3	> 500,000
Seattle			●				●	Python	Windows / Unix / Linux / MacOS / portable devices	MIT	> 1,000
Condor					●	●	●	C, Java, non-interactive apps	Windows / Unix / Linux / MacOS	Apache v2	> 300,000

3. DEVELOPMENT TOOLKITS

In this section we want to introduce some development toolkits designed to simplify development of distributed applications. Some toolkits are tied to a specific platform or middleware, like BOINC APIs or the Globus Toolkit. Other ones like Appia, Mace or ProtoPeer can be deployed in a testbed like PlanetLab or a private cluster. Some of them are designed for experimental research or educational applications, and other ones are used in production applications.

Appia [39] [40] [41] is an open source communication toolkit implemented in Java. It provides a core to compose protocols and a set of protocols that provide group communications, atomic broadcasting and more protocols.

Cactus [42] [40] [43] is a project to develop a design and implementation framework for supporting dynamic fine-grain Quality of Service (QoS) attributes in distributed systems. The task support layer provides QoS guarantees for task execution, and the distribution support layer provides QoS guarantees for network communications. Prototype implementations are available in C, C++ and Java.

Mace [44] [45] is a development framework designed to develop high performance distributed applications. It consists of an extension of C++ language to describe a distributed application behavior, a compiler from Mace to C++ and some tools. Mace provides functionality for network management, event handling, concurrency and messaging layers. Programmers can use their C++ tools to compile and debug Mace applications. Many distributed algorithms and applications have been already implemented in Mace to show sample Mace applications.

P2 [46] [47] is a facility that allows constructing declarative overlay networks. The application can describe an overlay network using a high-level declarative language called OverLog, and it can call P2 runtime library to construct and maintain that overlay network. A Chord structured overlay can be done with only 47 rules.

ProtoPeer [7] [8] is a toolkit developed in Java for rapid distributed systems prototyping. It provides functionality for network management, message passing and queuing, timer operations, measurement logs and neighbor overlay management. The same application can be executed in a simulated environment or deployed in a real distributed platform like PlanetLab with only changing the configuration file.

Seda (Staged Event-Driven Architecture) [48] [49] is a distributed application architecture that decomposes a complex event-driven application into a set of stages connected by queues. It's implemented in Java and uses the NBIO [50] package to provide nonblocking I/O support. Some open source applications use Seda architecture and NBIO, like OceanStore [81].

Splay [51] [52] is an integrated system designed to simplify developing, deployment and testing of large-scale distributed applications. The user can write his distributed application or algorithm in a language based on LUA [53] and use provided libraries designed to facilitate the development of distributed applications and overlays. Splay can deploy, execute and monitor the application in a large number of hosts on PlanetLab, Emulab, ModelNet, networks of idle computers, personal computers or a mixed environment. Applications and libraries are platform-independent and are executed in a sandbox. Splay provides a configurable churn manager to reproduce the dynamics of a real distributed system with nodes continuously joining and leaving.

Some grid computing platforms use middleware software (***grid middleware***) which includes libraries that distributed applications must/can use to simplify communications and other common functionality. Applications developed with these APIs are focused on distributed computing and are designed to be deployed on platforms running this grid middleware software.

Boinc and **Seattle** are two volunteer computing platforms described at section 2. They have programming libraries to be used by applications executed in these platforms.

gLite [54] [55] is a lightweight grid middleware software used by the CERN (European Organization for Nuclear Research) LHC (Large Hadron Collider) experiments and other scientific organizations. It's used by more than 15,000 researchers at more than 250 sites around the world. It uses web services approach for some services, but currently it's not following OGSA and WSRF standards. It uses R-GMA [56] for information and monitoring services.

Globus Toolkit (GT) [57] [58] [59] is an open source toolkit designed to build Grid systems and applications. GT architecture is based on the standards Open Grid Services Architecture (OGSA) [60] developed in the Open Grid Forum (OGF) [61] and the Web Services Resource Framework (WSRF) [62] developed by Oasis [63]. It uses stateful webservices, and client libraries to accessing these web services from languages like Java, C or Python (Java WS Core, C WS Core and Python WS Core). This Grid middleware software is used in many Grid platforms around the world (like TeraGrid).

Unicore (Uniform Interface to Computing Resources) [64] is an open source Grid middleware software. It is also based on OGSA and WSRF standards like Globus Toolkit, but they have different security model, basic services and interfaces. Unicore includes server and client software to create a grid platform.

Table 3 compares development toolkits described in this section.

Table 3. Development Toolkits

	Language	Operating System	Platform	License
Appia	Java	Java-enabled OS	Any	Apache v2
Boinc	C, C++, Fortran	Linux, Windows, MacOS	Boinc	LGPL v3
Cactus	C, C++ and Java	Linux, Unix, Windows	Any	BSD-style
gLite	C, C++, Java, Python	Linux	gLite middleware	Apache v2
Globus Toolkit	Java, C, Python	Linux, Unix, Windows	GT middleware	Apache-style
Mace	C++	Linux, MacOS, Windows	Any	BSD-style
P2	C++	Linux	Any	BSD-style
ProtoPeer	Java	Java-enabled OS	Any	GPL
Seattle	Python	Windows / Unix / Linux / MacOS / portable devices	Seattle	MIT
Seda	Java	Java-enabled OS	Any	BSD
Splay	Lua	Linux, Unix	Any	GPL v3
Unicore	Java	Java-enabled OS	Unicore middleware	BSD

4. DISTRIBUTED ALGORITHMS AND APPLICATIONS

If we want to learn or teach about distributed systems would be interesting to study already existing algorithms and implemented distributed applications. In this section we present some algorithms related with Distributed Hash Table concept (store key/value pairs among distributed hosts) and Network Coordinates (use host virtual coordinates to predict network latency between nodes). These algorithms presents common issues related to distributed applications and different approaches to manage them. They could be a good starting point to study and understood distributed systems. **Table 4** shows these proposed algorithms.

Table 4. Proposed Distributed Algorithms

Type	Description	Algorithms
Distributed Hash Table (DHT)	A Distributed Hash Table (DHT) algorithm allows a group of distributed hosts manage a collection of keys and data values mapped to those keys, without a fixed hierarchy. DHT algorithms must manage "churn", that is, hosts being added or removed to the DHT system.	CAN Chord Pastry Tapestry Bamboo Kademlia
Network Coordinates (NC)	Network Coordinates algorithms are used to predict Internet latency between two nodes based in their network coordinates. These network coordinates are calculated using a few messages to neighbor nodes. These algorithms are used in distributed applications for optimized message routing.	Vivaldi Pharos

The first Distributed Hash Table (DHT) algorithms available were Content Addressable Network (CAN), Chord, Pastry and Tapestry. Bamboo is another DHT algorithm based on Pastry. Common characteristics to DHT systems are decentralization (there is not a central coordination), scalability (system must operate efficiently even with thousands of nodes) and fault tolerance (nodes can continuously be failing, joining or leaving, the "churn" of the DHT). A good paper describing DHT algorithms is [65].

Content Addressable Network (CAN) [66] uses a virtual multi-dimensional Cartesian coordinate space which is dynamically partitioned among all the hosts in the system. To store key-value pairs, the key is mapped to a coordinate using a uniform hash function, and the value is stored in the node in charge of that coordinate zone.

Chord [67] [68] uses a SHA-1 hash function to get identifiers for nodes and keys. Identifiers are ordered on a circle called Chord ring. Keys and hosts are distributed among the Chord ring. Chord is decentralized and symmetric and manages node failures and joins. It can find the node responsible for a key using log(N), where N is the number of nodes. Some applications using Chord are:

- DHash [68] is a DHT implementation using Chord which provides a simple put and get interface to store and retrieve data.
- Cooperative File System (CFS) [69] is a distributed read-only storage system designed to be efficient and robust. It uses DHost for block storage, Chord to locate blocks and the SFS file system toolkit.
- Chord-based DNS [70] provides a DNS-like distributed lookup service, storing host names as keys and IP addresses and other host information as values.

Pastry [71] [72] uses Plaxton-like [73] prefix routing to build an overlay routing network. When a node joins the network receives a randomly generated identifier (NodeID) of 128 bits. Pastry routes messages to the host whose NodeID is numerically closest to the key. Each node manages a routing table and uses network metrics to optimize it. This reduces the cost of routing packets among nodes. FreePastry [72] and SimPastry [74] are two available implementations of Pastry. The following distributed applications use Pastry:

- Scribe [75] is a distributed publish/subscribe multicast messaging system. It uses Pastry to store multicast trees to deliver messages to the proper group. It supports large groups.
- SplitStream [76] is a high-bandwidth content distribution utility based on Pastry and Scribe. It distributes the forwarding load among the participant nodes.
- Squirrel [77] is a decentralized and distributed web cache based on Pastry where each peer request items when browsing the web, store items and serve items to other peers.
- PAST [78] is a large scale distributed persistent storage application based on Pastry.

Tapestry [79] is another DHT algorithm similar to Pastry. It also uses Plaxton-like [73] prefix routing. The difference between Pastry and Tapestry is how they handle network locality and data replication. Chimera [80] is a C implementation of Tapestry. Some applications based on Tapestry:

- OceanStore [81] is a high available storage application deployed on PlanetLab.
- Bayeux [82] is an efficient application level multicast system.
- SpamWatch [83] is a distributed and decentralized spam filtering application.

Bamboo [84] [85] is a open source DHT implementation based on Pastry, using the same geometry of neighbor links but different joining or neighbor management algorithms. Its algorithms are more incremental than Pastry algorithms, resulting on better efficiency in environments with large membership changes, continuous churn in membership or reduced network bandwidth. Bamboo DHT was deployed on PlanetLab and available as a free service named OpenDHT.

Kademlia [86] [87] is a DHT algorithm used in Kad Network (eMule), BitTorrrent, Overnet Network and Gnutella. A node chooses a unique NodeId, and it uses exclusive or (XOR) metrics on NodeIds to calculate distance between nodes. The node manages as many lists as number of bits on NodeId to store all contacted nodes, arranged by distance. In this way, the first list contains all nodes whose first bit in NodeId is different; the second list contains all nodes with the same first bit in NodeId and a different second bit, and so on. These lists are called k-buckets because they contain a maximum of k nodes per list (k is a constant in the algorithm, usually 20). Khashmir DHT [88] is a library implementing Kademlia used by BitTorrent application.

A distributed application can use Network Coordinates algorithms to calculate virtual coordinates for the nodes and predict with more or less accuracy the latency among nodes. This can be useful to optimize routing protocols and in this way send messages through the short path or locate nearest nodes. Some algorithms use landmark nodes as references to calculate node coordinates (like IDMaps [89], GNP [90] or NPS [91]), but in this paper we propose two distributed algorithms whose don't use landmarks, Vivaldi and Pharos.

Vivaldi [92] [93] is a distributed network coordinates algorithm which can calculate virtual coordinates for a node contacting with a few other nodes. It doesn't need dedicated network infrastructure. It's used in Azureus/Vuze [94] BitTorrent client.

Pharos [39.1] is another network coordinates algorithm. It uses Vivaldi algorithm to calculate two sets of coordinates, one for long link predictions (global NC) and other for short link predictions (local NC). This improves Internet latency prediction compared to original Vivaldi algorithm.

Table 5 shows a comparison among implementations of the proposed distributed algorithms.

Table 5. Comparison among implementations of proposed distributed algorithms

Type	Algorithm	Implementation	Language	Operating System	License
Distributed Hash Table (DHT)	CAN (Content Addressable Network)				
	Chord	Chord/DHash	C++	Unix, Linux	MIT/X11-style
	Pastry	FreePastry	Java	Linux, Windows	BSD-like
		SimPastry	C#	Windows	Free for non-commercial use
	Tapestry	Tapestry	Java	Linux	BSD
		Chimera	C	Linux, Windows	GPLv2
	Bamboo	Bamboo DHT	Java	Linux, Windows	BSD
	Kademlia	Khashmir DHT	Python	Linux, Windows	MIT
Network Coordinates	Vivaldi	Azureus/Vuze	Java	Linux, Windows	GPL
	Pharos				

5. TOOLS

This section presents tools used in distributed systems with large number of nodes. In these platforms the user must prepare all the nodes to execute the distributed application, and tools able to execute tasks in a set of hosts at the same time can be very useful.

Table 6 shows the categories used to classify the tools. They are related to necessary steps when executing distributed applications.

Table 6. Categories for tools used in distributed systems

Deployment	This category applies to systems designed to simplify deployment of distributed applications among all nodes where they will be executed.
Discovery	In some distributed platforms is necessary select the nodes where the applications should be executed. Applications have some resource requirements and nodes offer some available resources. This category of tools allows selecting nodes whose meet application requirements and can be used to execute the distributed application.
Execute	This category of tools applies to applications that simplify the simultaneous execution of the distributed application in all the nodes.
Configuration	This other category applies to tools designed to simplify the configuration of all nodes where the application will be executed or to manage the configuration of the distributed application or service.
Monitoring	This category applies to tools whose facilitate monitoring of the distributed application in all the nodes where it is running.

Cfengine [96] [97] is a tool based in a new configuration language designed to create powerful administration scripts. These scripts allow testing and configuring network interfaces, editing text files, mounting NFS file systems, executing other scripts, managing symbolic links, files, permissions and ownerships, and other checks.

CoDeeN [98] is a Content Distribution Network (CDN) built on top of PlanetLab testbed. It has been developed at Princenton University. There are other interesting tools related to the CoDeeN project:

- **CoDeploy** [99] distribute content from one source to many PlanetLab nodes (hundreds). It uses the CoDeeN content distribution network and can replicate a folder from the source to other folder in a list of PlanetLab nodes. CoDeploy calculates the checksum for the files to compare with the files in the destination folder and copy only the different ones. It also splits large files to improve efficiency and cacheability.

- **CoMon** [100] [101] is a web-based monitoring tool designed specifically to monitor activity on PlanetLab nodes. It's extensible and provides information to both administrators and users. CoMon allows monitoring resources consumed by an experiment, identifying problematic nodes or to know what experiments are impacting on other ones.

- **CoTop** [102] is a monitoring tool like the top utility, but showing information about slices instead of process. It uses the SliceStat sensor to gather information from the host.

- **CoVisualize** [103] shows usage of PlanetLab. It uses CoMon project to get some metrics every five minutes and shows different graphics about the usage of PlanetLab.

- **SliceStat** [104] provides information about resources (CPU, memory, bandwidth, number of processes, etc.) used by slices on each PlanetLab node.

Distributed Service Management Toolkit (DSMT) [105] can select PlanetLab nodes, deploy the application to these nodes, and finally execute and monitor the application. It can restart nodes or reallocate nodes as necessary.

Ganglia [106] [107] is a scalable distributed monitoring system designed for clusters and grids, and currently it's working on more than 500 clusters around the world. It has been deployed on PlanetLab successfully, but it was designed for systems in a local area network and presents some performance issues. It uses RRDtool [108] to store and visualize the information.

Gexec [109] is a remote execution tool for parallel and distributed jobs. It uses RSA authentication and provides transparent forwarding of stdin, stdout, stderr and signals to and from remote processes. It internally creates a communication tree between gexec daemons to distribute the workload and the control information. Gexec scales to more than 1000 nodes.

Gush (GENI User Shell) [110] is an extensible execution management system for GENI [111]. It can be used to deploy, run, monitor and debug distributed applications on PlanetLab, Emulab or other resources integrated in GENI. Users can describe their experiments in a XML document and Gush can prepare the environment and execute the distributed application. It supports Nebula [112] as GUI interface. Gush is based on Plush. It can be integrated with Sword for discovery or with Stark for package installation.

Management Overlay Networks (MON) [113] is a monitoring tool which allows dynamically querying application status using a SQL-like language and some commands. It constructs a monitoring overlay network on demand, collects the information and discards the overlay network. It's a complementary approach to continuous monitoring tools. It has a web interface to query PlanetLab status and integrates with CoMon daemons.

Nixes [114] is a set of bash scripts to install RPM packages and execute commands on PlanetLab. All the scripts work in parallel with a list of nodes.

PlanetLab Application Manager [115] tool helps to deploy, monitor and run applications on a PlanetLab slice. It uses a web user interface. The server side part requires a web server with PHP and a PostgreSQL database, and the client side part is composed of bash scripts. Client side scripts must be configured.

PlanetLab Experiment Manager (PIMan) [116] is a tool designed to simplify the deployment, execution and monitoring of a PlanetLab experiment. It has a GUI to select nodes for the slice and choose nodes where to run the experiment, deploy files, execute commands in parallel on every node and monitoring the progress of the experiment. It also presents a scripting interface to allow be used by scripts written in Perl, Python, Ruby, Java and C/C++.

PLDeploy [117] is a set of Perl scripts to create slices, add or remove nodes, deploy and execute services on PlanetLab nodes.

Plush [118] [119] is a generic platform to control distributed applications that can be configured using XML files. It can manage the application deployment, execution, monitoring and synchronization. In the case of error, Plush can execute previously configured actions. Plush consists on a controller running on the user computer and a lightweight client running on each node to be controlled. It can be used on PlanetLab, a Grid or a cluster. Nebula [112] is a GUI interface for Plush.

pShell [120] is a tool with an interface like a Linux shell that provides some commands to interact with a PlanetLab slice. These commands include add or remove nodes from the slice, list nodes in slice and their status, upload or download files from nodes to local machine and execute commands in slice nodes.

Pssh [121] package provides parallel versions of the openssh tools. It includes the parallel versions of ssh, scp, rsync, nuke and slurp: pssh, pscp, prsync, pnuke and pslurp.

SmartFrog [122] [123] (Smart Framework for Object Groups) is a framework to describe a distributed system and manage deploy, execution and monitoring of that distributed system. It defines a language based on prototype supporting templates, used to describe all the elements of the distributed system, their software and configuration, how they are connected and how they must be started and stopped. The framework configures and starts the elements and monitors their execution. If an element fails it can start a configured recovering action or shut down the system in a controlled way.

Splay [51] [52] is described at section 3. It's an integrated system designed to simplify developing, deployment and testing of large-scale distributed applications.

Stork [124] is tool to install Linux packages (tarball and RPM) on PlanetLab nodes. Stork Slice Manager is a GUI for Stork to simplify the package management on PlanetLab slices. It shares downloads between nodes and uses public key encryption to provide package security.

Sword [125] [126] [127] is a decentralized resource discovery service for PlanetLab. Sword uses CoTop to collect measurements per-node (CPU load, free memory or free disk space) and inter-node (latency between nodes or bandwidth) for nodes in PlanetLab. Users send queries in XML format requesting nodes matching the query criteria. The query can include per-node and inter-node attributes.

Vxargs [128] provides the parallel version of any arbitrary command, including ssh, rsync, scp, wget, curl and whatever. It redirects stdout and stderr of each individual job to files.

Table 7 shows all these tools, their categories, operating system, language and license. It also shows if the tool support any platform or only some of them (like PlanetLab).

Table 7. Tools

	Category					Platform support	Operating System	Source Language	License
	Deployment	Discovery	Execute	Configuration	Monitoring				
Cfengine				●		Any	Linux, Windows	C	GPL v3, Commercial Open Source License
CoDeeN						PlanetLab	Linux		
CoDeploy	●					PlanetLab	Linux		
CoMon					●	PlanetLab	Linux		
CoTop					●	PlanetLab	Linux		
CoVisualize					●	PlanetLab	Linux		

Distributed Service Management Toolkit	●	●	●		●	PlanetLab	Linux	Java	BSD-like
Ganglia					●	Any	Linux, Unix, Windows	PHP, C, Perl, Python	BSD
Gexec			●			Any	Linux, Unix	C	BSD
Gush – GENI User Shell	●		●	●	●	GENI, Emulab, PlanetLab	Linux, Unix	C++, Perl	
Management Overlay Networks (MON)					●	PlanetLab	Linux	C++	BSD-like
Nixes	●		●	●	●	PlanetLab	Linux	Bash scripts	
PlanetLab Application Manager	●		●		●	PlanetLab	Linux	PHP, Bash scripts	BSD-like
PlanetLab Experiment Manager - PIMan	●	●	●		●	PlanetLab	Linux, MacOS, Windows	Java	
PLDeploy	●		●			PlanetLab	Linux	Perl	BSD-like
Plush	●		●		●	Any	Linux, Unix, MacOS	C++, Perl	Academic Free License (AFL)
pShell	●		●	●		PlanetLab	Linux	Python	
Pssh	●		●			Any	Linux	Python	BSD-like
SliceStat					●	PlanetLab	Linux		
SmartFrog	●		●	●	●	Any	Linux, Windows Unix, MacOS	Java	LGPL
Splay	●	●	●	●	●	Any	Linux, Unix	C, LUA	GPL v3
Stork	●			●		PlanetLab	Linux		
Sword		●				PlanetLab	Linux	Java	
Vxargs	●		●			Any	Linux, Unix	Python	LGPL

6. TEACHING EXPERIENCES

This section presents some teaching experiences related to distributed systems. They are only samples and describe different approaches to teaching this subject.

Polytechnic Institute of New York University proposes [129] to use games to teach about distributed systems. Students must analyze and manage a complex distributed system already working, and the teacher introduces some errors in the system. Students must look for these errors and solve them. They work on teams and receive game points for the time the application is working properly. In this way, using games students try to understand how the application works and learn about common errors in distributed applications.

StarHPC [130] is a teaching experience from the Massachusetts Institute of Technology. They propose to use Amazon Elastic Cloud Computing (Amazon EC2) to run practices on distributed systems. Practices are concentrated on some weeks per year, so the cost of this solution is cheaper than maintaining a dedicated compute cluster. Other advantage is resource availability, because with a dedicated cluster students must compete for resources to run their practices, and with EC2 they can use as many instances as they need at the same time. They have prepared an Amazon EC2 virtual machine image to run on the elastic compute cloud, a virtual machine image to run on the student computer and some administrative scripts to manage Amazon EC2 cluster.

PlanetLab@UOC [131] is another teaching experience from Open University of Catalonia (UOC) at Barcelona, Spain. They use a distributed architecture where student instances of a distributed application communicate and interact with other instances developed by the teacher and deployed on PlanetLab nodes. Students can focus on develop their instance of the application, and test it in a distributed and real environment like PlanetLab. Student implementation can interact with already working and deployed implementations of the application.

7. CONCLUSIONS

In this paper we tried to present a "state of the art" on distributed systems, showing an overview on this matter. Not all platforms, toolkits, algorithms and tools are presented here, but we think more important ones for each category have been presented. We hope that all resources included in this paper could be useful for people who want to learn or teach about distributed systems.

REFERENCES

[1] Tselentis, G., Domingue, J., Galis, A., Gavras, A. and Hausheer, D. (2009): "Towards the Future Internet - A European Research Perspective". IOS Press, May 2009. ISBN 978-1-60750-007-0.

[2] Feldmann, A. (2007): "Internet clean-slate design: What and why?", in ACM SIGCOMM Computer Communication Review, vol. 37, no. 3, pp. 59-64, July 2007.

[3] Gavras, A., Karila, A., Fdida, S., May, M. and Potts, M. (2007): "Future internet research and experimentation: the FIRE initiative", in ACM SIGCOMM Computer Communication Review, ISSN:0146-4833, Volume 37, Issue 3 (July 2007), pages 89-92

[4] The Network Simulator ns-2. Retrieved March 22, 2010, from http://nsnam.isi.edu/nsnam/index.php/User_Information

[5] Chang, X. (1999): "Network Simulations with OPNET", in Proceedings of the 1999 Winter Simulation Conference.

[6] OPNET university program. Retrieved March 25, 2010, from http://www.opnet.com/university_program/index.html

[7] Galuba, W., Aberer, K., Despotovic, Z. and Kellerer, W. (2009): "ProtoPeer: a P2P toolkit bridging the gap between simulation and live deployement", in Proceedings of the 2nd international Conference on Simulation Tools and Techniques (Rome, Italy, March 02 - 06, 2009). International Conference On Simulation Tools And Techniques For Communications, Networks And Systems & Workshops. ICST (Institute for Computer Sciences Social-Informatics and Telecommunications Engineering), ICST, Brussels, Belgium, 1-9.

[8] ProtoPeer. Distributed Systems Prototyping Toolkit. Retrieved May 15, 2010, from http://protopeer.epfl.ch/

[9] Mahadevan, P., Yocum, K. and Vahdat A. (2002): "Emulating large-scale wireless networks using ModelNet". Poster and Abstract Mobicom 2002, September 2002.

[10] ModelNet. UCSD Computer Science. Retrieved April 7, 2010, from https://modelnet.sysnet.ucsd.edu/

[11] White, B., Lepreau, J., Stoller, L., Ricci, R., Guruprasad, S., Newbold, M., Hibler, M., Barb, C. and Joglekar, A. (2002): "An Integrated Experimental Environment for Distributed Systems and Networks". In Proceedings of the Fifth USENIX Symposium on Operating System Design and Implementation (OSDI), pages 255-270, Boston, Massachusetts, December 2002.

[12] Emulab. Total network testbed. Flux Group, School of Computing, University of Utah. Retrieved April 7, 2010, from http://www.emulab.net/

[13] Bavier, A., Bowman, M., Culler, D., Chun, B., Karlin, S., Muir, S., Peterson, L., Roscoe, T., Spalink, T. and Wawrzoniak, M. (2004): "Operating System Support for Planetary-Scale Network Services". In Proc. 1st NSDI, San Francisco, California, March 2004.

[14] Peterson, L. and Roscoe, T. (2006): "The Design Principles of PlanetLab", ACM SIGOPS Operating Systems Review archive, 40(1): 11-16, January 2006.

[15] PlanetLab. An open platform for developing, deploying, and accessing planetary-scale services. Retrieved May 17, 2010, from http://www.planet-lab.org/

[16] SatelliteLab, Internet Access Networks project, Max Planck Institute for Software Systems. Retrieved April 9, 2010, from http://satellitelab.mpi-sws.org/

[17] Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I. and Zaharia, M. (2009): "Above the Clouds: A Berkeley View of Cloud Computing". UC Berkeley RAD Laboratory, http://berkeleyclouds.blogspot.com/. February 2009.

[18] Amazon Elastic Compute Cloud (Amazon EC2). Retrieved April 8, 2010, from http://aws.amazon.com/ec2/

[19] Murty, J.(2008): "Programming Amazon Web Services: S3, EC2, SQS, FPS, and SimpleDB". O'Reilly Media Inc., 2008. ISBN 0596515812, 9780596515812

[20] Ciurana, E. (2009): "Developing with Google App Engine". Firstpress, Apress. ISBN: 1430218312

[21] Google App Engine. Run your web apps on Google's infrastructure. Retrieved May 18, 2010, from http://code.google.com/intl/en/appengine/

[22] Chappell, D. (2009): "Introducing the Windows Azure Platform". Retrieved May 30, 2010, from http://view.atdmt.com/action/mrtyou_FY10AzurewhitepaperIntroWindowsAzurePl_1

[23] Hay, C. and Prince, B. H. (2010): "Azure in Action". ISBN: 9781935182481

[24] Windows Azure Platform. Retrieved May 18, 2010, from http://www.microsoft.com/windowsazure/

[25] Science Clouds. Retrieved June 6, 2010, from http://www.scienceclouds.org/

[26] Nimbus. Retrieved June 6, 2010, from http://www.nimbusproject.org/

[27] GridCafé. The place for everybody to learn about grid computing. Retrieved June 6, 2010, from http://www.gridcafe.org/

[28] Foster, I. and Kesselman, C. (2003): "The Grid 2: Blueprint for a New Computing Infrastructure", Oxford, UK Morgan Kaufmann Publishers, Inc. ISBN: 1558609334

[29] Beckman, P. H. (2005): "Building the TeraGrid". Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences, Vol. 363, No. 1833, Aug 2005, pp. 1715 – 1728.

[30] TeraGrid. Retrieved May 18, 2010, from https://www.teragrid.org/

[31] Grid'5000 homepage. Retrieved June 6, 2010, from http://www.grid5000.fr

[32] Anderson, D.P. (2004): "BOINC: A System for Public-Resource Computing and Storage". 5th IEEE/ACM International Workshop on Grid Computing, 2004.

[33] Boinc. Open-source software for volunteer computing and grid computing. Berkeley University of California. Retrieved April 8, 2010, from http://boinc.berkeley.edu/

[34] The BOINC application programming interface (API). Retrieved June 6, 2010, from http://boinc.berkeley.edu/trac/wiki/BasicApi

[35] Cappos, J., Beschastnikh, I., Krishnamurthy, A. and Anderson, T. (2009): "Seattle: A Platform for Educational Cloud Computing,". In Proceedings of the 40th ACM technical symposium on Computer science education. Chattanooga, TN, USA.

[36] Seattle. Open Peer-to-Peer Computing. Computer Science & Engineering. University of Washington. Retrieved March 15, 2010, from https://seattle.cs.washington.edu/wiki

[37] Thain, D., Tannenbaum, T. and Livny, M. (2005): "Distributed Computing in Practice: The Condor Experience". Concurrency and Computation: Practice and Experience, Vol. 17, No. 2-4, pages 323-356, February-April, 2005.

[38] Condor. Hight Throughput Computing. Computer Sciences. University of Wisconsin. Retrieved April 8, 2010, from http://www.cs.wisc.edu/condor/

[39] Miranda, H., Pinto, A., Carvalho, N., Rodrigues, L. (2005): "Appia Protocol Development Manual". Retrieved June 1, 2010, from http://appia.di.fc.ul.pt/docs/appia-pdm-2-1.pdf

[40] Mena, S., Cuvellier, X., Gregoire, C. and Schiper, A. (2003): "Appia vs. Cactus: comparing protocol composition frameworks". Proceedings 22nd International Symposium on Reliable Distributed Systems (SRDS'03), pages 189–198, October 2003.

[41] Appia Communication Framework. Retrieved April 3, 2010, from http://appia.di.fc.ul.pt/wiki/

[42] Hiltunen, M. A. and Schlichting, R. D. (2000): "The Cactus approach to building configurable middleware services," in Proceedings of the Workshop on Dependable System

Middleware and Group Communication (DSMGC 2000), (Nuremberg, Germany), October 2000.

[43] The Cactus Project. The University of Arizona, Computer Science Department. Retrieved June 1, 2010, from http://www.cs.arizona.edu/projects/cactus/

[44] Killian, C. E., Anderson, J. W., Braud, R., Jhala, R. and Vahdat, A. M. (2007): “Mace: language support for building distributed systems”. In PLDI’07.

[45] The Mace Project. Retrieved May 18, 2010, from http://www.macesystems.org/

[46] Loo, B. T., Condie, T., Hellerstein, J., Maniatis, P., Roscoe, T. and Stoica, I. (2005): "Implementing declarative overlays", in Proceedings SOSP, Brighton, UK, Oct. 2005.

[47] P2. Declarative Networking. Retrieved June 6, 2010, from http://p2.cs.berkeley.edu/

[48] Welsh, M., Culler, D. and Brewer, E. (2001): "SEDA: An Architecture for Well-Conditioned, Scalable Internet Services", in Proceedings of the Eighteenth Symposium on Operating Systems Principles (SOSP-18), Banff, Canada, October, 2001.

[49] SEDA: An Architecture for Highly Concurrent Server Applications. Retrieved June 6, 2010, from http://www.eecs.harvard.edu/~mdw/proj/seda/

[50] NBIO: Nonblocking I/O for Java. Retrieved June 6, 2010, from http://www.eecs.harvard.edu/~mdw/proj/java-nbio/

[51] Leonini, L., Rivière, É. and Felber, P. (2009): “SPLAY: distributed systems evaluation made simple (or how to turn ideas into live systems in a breeze)”. In Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation (Boston, Massachusetts, April 22 - 24, 2009). USENIX Association, Berkeley, CA, 185-198.

[52] Splay Project. Distributed applications made simple. Retrieved May 20, 2010, from http://www.splay-project.org/

[53] Lua, the programming language. Retrieved June 5, 2010, from http://www.lua.org/

[54] Laure, E., Fisher, E., Fisher, S. M., Frohner, A., Grandi, C. and Kunszt, P. (2006): "Programming the Grid with gLite". Computational Methods in Science and Technology, 12(1):33–45, 2006.

[55] gLite. Lightweight Middleware for Grid Computing. Retrieved June 6, 2010, from http://glite.web.cern.ch/glite/

[56] R-GMA: Relational Grid Monitoring Architecture. Retrieved June 6, 2010, from http://www.r-gma.org/

[57] Foster, I. (2005): “Globus Toolkit Version 4: Software for Service-Oriented Systems”. IFIP International Conference on Network and Parallel Computing, Springer-Verlag LNCS 3779, pp 2-13, 2005.

[58] Foster, I. and Kesselman, C. (1997): “Globus: A Metacomputing Infrastructure Toolkit”. Intl J. Supercomputer Applications, 11(2):115-128, 1997.

[59] Globus Toolkit Homepage. Retrieved May 13, 2010, from http://www.globus.org/toolkit/

[60] OGSA Primer. Retrieved June 5, 2010, from http://www.ogf.org/OGSA_Primer/

[61] Open Grid Forum. Open forum, Open standards. Retrieved June 5, 2010, from http://www.ogf.org/

[62] OASIS Web Services Resource Framework (WSRC). Retrieved June 6, 2010, from http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf

[63] OASIS. Advancing open standards for the information society. Retrieved June 6, 2010, from http://www.oasis-open.org/home/index.php

[64] Unicore (Uniform Interface to Computing Resources). Retrieved June 5, 2010, from http://www.unicore.eu/

[65] Lua, K., Crowcroft, J., Pias, M., Sharma, R., Lim, S. (2005): "A survey and comparison of peer-to-peer overlay network schemes", in Communications Surveys & Tutorials, IEEE, 2005.

[66] Ratnasamy, S., Francis, P., Handley, M., Karp, R. and Shenker, S. (2001): “A scalable content addressable network”, in Proceedings of the ACM SIGCOMM, 2001, pages 161–172.

[67] Stoica, I., Morris, R., Karger, D., Kaashoek, F. and Balakrishnan, H. (2001): “Chord: A scalable peer to peer lookup service for internet applications”, in Proceedings SIGCOMM, 2001.

[68] Dabek, F., Brunskill, E., Kaashoek, M. F., Karger, D., Morris, R., Stoica, I. and Balakrishnan, H. (2001): "Building Peer-to-Peer Systems with Chord, a Distributed Lookup Service", in Proceedings of the 8th Workshop on Hot Topics in Operating Systems (HotOS), Schloss Elmau, Germany, May 2001. IEEE Computer Society

[69] Dabek, F., Kaashoek, M. F., Karger, D., Morris, R. and Stoica, I. (2001): “Wide-area cooperative storage with CFS”, in Proceedings of the eighteenth ACM Symposium on Operating Systems Principles (New York, NY, USA, 2001), ACM Press, pages 202–215.

[70] Cox, R., Muthitacharoen, A. and Morris, R. (2002): “Serving DNS using Chord”, in Proceedings of the First International Workshop on Peer-to-Peer Systems, March 2002.

[71] Rowstron, A. and Druschel, P. (2001): “Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems”. In Proceedings IFIP/ACM International Conference on Distributed Systems Platforms (Middleware) 2001, Heidelberg, Germany, pages 329–350.

[72] Pastry. A substrate for peer-to-peer applications. FreePastry. Retrieved June 2, 2010, from http://www.freepastry.org/

[73] Plaxton, C., Rajaraman, R. and Richa, A. (1997): “Accessing nearby copies of replicated objects in a distributed environment,” in Proceedings of the 9th Annual ACM Symposium on Parallel Algorithms and Architectures, 1997.

[74] SimPastry. Microsoft Research. Retrieved June 2, 2010, from https://research.microsoft.com/en-us/downloads/33f10bf8-42bd-4c7f-b24c-ad73d4ce2ff7/default.aspx

[75] Rowstron, A., Kermarrec, A.-M., Castro, M. and Druschel, P. (2001): "SCRIBE: The design of a large-scale event notification infrastructure", in Proceedings Third International Workshop on NGC, 2001.

[76] Castro, M., Druschel, P., Kermarrec, A.-M., Nandi, A., Rowstron, A. and Singh, A. (2003): "SplitStream: High-bandwidth content distribution in cooperative environments", in Proceedings SOSP, 2003.

[77] Iyer, S., Rowstron, A. and Druschel, P. (2002): "Squirrel: A decentralized peer-to-peer web cache", in Proceedings of the 21st Symposium on Principles of Distributed Computing (PODC), Monterey, California, USA, July 21-24 2002.

[78] Druschel, P. and Rowstron, A. (2001): "PAST: A large-scale, persistent peer-to-peer storage utility", in Proceedings of the 8th Workshop on Hot Topics in Operating Systems (HotOS-VIII). Schloss Elmau, Germany: IEEECompSoc, May 2001.

[79] Zhao, B. Y., Huang, L., Stribling, J., Rhea, S. C., Joseph, A. D. and Kubiatowicz, J. D. (2004): "Tapestry: A resilient global-scale overlay for service deployment". IEEE Journal on Selected Areas in Communications, vol. 22, no. 1, pp. 41–53, January 2004.

[80] Chimera. Retrieved June 3, 2010, from http://current.cs.ucsb.edu/projects/chimera/

[81] Kubiatowicz, J., Bindel, D., Chen, Y., Eaton, P., Geels, D., Gummadi, R., Rhea, S., Weatherspoon, H., Weimer, W., Wells, C. and Zhao, B. (2002): "OceanStore: An architecture for global-scale persistent storage", in Proceedings of the ACM ASPLOS, November 2002.

[82] Zhuang, S. Q., Zhao, B. Y., Joseph, A. D., Katz, R. H. and Kubiatowicz, J. D. (2001): "Bayeux: an architecture for scalable and fault-tolerant wide-area data dissemination", in Proceedings of the 11th international workshop on Network and operating systems support for digital audio and video, 2001, pages 11–20.

[83] Zhou, F., Zhuang, L., Zhao, B. Y., Huang, L., Joseph, A. D. and Kubiatowicz, J. D. (2003): "Approximate object location and spam filtering on peer-to-peer systems", in Proceedings of the Middleware, June 2003.

[84] Rhea, S., Geels, D., Roscoe, T. and Kubiatowicz, J. (2004): "Handling churn in a DHT", in USENIX Tech, 2004.

[85] The Bamboo Distributed Hash Table. A Robust, Open-Source DHT. Retrieved June 1, 2010, from http://bamboo-dht.org/

[86] Maymounkov, P. and Mazieres, D. (2002): "Kademlia: A peer-to-peer information system based on the xor metric", in Proceedings of IPTPS02, Cambridge, USA, March 2002.

[87] Kademlia: A Design Specification. Retrieved June 15, 2010, from http://xlattice.sourceforge.net/components/protocol/kademlia/specs.html

[88] Khashmir. Retrieved June 15, 2010, from http://khashmir.sourceforge.net/

[89] Francis, P., Jamin, S., Jin, C., Jin, Y., Raz, D., Shavitt, Y. and Zhang, L. (2001): "IDMaps: A global Internet host distance estimation service". IEEE/ACM Transactions on Networking, October 2001.

[90] Ng, T. S. E. and Zhang, H. (2002): "Predicting Internet network distance with coordinates-based approaches", in Proceedings of IEEE Infocom, pages 170–179, 2002.

[91] Ng, T. S. E. and Zhang, H. (2004): "A network positioning system for the Internet", in Proceeding of USENIX Conference, June 2004.

[92] Dabek, F., Cox, R., Kaashoek, F. and Morris, R. (2004): "Vivaldi: A decentralized network coordinate system", in Proceedings of the ACM SIGCOMM '04 Conference (Portland, Oregon).

[93] Ledlie, J., Gardner, P. and Seltzer, M. (2007): "Network coordinates in the wild", in Proceedings of NSDI'07, April 2007.

[94] Azureus – now called Vuze – Bittorrent Client. Retrieved June 3, 2010, from http://azureus.sourceforge.net/

[95] Chen, Y., Xiong, Y., Shi, X., Zhu, J., Deng, B. and Li, X. (2009): "Pharos: Accurate and decentralized network coordinate system", in IET Communications, vol. 3, no. 4, pages 539–548, April 2009.

[96] Burgess, M. (1995): "Cfengine: a site configuration engine". USENIX Computing systems, 8(3).

[97] Cfengine. Continuous datacenter automation and repair. Retrieved May 21, 2010, from http://www.cfengine.org/

[98] CoDeeN. A Content Distribution Network for PlanetLab. Retrieved June 4, 2010, from http://codeen.cs.princeton.edu/

[99] CoDeploy. A Scalable Deployment Service for PlanetLab. Retrieved June 4, 2010, from http://codeen.cs.princeton.edu/codeploy/

[100] Park, K. and Pai, V. S. (2006): "CoMon: A Mostly-Scalable Monitoring System For PlanetLab", SIGOPS Operating Systems Review, Vol. 40, Num. 1, pp. 65-74, 2006.

[101] CoMon. A Monitoring Infrastructure for PlanetLab. Retrieved May 21, 2010, from http://comon.cs.princeton.edu/

[102] Park, K. and Pai, V. CoTop. A Slice-Based Top for PlanetLab. Retrieved April 10, 2010, from http://codeen.cs.princeton.edu/cotop/

[103] CoVisualize. A Visualization of PlanetLab Usage. Retrieved June 4, 2010, from http://codeen.cs.princeton.edu/covisualize/

[104] Slicestat. Retrieved May 12, 2010, from http://codeen.cs.princeton.edu/slicestat/

[105] DSMT. Distributed Service Management Toolkit. An open platform for managing planetary-scale services. Retrieved June 3, 2010, from http://www.dsmt.org/

[106] Massie, M. L., Chun, B. N. and Culler, D. E. (2004): “The Ganglia distributed monitoring system: Design, implementation, and experience”. Parallel Computing, 30(7), July 2004.

[107] Ganglia Monitoring System. Retrieved April 3, 2010, from http://ganglia.info/

[108] About RRDtool. Retrieved June 3, 2010, from http://oss.oetiker.ch/rrdtool/index.en.html

[109] Chun, B.N., GEXEC, Scalable Cluster Remote Execution System, California Institute of Technology. Retrieved April 3, 2010, from http://www.theether.org/gexec/

[110] Gush: GENI User Shell. Retrieved June 4, 2010, from http://gush.cs.williams.edu/trac/gush/wiki

[111] GENI. Exploring Networks of the Future. Retrieved June 4, 2010, from http://www.geni.net/

[112] PlanetLab Management Using Plush and Nebula. Retrieved June 4, 2010, from http://plush.cs.williams.edu/nebula/

[113] Management Overlay Networks (MON). Retrieved June 3, 2010, from http://cairo.cs.uiuc.edu/mon/

[114] AquaLab. The Nixes Tool Set. Retrieved June 4, 2010, from http://www.aqualab.cs.northwestern.edu/nixes.html

[115] Huebsch, R. PlanetLab Application Manager. Retrieved March 24, 2010, from http://appmanager.berkeley.intel-research.net/

[116] PlanetLab experiment manager. Retrieved June 4, 2010, from http://www.cs.washington.edu/research/networking/cplane/

[117] PLDeploy. Retrieved June 4, 2010, from http://psepr.org/tools/pldeploy/HOWTO.php

[118] Albrecht, J., Tuttle, C., Snoeren, A. C. and Vahdat, A. (2006): “PlanetLab Application Management Using Plush”. ACM Operating Systems Review (SIGOPS-OSR), 40(1), January 2006.

[119] Plush. PlanetLab User Shell. Retrieved May 21, 2010, from http://plush.cs.williams.edu/

[120] pShell. An Interactive Shell for Managing PlanetLab Slices. Retrieved June 5, 2010, from http://cgi.cs.mcgill.ca/~anrl/projects/pShell/index.php

[121] PSSH. Retrieved June 4, 2010, from http://www.theether.org/pssh/

[122] Goldsack, P., Guijarro, J., Lain, A., Mecheneau, G., Murray, P. and Toft, P. (2003): “SmartFrog: Configuration and automatic ignition of distributed applications”. In HP OVUA, 2003.

[123] SmartFrog Home. Retrieved May 21, 2010, from http://www.smartfrog.org

[124] Stork. Retrieved June 5, 2010, from http://www.cs.arizona.edu/stork/

[125] Albrecht, J., Oppenheimer, D., Vahdat, A. and Patterson, D. A. (2008): “Design and implementation tradeoffs for wide-area resource discovery”. ACM Transactions on Internet Technology (TOIT), 8(4) (Sep. 2008), 1-44.

[126] Oppenheimer, D., Albrecht, J., Patterson, D. and Vahdat, A. (2004): “Distributed resource discovery on PlanetLab with SWORD”. First Workshop on Real, Large Distributed Systems (WORLDS '04).

[127] SWORD on PlanetLab: Scalable Wide-Area Resource Discovery. Retrieved May 21, 2010, from http://sword.cs.williams.edu/

[128] Mao, Y. VXARGS: running arbitrary command with explicit parallelism, visualization and redirection. Pennsylvania University. Retrieved April 3, 2010, from http://vxargs.sourceforge.net/

[129] Wein, J., Kourtchikov, K., Chang, Y. et al. (2009): “Virtualized Games for Teaching about Distributed Systems”, in Proceedings of the 40th ACM technical symposium on Computer science education, SIGCSE 2009, 246-250.

[130] Ivica, C., Ridley, J.T. and Shubert, C. (2009): “Teaching Parallel Programming within Elastic Compute Cloud”, in Proceedings of the ITI 2009.

[131] Marquès, J.M., Lázaro, D, Juan, A., Vilajosana, X., Domingo, M. and Jorba, J. (20xx): “PlanetLab@UOC: A Real LAb over the Internet to experiment with Distributed Systems”. Computer Applications in Engineering Education. ISSN: 1099-0542. [Accepted May 2010. Publication pending]