DotGrid: A .NET-based Cross-Platform Grid Computing Infrastructure

Alireza Poshtkuhi[1] Ali Haj Abutalebi[1,2] Leila Mahmoudi Ayough[2] Shaahin Hessabi[3]
[1] Islamic Azad University – Qazvin Branch
[2] Microelectronics Research & Design Center of IRAN
[3] Dept of Computer Engineering, Sharif University of Technology
dotgrid@gmail.com

Abstract— Recently, Grid infrastructures have provided wide integrated use of resources. DotGrid intends to introduce required Grid services and toolkits that are implemented as a layer wrapped over the existing operating systems. Our DotGrid has been developed based on Microsoft .NET in Windows and MONO .NET in Linux and UNIX. Using DotGrid APIs, Grid middlewares and applications can be implemented easily. We evaluated DotGrid capabilities by implementing some applications including a high throughput file transfer and solving a typical computational problem.

I. INTRODUCTION

GRID [1, 2] infrastructures provide the ability to share, select and aggregate distributed resources as computers, storage systems or other devices in an integrated way. Grid has solved many problems in science, engineering and commerce fields. This has come true via implementing various middlewares [3-8] for Grid. Middlewares are implemented as a layer over the existing operating systems (OS), making room for implementing Grid infrastructures. Studying existing middlewares unveils this fact that all of them use a set of common services that are needed for implementing a Grid. Regarding that these services have not been implemented at OS level, all middlewares have implemented them at scratch. Some efforts have been done for introducing an OS containing the required features for implementing a Grid [9-11], but no feasible structure meeting the Grid's demanding features has been proposed yet. In this paper we propose DotGrid infrastructure in pursue of filling up this gap. DotGrid tries to set up a cross-platform including Grid services and toolkits required for implementing Grid middlewares and applications. DotGrid is implemented as layer over the existing OS. This layered approach eliminates the dependency of Grid to the native OS. Hence, all middlewares and applications can be developed in such a Grid regardless of the type of the hardware and software over it, because DotGrid layer provides all needed services and system information needed for middlewares and Grid applications. Developing this layer leads us to implementing a homogenized system that is required for a Grid infrastructure. DotGrid proposes an implementation for Grid based on Microsoft .NET platform [12]. Although .NET is introduced in Windows OS but now it has become a multi-platform environment through some projects such as MONO .NET [13] that have made this platform available in other operating systems such as Linux, Solaris and other UNIX-based OSs. The other reason that biases .NET platform as our choice is that its rich library bypasses the need for implementing numerous required Grid services from scratch. Microsoft has also standardized .NET platform and its languages at ECMA [14, 15]. MONO .NET in Linux and Microsoft .NET in Windows together provides a cross-platform .NET where a wide range of machines regardless of their OS (Windows, Linux or UNIX) may build up a Grid. In comparison to other Grid middlewares, the cross-platform feature of DotGrid leads to true universal Grid infrastructure and bypasses incompatibility problems. DotGrid is implemented as a layer above the .NET platform. Figure 1 shows DotGrid structure.

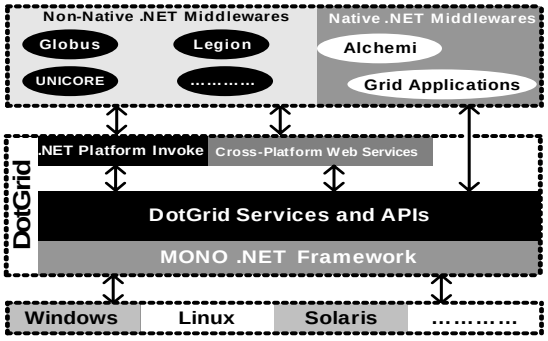

Fig. 1. DotGrid structure

Figure 2 shows current developed DotGrid services and APIs. Section II describes related works. Section III briefly describes DotGrid architecture and section IV presents some applications developed based on our DotGrid platform.

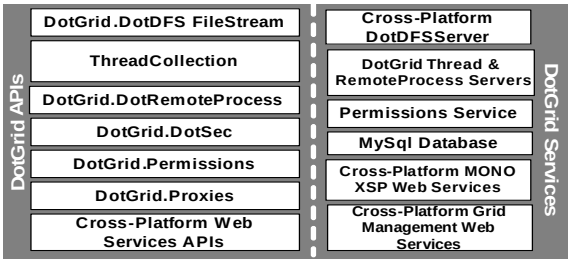

Fig. 2. Current developed DotGrid services and APIs

II. RELATED WORKS

Recently, some .NET-based Grid computing tools and middlewares have been created. Alchemi middleware at University of Melbourne is an open source software framework based on .NET that allows

to painlessly aggregate the computing power of networked Windows machines into a computational Grid [8]. WSRF.NET that has been created at University of Virginia is a .NET-based hosting environment for Grid services that implements Web Services Resource Framework (WSRF) and is supporter of the Open Grid Services Architecture (OGSA) on .NET Framework [16].

III. Architecture

DotGrid benefits from a layered and service-oriented architecture and provides the necessary services and toolkits that are required for building Grid, cluster and peer-to-peer computing environments. DotGrid services which are executed as servers (daemons) run in every operating system such as Windows, Linux and Solaris. This way, Grid or cluster middlewares can be developed easily in heterogeneous or homogeneous distributed environments. Based on DotGrid toolkit one can interface his/her systems to other existing Grid middlewares such as Globus [3] via .NET Platform Invoke and cross-platform web services. In the following sub-sections we describe some DotGrid components.

A. DotGrid.DotDFS Service

DotDFS is a Grid-based High-Throughput Distributed File System that is used as an infrastructural framework for resource sharing and files transferring in DotGrid platform. DotDFS proposes a set of open and cross-platform binary protocols. Most DotGrid services use DotDFS. Figure 3 shows layered model of DotDFS architecture.

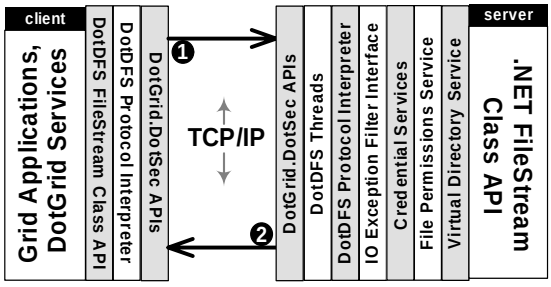

Fig. 3. DotDFS architecture in a peer-to-peer file transmission

1) DotDFS Protocol: .NET FileStream class provides appropriate tools for file manipulations. DotDFS protocol defines various headers and mechanisms based on methods and properties of this class that manage states of both client-server communications and data file transfers on networks. All data and headers are in binary format. DotDFS protocol has been defined considering all factors in order to achieve the best network bandwidth efficiency. Defined methods and properties in DotDFS protocol altogether develop a flexible infrastructure. Applications that require different file systems can be distributed conveniently on Grid computing environments using this infrastructure. This protocol supports both random and sequential access to distributed files. More features of DotDFS protocol are as follows:

2) Reliable transfer of data files: Reliable transfer of data files is one of the vital objectives of Grid applications in Grid computing. DotDFS protocol defines procedures that using them along with IO Exception Filter Interface layer shown in figure 3 and available API implementations at client-side make reliable and confidential data files transfers on the network possible. DotDFS protocol retransfers network packets that had been transferred in unreliable network situations to the opposite side by pre-defined policies.

3) Parallel file access: DotDFS Protocol allows parallel access to a single file system from unlimited numbers of DotDFS clients. .NET file lock can prevent collisions and ensures data consistency. Parallel file access accelerates processing of Grid applications that use thread parallelisms and share data, this way multiple copies of data is not needed and data storage space is reduced. This approach is suitable for some data-intensive Grid environments such as large file transfers with parallel data transfer mechanisms and high-energy physics applications.

4) Partial file transfers: Partial file transfer means that it is possible that Grid processes and applications tend to transfer only fragments of a file. Naturally, this concept is implemented in Read and Write methods of .NET FileStream class with any favorite offset to any length.

5) Stateless architecture: DotDFS protocol introduces a natural stateless architecture. This states that DotDFS servers don't keep track of DotDFS client requests according to which files have been opened, file positions and etc. This approach leads DotDFS clients can fail and resume without disturbing our system as whole and as a result this mechanism allows fast developments of Grid-based parallel cluster file systems, in future.

6) Transfer security modes: DotDFS protocol specifies three modes for transmitting data files on networks, secure, non-secure and semi-secure modes. In secure mode, all data are encrypted and then transferred on network, but in non-secure mode, only user credentials and file handles or file path names are encrypted according to DotGrid.DotSec protocol. For more flexibility, DotDFS protocol introduces semi-secure mode in which Grid developers can use secure and non-secure modes simultaneously in order to improve performance.

B. Unified DotSec Security Model

Since Virtual organizations share various resources in Grid environments, Grid computing development requires more security. DotSec has been designed based on DotDFS architectures for increasing performance and file transferring security. Besides, DotSec supports many available security protocols including SSL3, TLS1 and GSI and also defines some open protocols and procedures that all Grid

applications can use them based on their local policies independently. The major DotSec core consists of the following three skeletons.

Transmission Security Interface (TSI): TSI states some mechanisms through which, encrypted data can be received from socket interface and also pure data can be delivered to DotDFS or DotGridThread Protocol Interpreter module after decryption. At the same time, TSI is responsible to reserve data file integrities between two endpoints by using MD5 or SH1 hashing algorithm.

Symmetric Cryptography Interface (SCI): Symmetric encryption or private key cryptography provides ways to encrypt and decrypt data with a unique private key. Symmetric encryption algorithms also use IV (Initialization Vector). DotSec uses Rijndael and Triple-DES encryption algorithms.

Asymmetric Cryptography Interface (ACI): An asymmetric cryptographic algorithm, also known as public-key algorithms, requires a public and private key pair for data encryption and decryption. DotSec uses asymmetric encryption RSA algorithm for encryption and decryption of symmetric cryptographic keys such as Rijndael.

C. DotGrid.DotThreading Service

DotThreading service causes applications that want to use threading for parallelism and want to distribute threads in a network, execute their threads remotely in a Grid or Cluster environment and get the results. DotThreading service consists of one class API named ThreadCollection at the client side and a server named DotGridThreadServer. Figure 4 depicts mechanisms based on which a client runs a set of threads over a remote DotGridThreadServer session. In this figure, just the DotGrid services and APIs that are involved in this scenario are shown and the remaining and services and APIs are dropped.

1) ThreadCollection Client API: This class is responsible for creating and managing n remote threads on a remote thread server. Authentication is performed once for all of these n threads on the server. Creating a set of threads via this class is equivalent to a process that has been remotely executed. This client API has secure and non-secure modes for data transfer between clients and servers based on DotSec Model.

2) DotGridThreadServer Server: this server is responsible for receiving and executing client API requested threads. This server like DotDFS server is assigned to client authentication and management of client consumed system resources. To understand this server architecture, let's study the scenario shown in Figure 4. In this figure, ❶ a client located at "node i" contacts DotGridThreadServer located at "node j" by using ThreadCollection class, remotely. For contacting, client mentions items such as method delegates (similar to function pointers in C language) for a remote run, module dependencies and user credentials. After authenticating, serialized data of classes that are to execute remotely are transferred to "node j". ❷❸❹ At this time "node j" peer is connected to "node i" peer via DotDFS FileStream API and DotDFS downloads the files that have been declared by the client in "node i" as module dependencies to local hard disk of "node j". ❺ Then "node j" creates a .NET Application Domain and after configuring its related permissions, starts executing the client threads and finally results are returned back to client in an array of objects.

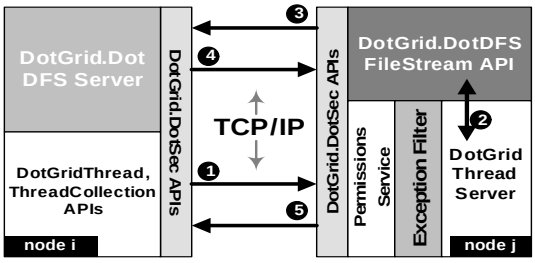

Fig. 4. DotThreading architecture and interaction between DotGrid services and its APIs in a peer-to-peer scenario

D. DotGrid Permissions Service

Management of complex, distributed, and dynamically changing job executions and resource usages are some central problems in Grid environments. All modern operating systems declare an elaborated set of permissions in order to handle users' level of access to system resources and enforcing local policies. Regarding that DotGrid is an infrastructure that intends to develop cross-platform Grid middlewares, it includes a service named FPS designed for managing users' distributed file system access level and enforcing Grid policies. DotGrid uses FPS for enforcing all processes to use DotDFS File Stream APIs and VDS (Virtual Directory Services). DotGrid uses .NET Code Access Security (CAS) model for investigating system resources access permissions during remote code runtime execution of services such as DotThreading and DotRemoteProcess. Based on CAS model, each application that uses .NET Common Language Runtime (CLR) for execution must interact with runtime's system security. At the time of an application execution, system security investigates the permission set declared by CAS. If that application is using a resource that is not permitted to by CAS, runtime's system security throws a security exception based on pre-defined DotGrid permission policies at that application domain's space. This way, that application will not be able to use that forbidden resource(s). These permissions have been set in application domain's space of the related Grid application before run. Enforcement of these CAS permissions has only been applied to native .NET applications that use DotThreading or DotRemoteProcess services. For none-native .NET job submissions based on DotRemoteProcess that execute Grid jobs on remote nodes on Linux or Windows OS, these mechanisms can not be applied, in these cases DotGrid can only prevent the native code execution before run. DotGrid defines two types of user accounts for enforcing permissions:

"Administrator" and "Others". There is no limit for "Administrator" account application execution for using system resources. For "Others" account, imperative permissions is applied. In figure 5 typical imperative permissions for an "Others" account is depicted. This information is stored in MySql database and is formatted in XML documents.

```
<?xml version="1.0" encoding="utf-8"?>
<permissions AccountType="Others">
    <UnmanagedCode value="True">
Ability to call unmanged code.
</UnmanagedCode>
    <SocketPermission value="False"/>
    <Execution value="False"/>
    <FileIOPermission value="False">
Controls the ability to access files and
folders.</FileIOPermission>
     <RegistryPermission value="False"/>
     <SqlClientPermission value="True"/>
</permissions>
```

Fig. 5. Sample XML-based DotGrid permissions representation

E. .NET Platform Invoke Service

One of the most effective powers of Microsoft .NET platform is its capability of interoperating with unmanaged code via Platform Invoke service relied on metadata to locate native exported functions and marshal their arguments at run time. .NET managed code calls dynamic link libraries like DLLs (.dll files) in Windows and shared objects (.so files) in Linux through P-Invoke. For example some managed Microsoft .NET and MONO .NET class library APIs are implemented by direct calling native Windows (Win32 or Win64) and Linux kernel APIs through this mechanism. The C# language function declaration shown in Fig 6 would invoke the POSIX **getpid()** system call on platforms that have the "libc.so" library. Hence Grid middlewares and applications developed based on DotGrid may use APIs and services of other native middlewares such as Globus developed based on standard C and C++ compilers.

```
using System;
// for DllImport
using System.Runtime.InteropServices;

[DllImport ("libc.so")]
private static extern int getpid();
```

Fig. 6. Interoperating with a native POSIX function from DotGrid through .NET Platform Invoke and C# language on Linux OS

F. DotGrid.DotRemoteProcess Service

DotRemoteProcess extends .NET Process class and adds remote process creation and management capabilities. One of DotRemoteProcess's features is providing mechanisms for remote job submissions based on the nature of the process for running on a special operating system (native .NET jobs or non-native jobs). In this case DotRemoteProcess creates and manages a remote process and DotDFS transfers input, output and application program files between two nodes.

G. DotGrid Summery

Such an infrastructure stated in the above sections causes forming of the architecture of networks with different deployment topologies like master-slaves, hierarchical and complete graph (currently, the complete graph topology is realized with DotDFS service that provides resource sharing in DotGrid platform between all Grid nodes). These topologies are depicted in figure 7. Furthermore, DotGrid intrinsically supports development of applications that are based on SPMD (Single Program, Multiple Data) and MPMD (Multiple Program, Multiple Data) architectures with its remote Grid threading and process model that are DotThreading and DotRemoteProcess services.

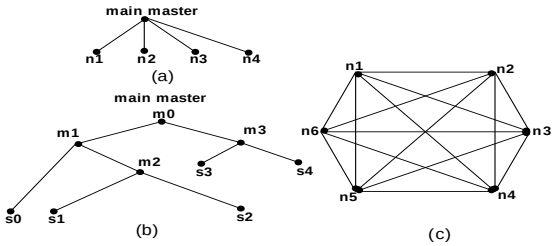

Fig . 7. Some typical DotGrid deployment topologies: (a) the master-slaves (b) the hierarchical (c) the complete graph, model

IV. Performance Evaluations

In order to prove the concept of implementation of a high performance cross-platform Grid middleware that is based on DotGrid platform, a middleware based on master-slaves architecture was designed. This middleware has two operating modes. These architectures are depicted in figures 8 and 9. As it is shown in these pictures, Distributor Node (DN) is responsible for scheduling, resource management and job submission based on DotGrid services. DN collects statistical information such as available memory, CPU usage and a list of running Grid tasks from all existing nodes in the cluster through DotRemoteProcess service and uses them for appropriate distribution of Grid tasks in pursue of getting best performance of computational time. A Distributor Node has two operating modes that provide strategical tools for deployment of any kind of Grid service in collaborative virtual organizations, high performance grid and cluster environments.

Direct Mode (DM): In this mode, after a client requests DN for submitting Grid tasks via DotGrid.DotThreading or DotRemoteProcess service, DN gives the client a unique GUID along with IP address of nodes with sufficient resources in the cluster. DN does this after client authentication. This GUID along with other defined permissions for this client via DotGrid Permissions service whose information is saved in MySQL Database will be distributed among all nodes by DN. This is when client APIs begin to distribute Grid tasks among Grid nodes peer-to-peer through TCP/IP sessions and using this GUID for single sign-on mechanisms. In fact, in

this operating mode, if clients connect to Grid nodes directly bypassing DN, a great enhancement will occur in system's performance and throughput, because if DN receives several requests and regarding that DN manages and distributes all Grid tasks, not only DN would turn into a bottleneck but also all Grid tasks related data files should be transmitted once from client to DN and once again from DN to all nodes associated in Grid.

Transparent Proxy Mode (TPM): This mode resembles to architecture of Alchemi [8] middleware. In this mode, DN itself starts to distribute tasks among Grid nodes after it receives client requests for performing Grid jobs and then returns the results to the client. The major difference between Alchemi and this operating mode is that DN behaves like a real proxy in performing "Rely" for transmitting files from clients to Grid nodes (In this operating mode all DotDFS related requests are relied and redirected transparently to Grid nodes via DN). Just like DM, TPM can be appropriate for jobs with large files, because in this mode files can be saved in hard disks of Grid nodes directly and there is no need to save them in distributor node temporarily.

Direct Mode provides lower security for Grid nodes because clients know the IP address of some nodes associated in the Grid and interact with them directly. But this mode can be used in private LAN networks for incrementing performance and system throughput regarding that all transactions are peer-to-peer and DN plays a minor role. Transparent Proxy Mode provides high security and all Grid jobs are distributed among Grid nodes via DN. This mode can be used in all LAN, MAN, WAN and Internet networks for DotGrid clusters secure connection around the world.

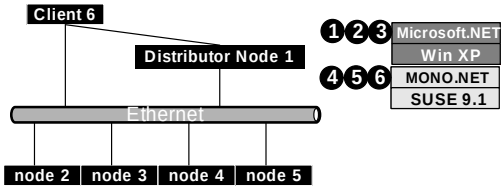

Fig. 8. Heterogeneous DotGrid Direct Mode deployment

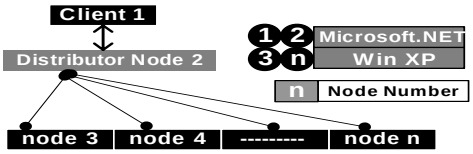

Fig. 9. Homogenous DotGrid Transparent Proxy Mode deployment

1) High Throughput File Transmission: A typical testbed application similar to Globus GridFTP [17] has been developed for testing file transfer capability of DotDFS services. DotDFS protocol is equipped with binary format headers and it doesn't use RPC or .NET Remoting for file transfers. Therefore, high throughput file transfers are expected. Interesting results are achieved in this phase by four experiments. Our tests consisted of two parallel streams of files in non-secure and secure modes and two single streams in non-secure and secure modes. In all experiments, we transferred files of size 1, 10, 100 and 1000 Mbytes in a LAN network with a bottleneck link of 1 Gbit/s between a DotDFS client and server. Both client and server machines had a dual processor 1.4 GHz Pentium, 512 MB memory and 80 GB hard disk. Client and server operating systems were Microsoft Windows Server 2003 and Microsoft Windows XP, respectively. Our tested Windows file system was NTFS without compression enabled. In parallel tests DotDFS service transferred mentioned files via 10 concurrent connections. In single stream tests, we transferred the same files in non-secure and secure modes without thread parallelism. We computed needed length of bytes for DotDFS Protocol headers while transferring a 1000 Mbytes file and we came to 64KB of network stream buffer size. Figures 10 and 11 depict the average performance of 20 test runs for each file transfer. These obtained results demonstrate that DotDFS service is a high throughput file transfer protocol especially for large files and DotGrid can be used in some Grid applications where petabytes of data are generated.

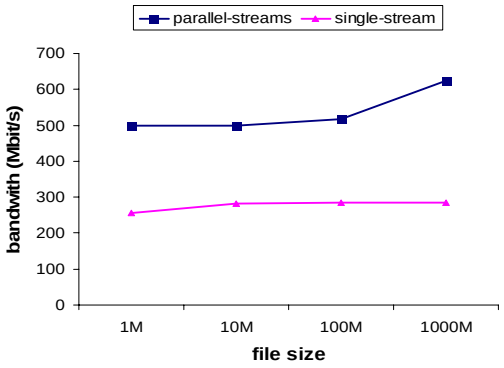

Fig. 10. DotDFS throughput in non-secure mode

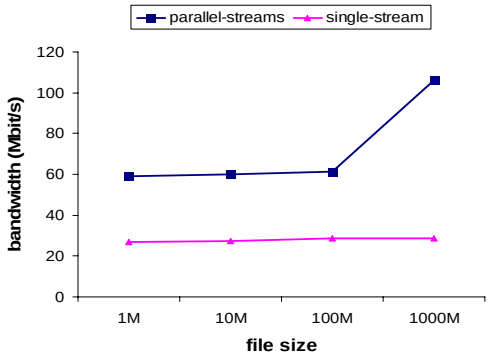

Fig. 11. DotDFS throughput in secure mode

2) A Computational Problem: For evaluating the computing capacity of DotGrid, we implemented a typical application that is based on DotGrid Direct Mode architecture. The test architecture is shown in figure 8. This testbed is a heterogeneous Grid environment containing 6 nodes in a LAN network with a bottleneck link of 100 Mbit/s. All machines have dual processors 1.5 GHZ Pentium, 1 GB RAM and 80 GB IDE Hard Disk. The operating systems are both Windows XP with Microsoft Framework 1.1 [12] and SUSE Linux 9.1 with MONO .NET Framework 1.1.8 [13]. DotDFS uses NTFS and ReiserFS file systems in Windows and Linux

respectively. In this typical application π number is calculated up to n digits [18] utilizing SPMD (Single Program, Multiple Data) DotGrid model based on DotThreading service stated in section III.C. The Distributor Node is responsible for managing the cluster and the client node is responsible for distributing threads over the four remaining nodes. Figure 12 depicts the results. Obtained results demonstrate DotGrid large calculating capacity. The more interesting subject was the interoperability between Microsoft .NET and MONO .NET Framework. We didn't see any problem in interoperability of these two separate implemented frameworks in Windows and Linux in the runtime execution. We compiled all DotGrid source codes once with Microsoft C# compiler and once again with MONO C# compiler. These results demonstrate that DotGrid can be used in solving large-scale computational problems where huge number of calculations is demanded in a real heterogeneous Grid environment homogenized with DotGrid platform.

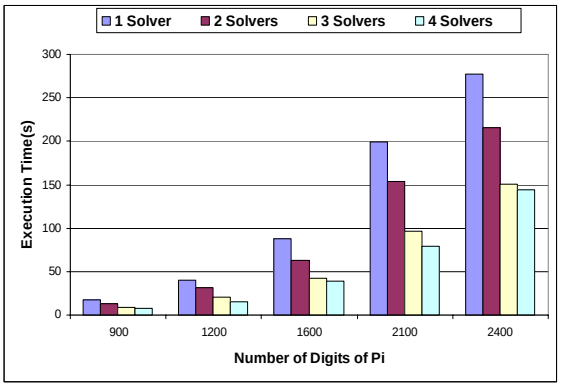

Fig. 12. Computation time for different number of digits of Pi with different number of nodes

V. Conclusion and Future Work

Our initial endeavors were to construct a flexible and cross-platform environment that is available in all operating systems. We are going to port DotGrid to Win32, Win64 and 32 and 64 bit versions of Linux kernels. We expect that this approach can link .NET-based DotGrid and future versions of native OS-based DotGrids via .NET Platform Invoke and open DotSec & DotDFS protocols. Furthermore; porting to native codes and modern 64 bit computing technologies based on AMD and Intel 64 bit processors, will contribute to develop efficient and high performance Grid applications based on DotGrid platform. DotGrid will homogenize and unify the heterogeneous nature of distributed and grid computing environments.

VI. References

[1] I. Foster, C. Kesselman, S. Tuecke, "The Anatomy of the Grid Enabling Scalable Virtual Organizations", *International J. Supercomputer Applications*, 15(3), 2001.

[2] I. Foster, C. Kesselman, J. Nick, 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.

[3] I. Foster, "Globus Toolkit Version 4: Software for Service-Oriented Systems", *IFIP International Conference on Network and Parallel Computing*, 2005.

[4] D. Thain, T. Tannenbaum, M. Livny, "Condor and the Grid", in Fran Berman, Anthony J.G. Hey, Geoffrey Fox, editors, *Grid Computing: Making The Global Infrastructure a Reality, John Wiley*, 2003. ISBN: 0-470-85319-0.

[5] A. S. Grimshaw, W. A. Wulf, and the Legion team, "The legion vision of a worldwide virtual computer", *Communications of the ACM*, 1997.

[6] V. Huber, "UNICORE: A Grid computing environment for distributed and parallel computing", Proceedings PaCT, Springer Verlag.

[7] R. Buyya, S. Venugopal, "The Gridbus Toolkit for Service Oriented Grid and Utility Computing: An Overview and Status Report", *Proceedings of the First IEEE International Workshop on Grid Economics and Business Models*.

[8] A. Luther, R. Buyya, R. Ranjan, S. Venugopal, "Alchemi: A. NET-Based Enterprise Grid Computing System", *Proceedings of the 6th International Conference on Internet Computing (ICOMP'05)*, 2005, USA.

[9] P. Padala, J. N Wilson, "GridOS: Operating System Services for Grid Architectures", *In Proceedings of International Conference On High Performance Computing (HiPC'03)*, 2003.

[10] D. Talia, "Towards GRID Operating Systems: from GLinux to a GVM", *Proc. Workshop on Network Centric Operating Systems*, Brussels, 16-17 march 2005.

[11] B. Matthews, A. Arenas, M. Wilson, D. Mac Randal, A. Svirskas, J. Gallop, J. Bicarregui, S. Lambert, "Towards a Knowledge Grid: Requirements for a GridOS to Support the Next Generation Grid", *Proc. COreGRID workshop on the Next Generation Grid*, Belgium, 2005.

[12] Microsoft Corporation, .NET Framework Home Page, *http://msdn.microsoft.com/netframework/*

[13] MONO .NET Project Home Page, *http://www.mono-project.com/*

[14] ECMA-334: C# Language Specification, *http://www.ecma-international.org/publications/standards/Ecma-334.htm*

[15] ECMA-335: Common Language Infrastructure (CLI),*http://www.ecma-international.org/ publications/techreports/E-TR 084.htm*

[16] Grid Computing Group at UVa, *http://www.cs.virginia.edu/~humphrey/GCG.htm*

[17] B. Allcock, J. Bresnahan, R. Kettimuthu, M. Link, C. Dumitrescu, I. Raicu, I. Foster, "The Globus Striped GridFTP Framework and Server", *High Performance Distributed Computing Conference (HPDC 14)*, 2005.

[18] F. Bellard, "Computation of the n'th digit of pi in any base in O(n^2)", *http://fabrice.bellard.free.fr/pi/pi_n2/pi_n2.html*