Plan 9 – an Integrated Approach to Grid Computing

Andrey Mirtchovski
University of Calgary
mirtchov@cpsc.ucalgary.ca

Rob Simmonds
University of Calgary
simmonds@cpsc.ucalgary.ca

Ron Minnich
*Los Alamos National Laboratory**
rminnich@lanl.gov

Abstract

This paper describes the use of the "Plan 9 from Bell Labs" distributed operating system as a Grid Computing infrastructure. In particular it compares solutions using the de facto standard middleware toolkit for grids, Globus, to an environment constructed using computers running the Plan 9 operating system. These environments are compared based on the features they offer in the context of grid computing: Authentication, Security, Data Management, and Resource Discovery.

1. Introduction

Grid Computing describes computation in which jobs are run on a distributed computational unit spanning two or more administrative domains. It has sparked tremendous excitement among scientists worldwide and has renewed the interest of the scientific community toward distributed computing, an area which was almost forgotten during the 90's.

A relatively new but fast growing field in Computer Science, Grid Computing involves the utilization of disparate resources toward the goal of solving complex problems requiring technology at a level not always available to a single organization. Many scientists believe that the next evolutionary step of High Performance Computing will require a departure from the single all-encompassing supercomputer toward arrays of small, heterogeneous *clusters* of computers, much like today's Beowulf-style [1] installations [2].

Due to their single-purpose design and the fact that they're tightly coupled with the specific organization that builds them, clusters do not address one of the major requirements of today's scientific world – cross organization collaborational computational. Supercomputers are still quite expensive to buy, and it is often the case that a research or commercial organization is able to afford only one such system. In contrast, by allowing other organizations to share their computational resources, a distributed environment could be created comprising a sum of the shared entities. In other words, scientists need software that is able to utilize resources not only physically, but administratively separated from the original host's environment.

Several toolkits are being developed that provide a distributed environment without requiring large modifications in the code of the programs being run. Here we discuss one of them: *Globus*, the open source toolkit which has become the *de facto* standard in Grid Computing and compare it with *Plan 9 from Bell Labs*, a distributed operating system able to create and maintain per-process distributed environments independent of the physical location of resources.

Globus and Plan 9 represent two different approaches to creating Grids. Globus provides an add-on to enable Grid computing on top of other operating systems, while Plan 9 allows for the creation of Grids with no changes or additional middleware.

Globus offers a set of user-level tools to provide an enabling grid infrastructure for legacy operating systems such as Unix and Windows. Since the basic architecture of these operating systems predates networking and the Internet, they are not well suited to a distributed environment. Globus is therefore needed so that they can function as components of a distributed computing architecture.

Plan 9 on the other hand was designed from the ground up as a distributed system: the architecture of Plan 9 is inherently *grid-enabled*. As we show in this paper, our Grid system "*9grid*" is functioning already as a Grid with no additional software needed; it took just a few days to set up the initial sites and start computations. Thus, Plan 9 can be

*Los Alamos National Laboratory, an affirmative action/equal opportunity employer, is operated by the University of California for the United States Department of Energy, under contract W-7405-ENG-36. LANL publication: LA-UR-03-8428

thought of as taking an *integrated* approach to Grid computing.

Each system has its advantages. Globus provides add-on support for legacy systems that could not be used in a Grid in any other way, and hence can take advantage of the huge base of applications software that those legacy systems support. Plan 9, with its integrated approach to Grid computing, has an advanced security infrastructure and is less difficult to set up in a distributed environment.

Other, commercial solutions for Grid Computing exist, however they haven't matured and gained wide acceptance yet, partly due to the fact that their closed-source nature does not fit well with the current trend towards Open Source solutions among the scientific community.

1.1. Globus

The Globus toolkit [17] was created in the late 1990s as part of a joint research project between Argonne National Laboratory and the Information Sciences Institute at the University of Southern California. Its aim was to provide a solution to the computational needs of large virtual organizations [4] that span multiple institutional and administrative domains. Globus is a middleware toolkit that provides fundamental distributed computing services such as authentication, job starting and resource discovery.

While there are other Grid computing toolkits available, Globus currently provides a de facto standard due to its development being closely tied to the formulation of the Open Grid Services Architecture (OGSA) standard [3]. Once OGSA and other standards being developed by the Global Grid Forum (GGF) are finalized, it is likely that commercial toolkits that adopt these standards, and therefore can inter-operate with Globus environments, will become more popular.

Globus provides a collection of *services* [5] including: **GSI**, Grid Security Infrastructure which provides authentication based on a Certificate Authority trust model; **GRAM**, Grid Resource Allocation Manager which handles job starting or submission; **GridFTP**, providing extensions to the FTP standard to provide GSI authentication and high performance transfer; **MDS**, Monitoring and Discovery Service enabling remote resource discovery.

By itself Globus does not provide all of the tools and services required to implement a full featured distributed computing environment. Additional tools are available to fill some of the gaps. The National Center for Supercomputing Applications (NCSA) provides a patch to add GSI authentication to OpenSSH. This allows Globus environments to have terminal based single-signon. Globus does not provide any scheduling functionality, but rather relies on the client operating system scheduler or batch schedulers such as OpenPBS [20] to handle local scheduling activities. Global scheduling between Globus processes can be provided by meta-schedulers, such as Condor-G [21]. Condor-G submits jobs to the GRAM service running on Globus nodes and GRAM handles the task of submitting the job to the local scheduling system.

Other additional services, such as high level data management services and credential repositories will be discussed in later sections.

1.2. Plan 9

Plan 9 [7] is a distributed operating system created in the late 80's by a research group in Bell Labs, the same one that designed UNIX; the first Plan 9 distribution was released in 1991. Plan 9 is an attempt to address fundamental issues with UNIX's design, some of which come from the fact that it was originally concieved in an environment lacking many familiar tools, such as networking or graphical display terminals. Both networking and graphics were added later on in the life of the OS, but every new layer accommodating a new type of hardware has created more complexity due to the large amounts of code associated with it, and the constantly changing interface requirements. Plan 9 is designed with simplicity in mind, which is best illustrated by comparing the two code bases for Globus and Plan 9: the Globus Toolkit v2.4 consumes, in archived form, as much space as the entire Plan 9 operating system image.

As issues in UNIX were addressed in Plan 9's design, the system was also extended beyond the timesharing mainframe and personal workstation environment to accommodate networks and the fast growing cheap commodity hardware market. The early trend of the 1990's was to switch away from the organizational mainframe and into cheaper workstation-based computing, thus Plan 9 was designed to create a distributed environment comprised of cheap computers as terminals and fast, expensive servers as stirage and CPU cycles providers [7].

Other distributed operating systems such as Sprite [22] and Amoeba [23] exist, but the environments they build are tightly coupled within the OS, making communication with external services difficult. Such systems suffer from the radical departure from the UNIX model, which also discourages portability of already existing software to the platform or in the case of Sprite, evolve into a file system running on top of UNIX kernels. The lack of developers, the very small range of supported hardware and the small, even compared to Plan 9, user base have also significantly slowed the adoption of those systems outside of their research communities. In retrospect, Plan 9 was the only research distributed OS from that time which managed to attract developers and be used in commercial projects long enough to warrant its survival to this day. In Plan 9 the ANSI-Posix environment [6] solves the portability problem. Most non-graphical UNIX

applications can easily be compiled for Plan 9, and in most cases rewriting an application to run natively on Plan 9 involves removing large chunks of code dealing with inconsistencies between UNIX variants, which simply do not exist in Plan 9.

What separates Plan 9 from other distributed systems is the ease with which design considerations for new computing models were accommodated with widely adopted and accepted UNIX paradigms. For example all resources in Plan 9, including files, devices, graphical, network and other subsystems, are represented as files and directories comprising a hierarchical file structure called a *namespace*. A uniform access protocol means that in contrast to Linux for example, which has almost 300 system calls to manage many different types of resources, Plan 9 has 40 system calls and a uniform method for enumerating and controlling resources, which are presented to applications as a directory structure.

Resource control is accomplished via a simple and consistent interface. Each device or file system presents at least two files for process interaction – one for control messages and one for data, usually called `ctl` and `data`. Control operations on the device are performed by writing plain text messages to the control file, status is obtained by reading from the same file. Any data that needs to be transferred to and from the device is read or written from the data file. This interface extends to all resources in the system and is universally adopted throughout the environment. It also works transparently over a network, as do all Plan 9 operations. For example, a process or a device such as network interface on a remote computer can be imported and controlled as though it was local. Resources are functionally equivalent regardless of their location on the grid.

The different services provided by the operating systems are joined together as needed in a single namespace, private to the process which created it. This process can pass the namespace on to its children unmodified, and even export it for descendants running on other nodes.

Plan 9 also separates CPU (doing the computation), terminal (providing end-user displays) and storage servers so that they can run on different hardware, which may be optimized particularly for the job; for example it is not uncommon to run multiprocessor servers as CPU nodes and have a hardware RAID controller serving as the storage device. This greatly reduces the cost of hardware and maintenance since it concentrates expensive equipment in the server room, while users may run on inexpensive terminals containing just a display, mouse and keyboard. The protocols used to access file servers are security hardened to the extent that servers need not hide behind firewalls. The main Plan 9 file server at Bell Labs is outside the Bell Labs firewall.

2. Authentication

Our discussion of authentication mechanisms in a grid environment is concerned with the ability of the environment management system to handle proving a user's identity and the deployment and storage of security information. This section presents the authentication mechanisms currently in use by Globus and Plan 9 and expands on possible ways of extending accounts on the grid to carry information relevant to an environment spanning multiple administrative domains.

2.1. Authentication in Globus

Globus employs an authentication scheme using X.509 certificates for user identification, and TLS and SSL protocols for transport layer security [16]. Each user has a password protected private key and an X.509 certificate that is signed by a *Certificate Authority* (CA). In order for a user to authenticate with a Globus service, the service must trust the particular CA that signed the user's certificate. This trust is established by installing a set of files describing the CA in a secure directory on the host running the Globus service. The service trusts the CA to determine the correct identity of a user that it issues a certificate to and to revoke certificates that have been compromised in some way.

To initialize their authorization environment a user runs a command that uses their private key and signed certificate to generate a new certificate called a proxy certificate, that contains a digest of the private key and signed certificate. This proxy certificate is then used to authenticate with Globus services and provides single-signon to the Grid environment.

Each proxy certificate is given a limited lifetime to reduce the chance that a proxy certificate is stolen by an adversary and used by this adversary to launch an attack on its original owner. Proxy certificates are stored in temporary directories and protected only by the file-permission mechanism employed by the host OS. This means that the level of risk of having a proxy certificate stolen depends on security of each host OS in the environment.

As jobs are started on a remote host, authorization privileges are delegated to the remote host by generating a new proxy certificate on it. One security feature of this scheme is the ability to create "limited" proxy certificates that are only recognized when using the GridFTP service. This prevents a stolen proxy from being used for anything other than authenticating with GridFTP servers, i.e., being used to transfer files.

The MyProxy service [11] from NCSA provides the means for users to store proxy certificates in an on-line repository and to use these proxy certificates to generate short lived proxy certificates when required. This removes

the need for a user to have access to their private key when they are away from their home system. They do need to be able to connect to the MyProxy server holding a valid proxy certificate. MyProxy can also be used by trusted services, such as for example a particular meta-scheduler, to renew a proxy certificate for a user before starting a job.

2.2. Authentication in Plan 9

Authentication in the Plan 9 distributed environment [9] is delegated outside of the application and is performed by dedicated authentication, or *auth* servers. It is completely independent of the software, services or architecture underlying the environment it authenticates for.

The authentication agent in Plan 9 is called *factotum*. Factotum is the only program in Plan 9 that understands authentication protocols, security keys and the mechanisms for their deployment. This agent stores the authentication keys for the programs with which it shares the environment and performs authentication both as a client and as a server, being able to assume both sides of an identity proof dialog in the course of a single session.

Factotum does not communicate with external programs directly, instead it is *consulted* by local entities sharing its namespace whenever authentication information is needed. After receiving a request for establishing a trust relationship, a client will act as a proxy – relaying communication messages between the client's and server's factotums until a mutual authentication is reached.

Factotum also serves a double-purpose as a *single sign-on* agent with its ability to remember the currently active authentication tokens for the private namespace it serves. Factotum stores user-provided authentication information, such as VNC or SSH passwords [9]. Instead of prompting the user on each authentication, it simply uses the tokens it already holds, prompting only when they fail, which may happen with one-time password schemes such as crypto cards or netkey. In effect, this creates an environment where passwords are not prompted for more than once, and once authenticated with, remote servers remember the job's owner identity without compromising the integrity of the underlying protocols.

Factotum protects the security of the account it serves by holding all security keys in protected areas of main memory, making it invisible for other users on the system. It also protects the running process' image by disabling a kernel's ability to swap it out to secondary storage. Factotum is unable to keep state between restarts, because all keys are kept in volatile memory cleared before starting each new Plan 9 process. To initialize it a user either supplies the passwords requested, or bootstraps factotum on start-up by reading the keys from a general-purpose encrypted data storage called *secstore* [9], where access is controlled by an authentication mechanism separate from the system.

The security protocols currently in use by Plan 9 include X.509 certificates, RSA keys for use with SSH, the DES challenge-response protocol, and some plain-text password schemes such as the ones used by Telnet, FTP and VNC [9].

2.3. Plan 9 authentication in a Grid Environment

The proliferation of distributed and grid-like environments, each having multiple sites and numerous authentication domains, requires a reevaluation of the currently used authentication mechanisms, usually based on a name and a unique number designating a particular user on the system. Plan 9 has already avoided that by not using a numerical user ID, however the username of a participant is not sufficient to provide all necessary information about the user's identity in a distributed environment anymore, and the potential username clashes between grid sites need to be dealt with by administrators early on in the environment's design.

The scheme currently used by Globus creates a global user namespace, where all necessary user information is carried in their authentication information. Local grid nodes then choose their own mapping from the global user ID to a local user ID, of which there is a particular set and naming convention created beforehand to be used by Globus patrons. For example jobs started by user `andrey` at a local terminal run as user `b102` at site `B` and user `c374` at site `C`. The conversion between users is done transparently by GridFTP while copying the files.

We are currently investigating a new system for identifying user's memberships in organizations and locations across the grid without necessarily storing all user information on all sites. It involves designating users as members of authentication domains, where membership is carried by the user's identification name as it appears on different grid systems. For example, a process owned by user `andrey`, member of the `ucalgary` authentication and administrative domain, running on a remote grid node will appear there as user `andrey@ucalgary` and will be unable to access resources which a member of the remote administrative domain with the same username may have access to.

Local authentication servers are able to request credentials from the master auth servers for a member of a particular domain without having to store them locally. They can also refuse authentication to untrusted members, even though their credentials with the remote server may be valid. This is implemented by extending the authentication server's capabilities to include authentication via remote systems and auth servers not handled by the immediate network being authenticated for via authentication proxies.

This authentication scheme aids grids by allowing users to keep their desired usernames across administrative do-

mains, while avoiding clashes with local users' processes and providing administrators with an easy way of identifying the location of a particular user's process. The authentication server the user is assigned to in this case acts as a global authentication agent for this user's processes, authenticating their identity everywhere on the grid. It also simplifies administration by keeping all administration related tasks close to the originating site, instead of delegating them to a centralized administrative entity.

3. Security

In the context of Grid Computing, a secure environment is one that is able to protect communication between jobs on the system from third party observers, protect jobs from adverse effect caused by other jobs' actions, and protect jobs from resource starvation (prevent DoS attacks, or illegal resource utilization by other users).

Before comparing Globus and Plan 9's solutions to these problems, we highlight what is possibly the most important security feature Plan 9 brings to Grid Computing – the concept of *private namespaces* underlying every aspect of the distributed environment.

3.1. Private namespaces as a security feature in a grid environment

Private namespaces [10] ensure that communication between processes or clients is restricted only to the parties involved and is invisible to others. Each process on Plan 9 sees a private view of the underlying system comprised of different resources' file servers bound together in a tree-like hierarchy called a namespace.

The primary security feature of private namespaces is to restrict other clients from snooping over private communication channels, or knowing that they exist. User-level mounts of remote systems in a process' name space ensure that remote grid-enabled resources can be brought in on demand, and invisibly from third parties.

In Plan 9 child processes share their parent's namespace by default, unless a special argument has been given to `fork()` to create a copy of the namespace. Clients willing to share parts of their namespace, can do so by posting a file descriptor pointing to the root of that namespace to a special directory, which acts as a bulletin board for file descriptors. There is no restriction on what may be shared, except the restrictions imposed by file permissions associated with the exported files, i.e., nobody will be able to access a resource exported without any read or write permissions set.

3.2. Security in Globus

Current grids are difficult to operate securely due to their highly distributed nature, the fact that they involve a multitude of different hardware and software platforms, and the lack of a single system administration authority. Being an addition to the computing environment and not a standalone system, Globus' security features deal primarily with proving a user's identity and encrypting the connection between hosts, not with protecting the general integrity of the system [16].

One particular difficulty that needs to be addressed in Globus is protecting jobs from a breach in the security of the underlying system. For example a user submitting Globus jobs to a cluster cannot be guaranteed that the computational result isn't compromised or erroneous, or that the intellectual property of the work is preserved. There is no mechanism to restrict interaction between processes on the same installation and in many environments such as *high performance computing* it is desirable that jobs running on the same host, or sets of hosts, are able to communicate between each other, but not in a way easily snooped by other applications.

Using virtual environments restricts communication with other processes and secures the system, but has performance implications that are difficult to predict at best. There have been implementations of private name spaces by means of virtualization of the host OS [18], whether through support for private namespaces in the kernel [8], or through third-party solutions like VMWare, where a client's session on a particular node is started as a virtual machine containing resources available only to the user.

Sand-boxing approaches, including an implementation of private namespaces for various operating systems on which Globus runs [8] have generally failed to yield a secure system due to them being constrained to the particular node they run on, and the fact that there is no standard implementation of sand-boxes that would work in a heterogeneous environment.

3.3. Security in Plan 9

Plan 9 as an operating system does not rely on third party additions to handle communications with hardware or between nodes. It has been designed to enforce a strict security policy to which all programs must adhere [9].

Key elements of the security infrastructure in Plan 9 are the lack of *superuser* account and encryption of all communication via a ticket-based protocol using authentication mechanisms independent of the session or environment. The OS also provides encrypted data storage as a service. Furthermore, in Plan 9 all processes operate in private namespaces.

The kernel delegates most of the security infrastructure considerations to the interprocess communication drivers such as `mount`, `bind` and the protocol carrying all information across the system – *9P*. A grid job does not have to be concerned with the underlying security infrastructure. As long as it can ensure on its end that the security of its factotum is not compromised, for example by exporting factotum's namespace to other processes with write permissions, its environment is secure.

4. Resource discovery

Resource discovery services are provided in order to allow queries to be made about the existence and features of services available in a particular environment. There are two different approaches to resource discovery on the grid: creating a brand new set of tools suited especially to the new distributed environment, or extending a familiar set of operating system utilities to work in a networked fashion between different nodes of the system.

4.1. Resource discovery in Globus

The Globus Toolkit's Monitoring and Discovery Service, MDS [19], has adapted the OpenLDAP [17] resource discovery protocol. OpenLDAP is an open source solution which presents clients with a hierarchical view of system resources, allowing remote agents to query it for particular subsets of information.

Several different types of resources may be of interest to Grid jobs. These could include hardware architecture and hardware configuration, i.e. the number of CPUs and their speed, amount of memory and disk space available, network capabilities, etc. Also the operating system running on the computer, available software and its versions, current or past system utilization or any restrictions imposed on the system externally, such as prior reservations. Grid sites often choose to add their own additional resources to the set of default ones.

In its current form MDS requires specialized OpenLDAP clients to query the information. Administrators usually provide their own set of tools for sorting and filtering the information in a particular manner, including formatting it for different viewing formats, such as web pages, customizing and replacing the OpenLDAP ones. Sometimes such tools are developed by the Grid administrators and are different at each site, which may lead to discrepancies between sites, and may require rewriting a program for each site.

4.2. Resource discovery in Plan 9

Resource discovery solutions in the context of a single distributed environment have existed since the very first Plan 9 release in 1991. A special program, called `srv`, serves as a filter, relaying messages between two namespaces. A process willing to share its namespace with others can *post* a file descriptor pointing to the root of a namespace via the `srv` utility to a specialized location on the system – `/srv`. Jobs then `mount` the desired file server hierarchy from `/srv`, if file permissions allow.

Extending this scheme to a grid environment can be accomplished by a file server hierarchy, used instead of the flat directory style of `/srv`. The first level of this hierarchy contains a subdirectory for each site being part of the grid and currently operational. Sites register their readiness to participate by posting a file descriptor pointing to their own `/srv`.

Locally, each site maintains a `/srv` where nodes post file descriptors announcing operational readiness. At the leaf directory is a description of the state of a particular node, containing a set of files describing architecture, number of processes, memory, swap space, and other statistics.

Note that any non-leaf `/srv` contains no knowledge of the actual resources that have subscribed to it. It simply serves as a link between a name of a site, and the file descriptor pointing to the root of that site's `/srv`. There is no polling for resources except by the end client.

Using this scheme it is possible to obtain any set of information about the entire grid without having to resort to anything other than a set of tools already familiar to the system's users. For example, to find the names and IP addresses of all nodes currently active in the grid, a client can import /srv from the a main repository on the grid and run the command: `cat /srv/*/*/name` which will display the file containing the name of the node from each site's subscribed set of services. More complex examples are built with the same set of operations used to write shell scripts.

One of the benefits of such scheme, as was just illustrated, is that there is no requirement for specialized tools and *resource viewers* as with OpenLDAP – programs, scripts and users can use their own set of familiar utilities, such as `ls`, `cd` and `cat` to accomplish what otherwise may require a separate utility complete with graphical interface.

5. Data management

Data management involves data transfer and replication services, allowing objects to be moved between different grid nodes in a fast and secure manner. Replication tracks versions of objects or parts thereof, and their locations.

5.1. Data management in Globus

Globus has been designed around the concepts of data management and replication from its very beginning. One of the first protocols to officially be designated as part of the

Globus Toolkit was GridFTP [12], which allows fast data transfer between nodes on the grid and underlies important parts of many utilities built on top of Globus, including file staging and remote job execution. GridFTP supports some advanced networking features and optimizations, such as partial file transfers and parallel transfers.

Replication services [13] aid the user in locating data on the grid, often scattered across several geographically diverse locations. A replica management service is able to track various copies of a data file, to tag versioning information to data sets and offers reliable means for putting together a complete data object out of geographically and administratively disparate data files. Most current attempts to provide replication services for grids are researched as solutions built on top of the Globus toolkit [14].

5.2. Data management in Plan 9

In Plan 9 the unit of communication is the data stream of file operations. The protocol underlying every transaction, *9p*, is universal. Every remote operation is done via a mounted, or *imported*, name space and it is very easy to extend the set of name space operations to incorporate new ideas such as compressed, encrypted or parallelized data transfers, or even fault tolerant and cached network connections. User level caching of file server connections, for example, has been implemented already and has been in use for years.

Unlike GridFTP, the data transfer mechanism on a Plan 9 grid is transparent to the user. As users run processes on remote Grid nodes, the namespace on the remote node is exported from local node and imported to the remote node with no involvement from any of the user's processes. Plan 9 has no need of a data transfer mechanism such as GridFTP, because all functionality it provides is abstracted away into the private name space mechanism, which handles communications with remote file servers.

A Plan 9 *advanced* import exercises one or more of features of the `import` program, such as compression, encryption, fault tolerance or parallel data transfer. This feature then persists for the entire length of the connection, meaning that even though different programs may be used over that link, none of those programs need to know whether their data traverses the link compressed or encrypted. Fault tolerance can be achieved by imposing a filter called *aan*, or *always available network*, which buffers 9P messages whenever the network is down, and reestablishes a data stream connection when the network is back up.

Data replication in Plan 9 is accomplished via a replica service which exists as part of the operating system. It usually contains a main repository to which people can *push* additions and modifications of files in a fashion similar to CVS, from which clients can selectively download (replicate) files they need, or which have been more recently updated than the local copy.

The Plan 9 replica does not require all files of a particular repository to be local to the system, allowing for building a hierarchical structure of replicas, where a master repository could be queried to examine whether any data files on any grid node has changed, or to examine whether any local replica sites have had any updates.

It is also possible to query a replica without committing any changes, or to examine its log file for the history of a particular data set. Paired with the infinite history and back-log capability of a the Venti [15] permanent storage server where the data is archived, it is possible to retrieve any version of any data file ever stored in a replica.

6. Conclusion

We have described and compared two possibilities for managing Grid Computing environments: Globus, a middleware toolkit that adds grid services to existing operating systems, and Plan 9, an operating system providing features require to support large scale distributed computing as part of its design.

Naturally no comparison will be complete without evaluating pros and cons of the systems compared. In terms of features and usability the Globus Toolkit eases the transition from local clusters to the cross-organizational nature of the grid. It builds upon a mature core of operating systems widely used in scientific environments. Plan 9 on the other hand requires a departure from the familiar set of UNIX paradigms, but with its radical design facilitates network computing at a level other systems are unable to achieve.

Several obstacles to Plan 9's widespread adoption have hindered its growth in the past. It had a closed-source development model for the first ten years of its existence and some prohibitive restrictions in its initial open source license. These issues have now been resolved so there is no reason why the research community shouldn't investigate this OS as a possible solution for building computational grids.

Although Plan 9 was designed to work in a single administrative domain, the simplicity, clarity and generality of its model allow it to be extended to the inter-organizational world of grid computing. This is easier than adapting a set of systems originally created for a timeshared environment.

Plan 9 shouldn't be looked at as a replacement for Grid Toolkits, instead we advocate examining Plan 9 in its native environment and taking its design decisions into consideration when building future grids. We would like to see environments connecting together around a unified lightweight protocol such as 9P and the ability to simplify the creation of grid services to the level of Plan 9, where to create a grid service one only needs to export the namespace in which it

presents its files. We believe this will create a simpler and more secure Grid Computing environment.

References

[1] T. L. Sterling, J. Salmon, D. J. Becker, D. F. Savarese: *How to Build a Beowulf*, The MPI Press, 1999.

[2] G. Bell, J Grey: *What's next in high performance computing?* Communications of the ACM, vol 45, issue 2, 2002.

[3] S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman: *Grid Service Specification, Draft 3*, Global Grid Forum, July 2002.

[4] I. Foster, C. Kesselman, S. Tuecke: *The Anatomy of the Grid: Enabling scalable virtual organizations*, International J. Supercomputer Applications, 15(3), 2001.

[5] J. Nick I. Foster, C. Kesselman, S. Tuecke: *The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration*, Open Grid Service Infrastructure WG, Global Grid Forum, June 22, 2002.

[6] H. Trickey: *APE - The ANSI/POSIX Environment*, Plan 9 Programmer's Manual, Volume 2, AT&T Bell Laboratories, Murray Hill, NJ, 1995.

[7] R. Pike, D. Presotto, S. Dorward, B. Flandrena, K. Thompson, H. Trickey, P. Winterbottom: *Plan 9 from Bell Labs*, Computing Systems, 8(3):221-254, 1995.

[8] R. Minnich: *Private Namespaces for Linux*, Dr. Dobb's Journal, Dec 2001.

[9] R. Cox, E. Grosse, R. Pike, D. Presotto, S. Quinlan: *Security in Plan 9*, Proceedings of the 11th USENIX Security Symposium, pp. 3–16, 2002.

[10] R. Pike, D. Presotto, K. Thompson, H. Trickey, P. Winterbottom: *The Use of Name Spaces in Plan 9*, Op. Sys. Rev., Vol. 27, No. 2, April 1993, pp. 72-76.

[11] J. Novotny, S. Tuecke, V. Welch: *An Online Credential Repository for the Grid: MyProxy. Proceedings of the Tenth International*, Symposium on High Performance Distributed Computing (HPDC-10), IEEE Press, August 2001.

[12] Globus Project: *GridFTP - Universal Data Transfer for the Grid*, White Paper. September 5, 2000.

[13] D. Dullmann, W. Hoschek, J. Jean-Martinez, A. Samar, H. Stockinger, K. Stockinger: *Models for Replica Synchronisation and Consistency in a Data Grid*, 10th IEEE Symposium on High Performance and Distributed Computing

[14] K. Ranganathan, Adriana Iamnitchi, and I. Foster: *Improving Data Availability through Dynamic Model-Driven Replication in Large Peer-to-Peer Communities*, Proceedings of Global and Peer-to-Peer Computing on Large Scale Distributed Systems Workshop, Berlin, Germany, May 2002.

[15] S. Quinlan,S. Dorward: *Venti: a new approach to archival storage*, Conference on File and Storage Technologies, Monterey, CA, 28–30 January 2002.

[16] V. Welch, F. Siebenlist, I. Foster, J. Bresnahan, K. Czajkowski, J. Gawor, C. Kesselman, S. Meder, L. Pearlman, S. Tuecke: *Security for Grid Services*, Twelfth International Symposium on High Performance Distributed Computing (HPDC-12), IEEE Press, June 2003.

[17] S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman, T. Maguire, T. Sandholm, P. Vanderbilt, D. Snelling: *Open Grid Services Infrastructure (OGSI) Version 1.0*, Global Grid Forum Draft Recommendation, 6/27/2003.

[18] R. Figueiredo, P. Dinda, J. Fortes: *A Case for Grid Computing on Virtual Machines*, In Proc. Intl. Conf. on Distributed Computing Systems (ICDCS), 04/2003.

[19] S. Fitzgerald, I. Foster, C. Kesselman, G. von Laszewski, W. Smith, S. Tuecke: *A Directory Service for Configuring High-Performance Distributed Computations*, Proc. 6th IEEE Symposium on High-Performance Distributed Computing, pp. 365-375, 1997.

[20] The OpenPBS Project: *http://www.openpbs.org*.

[21] The Condor Project: *http://www.cs.wisc.edu/condor/*.

[22] J. Ousterhout, A. Cherenson, F. Douglis, M. Nelson, and B. Welch: *The Sprite network operating system*, IEEE Computer, 21(2):23–36, February 1988.

[23] S.J. Mullender, G. Van Rossum, A.S. Tanenbaum, R. Van Renesse, H. Van Staveren: *Amoeba: A distributed operating system for the 1990s*, IEEE Computer, 14:365–368, May 1990.