A .NET-based Enterprise Grid Framework

Faculty Advisor

Dr. Mashiur Rahman
Assistant Professor
Dept. of EECS
North South University.

Prepared By

Mufid Islam  Ashfaqul Amin Antu
Id# 052-023-045  Id# 061-569-045
Abstract

Grid computing is the application of several computers to a single problem at the same time — usually to a scientific or technical problem that requires a great number of computer processing cycles or access to large amounts of data. Computational grids that couple geographically distributed resources are becoming the effective computing platform for solving large-scale problems in science, engineering, and commerce. Softwares to enable grid computing has been primarily written for Unix-class operating system, thus severely limiting the ability to effectively utilize the computing resources of the vast majority of desktop computers i.e. those running variants under the Microsoft Windows operating system. Here Alchemi has been used as a Windows based grid framework and runtime machinery to create a high-throughput computing environment by harnessing distributed resources where researcher of different areas can submit and process their jobs. Along with that an end user application has been developed where users can submit and manage their jobs easily without concerning the complexity of the system.
Chapter 1

Introduction
1.1 BACKGROUND

The idea of metacomputing is very promising as it enables the use of a network of many independent computers as if they were one large parallel machine, or virtual supercomputer for solving large-scale problems in science, engineering, and commerce. With the exponential growth of global computer ownership, local networks and Internet connectivity, this concept has been taken to a new level – grid computing. The idea is to build a powerful and cost effective metacomputing infrastructure without using high performance servers and ensure large scale adoption from it. So, Microsoft Windows based grid computing infrastructure will play the best role in the industry-wide adoption of grids due to the large-scale deployment of Windows within the enterprises. There are a number of computers in university LABs and dorms, which remains idle most of the time. Under a Grid environment idle CPU cycles can be effectively used to achieve a powerful computational system that can be used for research and educational purpose. In future, this local Grid infrastructure can be integrated with global grid resources through connectivity with Bangladesh Research and Education Network (BdREN).

1.2 DEVELOPMENT

The following things has been developed –

- Develop and install a windows based Grid system which is adapted with local infrastructures.

- A GRID application where end users can
- Submit and manage the execution of their jobs without concerning about the complexity or platform
- Share resources from all the computers that is connected with the grid
- Add new services
- Have detail resource information
  • Run applications

1.3 MOTIVATION

Computer researchers are looking to use idle computational cycles to achieve a powerful computational system at lower cost. On the other hand commercial companies want to save their money by utilizing their idle computers instead of buying expensive supercomputers. Grids are a form of distributed computing whereby a “super virtual computer” is composed of many networked loosely coupled computers acting in concert to perform very large tasks. There are number of computers in the LABs and offices, which remain idle time to time. A grid network can be created where idle CPU cycles can be effectively used to achieve a powerful computational system that can be used for business, research and educational purpose. Thus creating a grid lab can be very beneficial for research projects, implementation of grid scheduling algorithms and high performance computation.

1.3.1 NETWORK SPEED Vs. PROCESSOR SPEED

Gordon Moore (co-founder of Intel) predicted in 1965 that the transistor density of semiconductor chips would double roughly every 18 months. Moore's law describes a long-term trend in the history of computing hardware. Since the invention of the integrated circuit in 1958, the number of transistors that can be placed inexpensively on an integrated circuit has increased exponentially, doubling approximately every two years. The trend was first observed by Intel co-founder Gordon E. Moore in a 1965 paper. It has continued for almost half a century and in 2005 was not expected to stop for another decade at least.
Gilder's Law says that Bandwidth grows at least three times faster than computer power. While computer power doubles every eighteen months (Moore's law), communications power doubles every six months. This is a rough average; as with Moore's law, it may take a few years before the pace reaches its natural rate. Practical backbone bandwidth on a single cable is now a thousand times greater than the entire average traffic on the global communications infrastructure five years ago. More information can be sent over a single cable in a second than was sent over the entire Internet in 1997 in a month. These pictures analyze the above two laws-

![Gilder's Law vs. Moore's Law](image)

Figure 1.1: Gilder’s Law vs. Moore’s Law
Analyzing the graphs, it is obvious that the network speed increases faster than the processor speed. So, A Grid designed for high speed network is enabling to provide outstanding computational power where expensive resources are not really very important.

1.3.2 BdREN (Bangladesh Research and Education Network)

The BdREN (Bangladesh Research and Education Network) is basically a high performance data communication ICT network that would be owned and operated by the universities of Bangladesh. The BdREN network naturally will be an ICT based system having its own campus network (Intranet) connected to other campus networks, to national fiber backbone (Internet) and to other RENs globally, with its own routers, switches, servers, radio links and most important, its own fiber.
1.3.3 TEIN3 (Third generation of the Trans-Eurasia Information Network)

The third generation of the Trans-Eurasia Information Network (TEIN3) provides a dedicated high-capacity Internet network for research and education communities across Asia-Pacific. It currently connects 11 countries in the region, and provides direct connectivity to Europe’s GÉANT2 network. TEIN3 offers Asia-Pacific a gateway for global collaboration, enabling over 30 million users across the region to participate in joint projects with their peers in Europe and other parts of the world.

1.3.4 GRID in NSU perspective & Bangladesh

In Bangladesh, the opportunity of using any high performance computational system to expedite the Grid research is limited. If it is possible to develop a Grid system and share resources of different universities, it might be possible to achieve highly powerful computation system in a cost effective way. It also enables knowledge and resource sharing which helps everyone to go forward. This system will use the connectivity of BdREN, which will connect with the global REN networks through the TEIN 3 (Trans-Eurasia Information Network) of the European Union (EU) and the Asia- Pacific Network (APAN) and INTERNET 2 of USA. It will also connect with the Bangladesh National Internet Exchange (BDIX).
Chapter 2

GRID COMPUTING CONCEPTS
2.1 GRID COMPUTING

Grid computing, most simply stated, is distributed computing taken to the next evolutionary level. The goal is to create the illusion of a simple yet large and powerful self managing virtual computer out of a large collection of connected heterogeneous systems sharing various combinations of resources. The standardization of communications between heterogeneous systems created the Internet explosion. The emerging standardization for sharing resources, along with the availability of higher bandwidth, are driving a possibly equally large evolutionary step in grid computing.

2.2 GRIDS VERSUS CONVENTIONAL SUPERCOMPUTERS

“Distributed” or “grid” computing in general is a special type of parallel computing that relies on complete computers (with onboard CPU, storage, power supply, network interface, etc.) connected to a network (private, public or the Internet) by a conventional network interface, such as Ethernet. This is in contrast to the traditional notion of a supercomputer, which has many processors connected by a local high-speed computer bus.

The primary advantage of distributed computing is that each node can be purchased as commodity hardware, which when combined can produce similar computing resources to a multiprocessor supercomputer, but at lower cost. This is due to the economies of scale of producing commodity hardware, compared to the lower efficiency of designing and
constructing a small number of custom supercomputers. The primary performance disadvantage is that the various processors and local storage areas do not have high-speed connections. This arrangement is thus well suited to applications in which multiple parallel computations can take place independently, without the need to communicate intermediate results between processors. The high-end scalability of geographically dispersed grids is generally favourable, due to the low need for connectivity between nodes relative to the capacity of the public Internet.

There are also some differences in programming and deployment. It can be costly and difficult to write programs so that they can be run in the environment of a supercomputer, which may have a custom operating system, or require the program to address concurrency issues. If a problem can be adequately parallelized, a “thin” layer of “grid” infrastructure can allow conventional, standalone programs to run on multiple machines (but each given a different part of the same problem). This makes it possible to write and debug on a single conventional machine, and eliminates complications due to multiple instances of the same program running in the same shared memory and storage space at the same time.

2.3 SCOPES OF GRID COMPUTING

2.3.1 EXPLOITING UNDERUTILIZED RESOURCES

The easiest use of grid computing is to run an existing application on a different machine. The machine on which the application is normally run might be unusually busy due to an unusual peak in activity. The job in question could be run on an idle machine elsewhere on the grid. There are at least two prerequisites for this scenario. First, the application must be executable remotely and without undue overhead. Second, the remote machine must meet any special hardware, software, or resource requirements imposed by the application. In most organizations, there are large amounts of underutilized computing resources. Most desktop
machines are busy less than 5 percent of the time. In some organizations, even the server machines can often be relatively idle. Grid computing provides a framework for exploiting these underutilized resources and thus has the possibility of substantially increasing the efficiency of resource usage. Another function of the grid is to better balance resource utilization. An organization may have occasional unexpected peaks of activity that demand more resources. If the applications are grid-enabled, they can be moved to underutilized machines during such peaks. In fact, some grid implementations can migrate partially completed jobs. In general, a grid can provide a consistent way to balance the loads on a wider federation of resources. This applies to CPU, storage, and many other kinds of resources that may be available on a grid. Management can use a grid to better view the usage patterns in the larger organization, permitting better planning when upgrading systems, increasing capacity, or retiring computing resources no longer needed.

2.3.2 PARALLEL CPU CAPACITY

The potential for massive parallel CPU capacity is one of the most attractive features of a grid. In addition to pure scientific needs, such computing power is driving a new evolution in industries such as the bio-medical field, financial modelling, oil exploration, motion picture animation, and many others. The common attribute among such uses is that the applications have been written to use algorithms that can be partitioned into independently running parts. A CPU intensive grid application can be thought of as many smaller “sub-jobs,” each executing on a different machine in the grid. To the extent that these sub-jobs do not need to communicate with each other, the more “scalable” the application becomes. A perfectly scalable application will, for example, finish 10 times faster if it uses 10 times the number of processors. Barriers often exist to perfect scalability. The first barrier depends on the algorithms used for splitting the application among many CPUs. If the algorithm can only be
split into a limited number of independently running parts, then that forms a scalability barrier. The second barrier appears if the parts are not completely independent; this can cause contention, which can limit scalability.

For example, if all of the sub-jobs need to read and write from one common file or database, the access limits of that file or database will become the limiting factor in the application’s scalability. Other sources of inter-job contention in a parallel grid application include message communications latencies among the jobs, network communication capacities, synchronization protocols, input-output bandwidth to devices and storage devices, and latencies interfering with real-time requirements.

### 2.3.3 Virtual Resources and Virtual Organization for Collaboration

Another important grid computing contribution is to enable and simplify collaboration among a wider audience. In the past, distributed computing promised this collaboration and achieved it to some extent. Grid computing takes these capabilities to an even wider audience, while offering important standards that enable very heterogeneous systems to work together to form the image of a large virtual computing system offering a variety of virtual resources. The users of the grid can be organized dynamically into a number of virtual organizations, each with different policy requirements. These virtual organizations can share their resources collectively as a larger grid. Sharing starts with data in the form of files or databases. A “data grid” can expand data capabilities in several ways.
First, files or databases can seamlessly span many systems and thus have larger capacities than on any single system. Such spanning can improve data transfer rates through the use of striping techniques. Data can be duplicated throughout the grid to serve as a backup and can be hosted on or near the machines most likely to need the data, in conjunction with advanced scheduling techniques. Sharing is not limited to files, but also includes many other resources, such as equipment, software, services, licenses, and others. These resources are “virtualized” to give them a more uniform interoperability among heterogeneous grid participants. The participants and users of the grid can be members of several real and virtual organizations. The grid can help in enforcing security rules among them and implement policies, which can resolve priorities for both resources and users.

### 2.3.4 ACCESS TO ADDITIONAL RESOURCES

An addition to CPU and storage resources, a grid can provide access to increased quantities of other resources and to special equipment, software, licenses, and other services. The additional resources can be provided in additional numbers and/or capacity.

For example, if a user needs to increase his total bandwidth to the Internet to implement a data mining search engine, the work can be split among grid machines that have independent...
connections to the Internet. In this way, the total searching capability is multiplied, since each machine has a separate connection to the Internet. If the machines had shared the connection to the Internet, there would not have been an effective increase in bandwidth. Some machines may have expensive licensed software installed that the user requires. His jobs can be sent to such machines more fully exploiting the software licenses. Some machines on the grid may have special devices. Most of us have used remote printers, perhaps with advanced colour capabilities or faster speeds.

### 2.3.5 RESOURCE BALANCING

A grid federates a large number of resources contributed by individual machines into a greater total virtual resource. For applications that are grid-enabled, the grid can offer a resource balancing effect by scheduling grid jobs on machines with low utilization. This feature can prove invaluable for handling occasional peak loads of activity in parts of a larger organization.

This can happen in two ways:

- An unexpected peak can be routed to relatively idle machines in the grid.
- If the grid is already fully utilized, the lowest priority work being performed on the grid can be temporarily suspended or even cancelled and performed again later to make room for the higher priority work.

Without a grid infrastructure, such balancing decisions are difficult to prioritize and execute. Occasionally, a project may suddenly rise in importance with a specific deadline. A grid cannot perform a miracle and achieve a deadline when it is already too close. However, if the size of the job is known, if it is a kind of job that can be sufficiently split into sub-jobs, and if
enough resources are available after pre-empting lower priority work, a grid can bring a very large amount of processing power to solve the problem. In such situations, a grid can, with some planning, succeed in meeting a surprise deadline.

2.4 GRID CONCEPTS AND COMPONENTS

A grid is a collection of machines, sometimes referred to as “nodes,” “resources,” “members,” “donors,” “clients,” “hosts,” “engines,” and many other such terms. They all contribute any combination of resources to the grid as a whole. Some resources may be used by all users of the grid while others may have specific restrictions.

2.4.1 SCHEDULING, RESERVATION, AND SCAVENGING

The grid system is responsible for sending a job to a given machine to be executed. In the simplest of grid systems, the user may select a machine suitable for running his job and then execute a grid command that sends the job to the selected machine. More advanced grid systems would include a job “scheduler” of some kind that automatically finds the most appropriate machine on which to run any given job that is waiting to be executed. Schedulers react to current availability of resources on the grid. The term “scheduling” is not to be confused with “reservation” of resources in advance to improve the quality of service. Sometimes the term “resource broker” is used in place of “scheduler,” but this term implies that some sort of bartering capability is factored into scheduling. In a “scavenging” grid system, any machine that becomes idle would typically report its idle status to the grid management node. This management node would assign to this idle machine the next job which is satisfied by the machine’s resources. Scavenging is usually implemented in a way that is unobtrusive to the normal machine user. If the machine becomes busy with local non-
grid work, the grid job is usually suspended or delayed. This situation creates somewhat unpredictable completion times for grid jobs, although it is not disruptive to those machines donating resources to the grid.

To create more predictable behaviour, grid machines are often “dedicated” to the grid and are not pre-empted by outside work. This enables schedulers to compute the approximate completion time for a set of jobs, when their running characteristics are known. As a further step, grid resources can be “reserved” in advance for a designated set of jobs. Such reservations operate much like a calendaring system used to reserve conference rooms for meetings. This is done to meet deadlines and guarantee quality of service. When policies permit, resources reserved in advance could also be scavenged to run lower priority jobs when they are not busy during a reservation period, yielding to jobs for which they are reserved.

Thus, various combinations of scheduling, reservation, and scavenging can be used to more completely utilize the grid. Scheduling and reservation is fairly straightforward when only one resource type, usually CPU, is involved. However, additional grid optimizations can be achieved by considering more resources in the scheduling and reservation process. For example, it would be desirable to assign executing jobs to machines nearest to the data that these jobs require. This would reduce network traffic and possibly reduce scalability limits. Optimal scheduling, considering multiple resources, is a difficult mathematics problem. Therefore, such schedulers may use heuristics.
These heuristics are rules that are designed to improve the probability of finding the best combination of job schedules and reservations to optimize throughput or any other metric.

2.4.2 INTRAGRID TO INTERGRID

There have been attempts to formulate a precise definition for what a “grid” is. In fact, the concept of grid computing is still evolving and most attempts to define it precisely end up excluding implementations that many would consider to be grids. We will be pragmatic and not claim to make any definitive descriptions of what a grid is and is not. Therefore, the following descriptions of various kinds of “grids” must be taken loosely. Grids can be built in all sizes, ranging from just a few machines in a department to groups of machines organized as a hierarchy spanning the world. In this section, we will describe some examples in this range of grid system topologies.
As presented in Figure, the simplest grid consists of just a few machines, all of the same hardware architecture and same operating system, connected on a local network. This kind of grid uses homogeneous systems so there are fewer considerations and may be used just for experimenting with grid software. The machines are usually in one department of an organization, and their use as a grid may not require any special policies or security concerns. Because the machines have the same architecture and operating system, choosing application software for these machines is usually simple. Some people would call this a “cluster” implementation rather than a “grid.”

2.5 WORLDWIDE PROJECTS AND APPLICATIONS
Grids offer a way to solve Grand Challenge problems such as protein folding, financial modelling, earthquake simulation, and climate/weather modelling. Grids offer a way of using the information technology resources optimally inside an organization. They also provide a means for offering information technology as a utility for commercial and non-commercial clients, with those clients paying only for what they use, as with electricity or water.

Grid computing is being applied by the National Science Foundation's National Technology Grid, NASA's Information Power Grid, Pratt & Whitney, Bristol-Myers Squibb Co., and American Express. One of the most famous cycle-scavenging networks is SETI@home, which was using more than 3 million computers to achieve 23.37 sustained teraflops (979 lifetime teraflops) as of September 2001.

As of March 2008, Folding@home had achieved peaks of 1,502 teraflops on over 270,000 machines. The European Union has been a major proponent of Grid computing. Many projects have been funded through the framework program of the European Commission. Many of the projects are highlighted below, but two deserve special mention: BEinGRID and Enabling Grids for E-sciencE.

BEinGRID (Business Experiments in Grid) is a research project partly funded by the as an Integrated Project under the Sixth Framework Program (FP6) sponsorship program. Started in June 1 2006, the project will run 42 months, until November 2009. The project is coordinated by Atos Origin. According to the project factsheet, their mission is “to establish effective routes to foster the adoption of Grid Computing across the EU and to stimulate research into innovative business models using Grid technologies”.

To extract best practice and common themes from the experimental implementations, two groups of consultants are analyzing a series of pilots, one technical and one business. The results of these cross analyses are provided by the website Gridipedia. The project is significant not only for its long duration, but also for its budget, which at 24.8 million Euros, is the largest of any FP6 integrated project. Of this, 15.7 million is provided by the European commission and the remainder by its 98 contributing partner companies.

The Enabling Grids for E-science project, which is based in the European Union and includes sites in Asia and the United States, is a follow-up project to the European Data Grid (EDG) and is arguably the largest computing grid on the planet. This, along with the LHC Computing Grid (LCG), has been developed to support the experiments using the CERN Large Hadron Collider. The LCG project is driven by CERN's need to handle huge amounts of data, where storage rates of several gigabytes per second (10 petabytes per year) are required. A list of active sites participating within LCG can be found online as can real time monitoring of the EGEE infrastructure. The relevant software and documentation is also publicly accessible. Another well-known project is distributed.net, which was started in 1997 and has run a number of successful projects in its history.

The NASA Advanced Supercomputing facility (NAS) has run genetic algorithms using the Condor cycle scavenger running on about 350 Sun and SGI workstations.

Until April 27, 2007, United Devices operated the United Devices Cancer Research Project based on its Grid MP product, which cycle-scavenges on volunteer PCs connected to the Internet. As of June 2005, the Grid MP ran on about 3.1 million machines.
Another well known project is the World Community Grid. The World Community Grid's mission is to create the largest public computing grid that benefits humanity. This work is built on the belief that technological innovation combined with visionary scientific research and large scale volunteerism can change our world for the better. IBM Corporation has donated the hardware, software, technical services, and expertise to build the infrastructure for World Community Grid and provides free hosting, maintenance
Chapter 3

Alchemi – A .NET Grid Computing Framework
Alchemi is an open source .NET-based enterprise grid computing framework. It allows the user to combine the computing power of networked machines into a virtual supercomputer and develop applications to run on the grid with no additional investment. It has been designed to be easy to use without sacrificing power and flexibility. It supports the Microsoft Windows operating system which is seen as a key factor in industry adoption of grid computing technology. Alchemi.NET includes:

- The runtime machinery (Windows executables) to construct grids.
- A .NET API and tools to develop .NET grid applications and grid-enabled legacy applications.

There can be many reasons for choosing Alchemi.NET as the framework for building the computational grid and then further choosing to build a fault tolerant system for it. Some of these reasons are:

- Alchemi.NET is the first .NET based stable grid.
- Most important is that Alchemi.NET is open source. So one can do any number of changes in it.
- Another important reason is that most of the systems in our labs are running Windows Operating Systems.
- Fault tolerance research area is still under development in Alchemi.NET.
- Checkpointing is also not available in Alchemi.NET.

3.2 .NET Framework

The Microsoft .NET Framework is a software framework that can be installed on computers running Microsoft Windows operating systems. It includes a large library of coded solutions to common programming problems and a virtual machine that manages the
execution of programs written specifically for the framework. The .NET Framework is a Microsoft offering and is intended to be used by most new applications created for the Windows platform.

The .NET Framework is:

- Common Language Runtime – provides an abstraction layer over the operating system
- Base Class Libraries – pre-built code for common low-level programming tasks
- Development frameworks and technologies – reusable, customizable solutions for larger programming tasks

The .NET Framework allows to:

- Apply common skills across a variety of devices, application types, and programming tasks
- Integrate with other tools and technologies to build the right solution with less work
- Build compelling applications faster

3.3 Why .NET Framework?

Most of the machines worldwide run variants of Microsoft Windows operating system. Hence designing for Windows is seen as key factor in industry adoption of grid computing technology. Softwares to enable grid computing has been primarily written for Unix-class operating system. Most existing systems are also Unix-based / optimized for UNIX. Many features of the new .NET platform can be leveraged. This framework supports multiple
languages like writing API/libraries once in any .NET language and make use from any other .NET supported language.
Chapter 4

Alchemi Architecture
4.1 Architecture

Alchemi follows the master-worker parallel programming paradigm in which a central component dispatches independent units of parallel execution to workers and manages them. This smallest unit of parallel execution is a grid thread, which is conceptually and programmatically similar to a thread object (in the object-oriented sense) that wraps a "normal" multitasking operating system thread. A grid application is defined simply as an application that is to be executed on a grid and that consists of a number of grid threads. Grid applications and grid threads are exposed to the grid application developer via the object oriented Alchemi .NET API.

Alchemi offers four distributed components designed to operate under three usage patterns. They are discussed here with respect to their basic operation.

1. Manager
2. Executor
3. Owner
4. Cross-Platform Manager
1. Manager

The Manager manages the execution of grid applications and provides services associated with managing thread execution. The Executors register themselves with the Manager which in turn keeps track of their availability. Threads received from the Owner are placed in a pool and scheduled to be executed on the various available Executors. A priority for each thread can be explicitly specified when it is created within the Owner, but is assigned the highest priority by default if none is specified. Threads are scheduled on a Priority and First Come First Served (FCFS) basis, in that order. The Executors return completed threads to the Manager which are subsequently passed on or collected by the respective Owner.
2. Executor

The Executor accepts threads from the Manager and executes them. An Executor can be configured to be dedicated, meaning the resource is centrally managed by the Manager, or non-dedicated, meaning that the resource is managed on a volunteer basis via a screen saver or by the user. For non-dedicated execution, there is one-way communication between the Executor and the Manager. In this case, the resource that the Executor resides on is managed.
on a volunteer basis since it requests threads to execute from the Manager. Where two-way communication is possible and dedicated execution is desired the Executor exposes an interface (IExecutor) so that the Manager may communicate with it directly. In this case, the Manager explicitly instructs the Executor to execute threads, resulting in centralized management of the resource where the Executor resides. Thus, Alchemi’s execution model provides the dual benefit of:

- flexible resource management i.e. centralized management with dedicated execution vs. decentralized management with non-dedicated execution; and
- flexible deployment under network constraints i.e. the component can be deployed as nondedicated where two-way communication is not desired or not possible (e.g. when it is behind a firewall or NAT/proxy server).

Thus, dedicated execution is more suitable where the Manager and Executor are on the same Local Area Network while non-dedicated execution is more appropriate when the Manager and Executor are to be connected over the Internet.
Figure 4.3: Executor connected to a Manager.

3. Owner

Grid applications created using the Alchemi API are executed on the Owner component. The Owner provides an interface with respect to grid applications between the application developer and the grid. Hence it “owns” the application and provides services associated with the ownership of an application and its constituent threads. The Owner submits threads to the
Manager and collects completed threads on behalf of the application developer via the Alchemi API.

4. Cross-Platform Manager

The Cross-Platform Manager, an optional sub-component of the Manager, is a generic web services interface that exposes a portion of the functionality of the Manager in order to enable Alchemi to manage the execution of platform independent grid jobs (as opposed to grid applications utilizing the Alchemi grid thread model). Jobs submitted to the Cross-Platform Manager are translated into a form that is accepted by the Manager (i.e. grid threads), which are then scheduled and executed as normal in the fashion described above. Thus, in addition to supporting the grid-enabling of existing applications, the Cross-Platform Manager enables other grid middleware to interoperate with and leverage Alchemi on any platform that supports web services (e.g. Gridbus Grid Service Broker).

4.2 System Configurations

The components discussed above allow Alchemi to be utilized to create different grid configurations: desktop cluster grid, multi-cluster grid, and cross-platform grid (global grid).

Cluster (Desktop Grid)

The basic deployment scenario – a cluster (shown in Figure 3) - consists of a single Manager and multiple Executors that are configured to connect to the Manager. One or more Owners can execute their applications on the cluster by connecting to the Manager. Such an environment is appropriate for deployment on Local Area Networks as well as the Internet. The operation of the Manager, Executor and Owner components in a cluster is as described above.
Multi-Cluster

A multi-cluster environment is created by connecting Managers in a hierarchical fashion (Figure 34a). As in a single-cluster environment, any number of Executors and Owners can connect to a Manager at any level in the hierarchy. An Executor and Owner in a multi-cluster environment connect to a Manager in the same fashion as in a cluster and correspondingly their operation is no different from that in a cluster. The key to accomplishing multi-clustering in Alchemi's architecture is the fact that a Manager behaves like an Executor towards another Manager since the Manager implements the interface of the Executor. A Manager at each level except for the topmost level in the hierarchy is configured to connect to a higher-level Manager as an “intermediate” Manager and is treated by the higher level-Manager as an Executor. Such an environment is more appropriate for deployment on the Internet.

![Diagram of cluster (desktop grid) deployment]

[b] Legend

Figure 4.4: Cluster (desktop grid) deployment.
The operation of an intermediate Manager in a multi-cluster environment therefore, must be discussed with respect to the behavior of an Executor and is as follows. Once Owners have submitted grid applications to their respective Managers, each Manager has “local” grid threads waiting to be executed. As discussed, threads are assigned the highest priority by default (unless the priority is explicitly specified during creation) and threads are scheduled and executed as normal by the Manager’s local Executors. Note that an ‘Executor’ in this context could actually be an intermediate Manager, since it is treated as an Executor by the higher-level Manager. In this case after receiving a thread from the higher-level Manager, it is scheduled locally by the intermediate Manager with a priority reduced by one unit and is executed as normal by the Manager’s local ‘Executors’ (again, any of which could be intermediate Managers).

In addition, at some point the situation may arise when a Manager wishes to allocate a thread to one of its local Executors (one or more of which could an intermediate Manager), but there are no local threads waiting to be executed. In this case, if the Manager is an intermediate
Manager, it requests a thread from its higher-level Manager, reduced the priority by one unit and schedules it locally. In both of these cases, the effect of the reduction in priority of a thread as it moves down the hierarchy of Managers is that the “closer” a thread is submitted to an Executor, the higher is the priority that it executes with. This allows a portion of an Alchemi grid that is within one administrative domain (i.e. a cluster or multi-cluster under a specific “administrative domain Manager”) to be shared with other organizations to create a collaborative grid environment without impacting on its utility to local users. As with an Executor, an intermediate Manager must be configured for either dedicated or non-dedicated execution. Not only does its operation in this respect mirror that of an Executor, the same benefits of flexible resource management and deployment under network constraints apply.

Cross-Platform Grid

The Cross-Platform Manager can be used to construct a grid conforming to the classical global grid model. A grid middleware component such as a broker can use the Cross-Platform Manager web service to execute cross-platform applications (jobs within tasks) on an Alchemi node (cluster or multi-cluster) as well as resources grid-enabled using other technologies such as Globus.

4.3 Design and Implementation

4.3.1 Overview

The .NET Framework offers two mechanisms for execution across application domains – Remoting and web services (application domains are the unit of isolation for a .NET application and can reside on different network hosts). .NET Remoting allows a .NET object to be “remoted” and expose its functionality across application domains. Remoting is used for communication between the four Alchemi distributed grid components as it allows low-level
interaction transparently between .NET objects with low overhead (remote objects are configured to use binary encoding for messaging). Web services were considered briefly for this purpose, but were decided against due to the relatively higher overheads involved with XML-encoded messages, the inherent inflexibility of the HTTP protocol for the requirements at hand and the fact that each component would be required to be configured with a web services container (web server). However, web services are used for the Cross-Platform Manager’s public interface since cross-platform interoperability was the primary requirement in this regard. The objects remoted using .NET Remoting within the four distributed components of Alchemi, the Manager, Executor, Owner and Cross-Platform Manager are instances of GManager, GExecutor, GApplication and CrossPlatformManager respectively. GManager, GExecutor, GApplication derive from the GNode class which implements generic functionality for remoting the object itself and connecting to a remote Manager via the IManager interface. The Manager executable initializes an instance of the GManager class, which is always remoted and exposes a public interface IManager. A key point to note is the fact the IManager interface derives from IExecutor. This allows a Manager to connect to another Manager and appear to be an Executor. This is the means by which the architecture supports the building of hierarchical grids. The Executor executable creates an instance of the GExecutor class. For non-dedicated execution, there is one-way communication between the Executor and the Manager. Where two-way communication is possible and dedicated execution is desired, GExecutor is remoted and exposes the IExecutor interface so that the Manager may communicate with it directly. The Executor installation provides an option to install a screen saver, which initiates non-dedicated execution when activated by the operating system.
The Owner component - inside which instances of the `GApplication` object are created via Alchemi API - communicates with the Manager in a similar fashion to `GExecutor`. While two-way communication is currently not used in the implementation, the architecture caters for this by way of the `IOwner` interface.

The Cross-Platform Manager web service is a thin wrapper around `GManager` and uses applications and threads internally to represent tasks and jobs (the `GJob` class derives from `GThread`) via the public `ICrossPlatformManager` interface.
4.3.2 Grid Application Lifecycle

To develop and execute a grid application the developer creates a custom grid thread class that derives from the abstract **GThread** class. An instance of the **GApplication** object is created and any dependencies required by the application are added to its **DependencyCollection**. Instances of the **GThread** derived class are then added to the **GApplication**’s **ThreadCollection**.

The lifecycle of a grid application is shown in Figure 6 and Figure 7, showing simplified interactions between the Owner and Executor nodes respectively and the Manager. The **GApplication** serializes and sends relevant data to the Manager, where it is persisted to disk and threads scheduled. Application and thread state is maintained in a SQL Server / MSDE database.
Non-dedicated executors poll for threads to execute until one is available. Dedicated executors are directly provided a thread to execute by the Manager.

Threads are executed in .NET application domains, with one application domain for each grid application. If an application domain does not exist that corresponds to the grid application that the thread belongs to, one is created by requesting, desterilizing and dynamically loading...
the application’s dependencies. The thread object itself is then desterilized, started within the application domain and returned to the Manager on completion.

After sending threads to the Manager for execution, the \texttt{GApplication} polls the Manager for finished threads. A user-defined \texttt{GThreadFinish} delegate is called to signify each thread’s completion and once all threads have finished a user-defined \texttt{GApplicationFinish} delegate is called.
Chapter 5

Alchemi API: Grid Thread

Programming Model
5.1 Introduction

Grid Thread Model
Alchemi simplifies the development of grid applications by providing a programming model that is object-oriented and that imitates traditional multi-threaded programming. The atomic unit of parallel execution is a grid thread with many grid threads comprising a grid application (hereafter, ‘applications’ and ‘threads’ can be taken to mean grid applications and grid threads respectively, unless stated otherwise). Developers deal only with application and thread objects and any other custom objects, allowing him/her to concentrate on the grid application itself without worrying about the "plumbing" details. Furthermore, this kind of abstraction allows the use of programming language constructs such as events between local and remote code. All of this is offered via the Alchemi .NET API.

The additional benefit of this approach is that it does not limit the developer to applications that are completely or “embarrassingly” parallel. Indeed, it allows development of grid applications where inter-thread communication is required. While Alchemi currently only supports completely parallel threads, support for inter-thread communication is planned for the future. Finally it should be noted that grid applications utilizing the Alchemi .NET API can be written in any .NET-supported language e.g. C#, VB.NET, Managed C++, J#, JScript.NET.

Grid Job Model
Traditional grid implementations have offered a high-level abstraction of the "virtual machine", where the smallest unit of parallel execution is a process (typically referred to as a job, with many jobs constituting a task). Although writing software for the “grid job” model
involves dealing with processes, an approach that can be complicated and inflexible, Alchemi’s architecture supports it via web services interface for the following reasons:

. grid-enabling existing applications; and
. cross-platform interoperability with grid middleware that can leverage Alchemi

Grid tasks and grid jobs are represented internally as grid applications and grid threads respectively.

5.2 Overview of Grid Application Development with Alchemi

The two central classes in the Alchemi .NET API are GThread and GApplication, representing a grid thread and grid application respectively. There are essentially two parts to an Alchemi grid application. Each is centered on one of these classes:

. code to be executed remotely (a grid thread and its dependencies) and
. code to be executed locally (the grid application itself).

A custom grid thread is implemented by writing a class that derives from GThread, overriding the void Start() method, and marking the class with the Serializable attribute. Code to be executed remotely is defined in the implementation of the overridden Start method. This GThread-derived class and any dependencies (that are not part of the .NET Framework) must be compiled as one or more .NET Assemblies. Figure 8 shows a very simple grid thread that multiplies two integers.
Figure 5.1: Listing of a simple grid thread (C#).

The grid application itself can be any type of .NET application. It creates instances of the custom grid thread, executes them on the grid and uses each thread’s results. Figure 9 shows a very simple grid application that uses the custom `MultiplierThread` grid thread discussed above.

```csharp
using System;
using Alchemi.Core;
namespace Alchemi.Examples.Tutorial
{
    [Serializable]
    public class MultiplierThread : GThread
    {
        private int _A;
        private int _B;
        private int _Result;
        public int Result
        {
            get { return _Result; }
        }
        public MultiplierThread(int a, int b)
        {
            _A = a;
            _B = b;
        }
        public override void Start()
        {
            _Result = _A * _B;
        }
    }
}
using System;
using Alchemi.Core;

namespace Alchemi.Examples.Tutorial
{
    class GridApplication
    {
        [STAThread]
        static void Main(string[] args)
        {
            Console.WriteLine("[enter] to start grid application ...");
            Console.ReadLine();
            // create grid application
            GridApplication ga = new GridApplication("localhost", 9000);
            // add GridThread module as a dependency
            ga.Manifest.Add(new ModuleDependency(typeof(MultiplierThread), Module));
            // create and add 10 threads to the application
            for (int i = 0; i < 10; i++)
            {
                // create thread
                MultiplierThread thread = new MultiplierThread(i, i + 1);
                // set the thread finish callback method
                thread.FinishCallback = new GThreadFinish(ThreadFinished);
                // add thread to application
                ga.Threads.Add(thread);
            }
            // set the application finish callback method
            ga.FinishCallback = new GApplicationFinish(ApplicationFinished);
            // start application
            ga.Start();
            Console.ReadLine();
        }

        static void ThreadFinished(GThread th)
        {
            // cast GThread back to MultiplierThread
            MultiplierThread thread = (MultiplierThread) th;
            Console.WriteLine("thread # [0] finished with result '\{1\}', thread.Id, thread.Result");
        }

        static void ApplicationFinished()
        {
            Console.WriteLine("application finished\n");
            Console.WriteLine("[enter] to continue ...");
        }
    }
}

Figure 5.2: Listing of a simple grid application (C#).

5.3 The Alchemi Job Submitter

The Job Submitter is a console application ([SDK]\alchemi_jsub.exe) can be used to submit tasks/jobs and retrieve their results. Its use is demonstrated here by a simple example.

The file reverse.task.xml contains an example representation of a task:
The Job Submitter must be started with the host and port of an Alchemi Manager and user credentials. The following screenshot shows how the Job Submitter can be used to submit a task, monitor its jobs and retrieve results:
Chapter 7

Future Expansion Ability: BDREN, TEIN3 & NSU
7.1 BDREN (BANGLADESH RESEARCH AND EDUCATION NETWORK)

The BdREN (Bangladesh Research and Education Network) is a high-performance data communication ICT network that would be owned and operated by the universities of Bangladesh. Using the connectivity with global resources we can make our Grid system connected with worldwide network through BDREN. This will lead our researchers and students are also involved with exchanging their high performance research projects and resources in “Global arena”. Even general student unaware to handle the grid efficiently can able to use the grid and have their high performance computation accomplished.

The BdREN network naturally will be an ICT based system having its own campus network (Intranet) connected to other campus networks, to national fiber backbone (Internet) and to other RENs globally, with its own routers, switches, servers, radio links and most important, its own fiber.

BdREN will have a countrywide fiber backbone. Initially a backbone in Dhaka will be prepared. Gradually this network will be extended to Chittagong, Khulna, Rajshahi, Sylhet and other relevant areas. Primarily, all the public and private universities are assumed to be the member of BdREN. Essentially, all universities of the country will be connected to this single network. Thus through the BDREN network we will able to be connected with TEIN 3 (Trans-Eurasia Information Network) of the European Union (EU) and the Asia-Pacific
Network (APAN) and INTERNET 2 of USA. It will also connect with the Bangladesh National Internet Exchange (BDIX).

![Network Diagram](image)

Figure 7.1: connecting NSU with BdRen network

### 7.2 TEIN3 (TRANS-EURASIA INFORMATION NETWORK)

The Trans-Eurasia Information Network (TEIN) is one of the new initiatives endorsed by ASEM III (October 2000, Seoul, Korea) to connect research networks between Asia and Europe by linking EU's GEANT, the pan-European gigabit research network, with Asia's research networks such as the APII Test beds in order to promote information exchanges in research and development and education. At ASEM III, Korea, European Commission (EC)
and Singapore jointly proposed the TEIN project. The ASEM leaders emphasized the need to establish and expand information and research networks between the two regions and among ASEM partners in order to facilitate the flow of knowledge and information as well as research endeavors. This initiative aims to contribute to enhancing exchanges and cooperation between Asia and Europe through increased and more effective information flows; enhance and diversify research exchanges and cooperation between Asia and Europe; expand and diversify speedier and more powerful telecommunication connections between Asia and Europe.

**GÉANT**

GÉANT is the main European multi-gigabit computer network for research and education purposes. GÉANT link speeds range from 155 Mbit/s on the slowest spur links to 10 Gbit/s in the core optical fiber network. It is intended that GÉANT interconnect with other regional research networks to create a single global research network. It currently has gigabit links to North America and Japan. The GÉANT network is managed by DANTE (Delivery of Advanced Network Technology to Europe).

**THE TEIN3 PROJECT**

The third generation of the Trans-Eurasia Information Network (TEIN3) provides a dedicated high-capacity Internet network for research and education communities across Asia-Pacific. It currently connects 11 countries in the region, and provides direct connectivity to Europe’s GÉANT2 network. TEIN3 offers Asia-Pacific a gateway for global collaboration, enabling over 30 million users across the region to participate in joint projects with their peers in Europe and other parts of the world.
Following the success of its predecessor, TEIN3 extends the project until 2011. TEIN3 aims to:

- Increase Internet connectivity for research and education between Europe and Asia via high-speed direct links to GÉANT2
- Improve intra-regional connectivity across Asia with upgraded link capacities
- Expand geographical coverage of the network to include other Asian countries
- Support and promote collaborative network applications
- Act as a catalyst for the development of national research networking in the developing countries in the Asia-Pacific region
- Transfer project management to Asian ownership
- Develop funding models for long-term sustainability
- Pave the way for TEIN4

**FUNDING**

Building on the success of TEIN2, at the end of 2007 the European Commission announced a new financial package for the seamless continuation of the network beyond 2008. The EC is contributing €12m towards the cost of TEIN3. Further substantial funding and link capacity are provided by the Asian partners.

**GEOGRAPHICAL EXPANSION**

Following a highly successful tender process, TEIN3 entered service in 2009. It builds on the achievements of the TEIN2 network which already connects China, Indonesia, Japan, Korea, Laos, Malaysia, the Philippines, Singapore, Thailand, Vietnam and Australia. Studies are
underway to extend the network into the South Asian subregion and other interested Asian countries, such as Cambodia.

ASEM SUCCESS STORY

The Asia-Europe Meeting (ASEM) was established in 1996 as an informal Euro-Asia cooperation forum. Membership currently comprises the 27 European Union member states, the European Commission, 16 Asian countries and the Association of Southeast Asian Nations (ASEAN) Secretariat. ASEM holds biennial Summit meetings at the level of heads of state and government.

The Trans-Eurasia Information Network (TEIN) initiative was launched at the ASEM 3 Summit in Seoul in 2000 to improve Euro-Asian research networking. The first result was the installation of a France-Korea connection (TEIN1) in 2001 which was upgraded several times in the following years to meet the increasing demands from users. ASEM 6 in Helsinki in September 2006 marked the official inauguration of TEIN2, which extended the bilateral success of TEIN1 to the regional level by creating the first large-scale regional data-communications network for research and education across the Asia-Pacific region.

THE TEIN3 NETWORK

The diagram below shows the new TEIN3 network brought into service in early 2009 to seamlessly replace the TEIN2 network.
Extension to South Asia

One of the central objectives of the TEIN3 project is to extend the TEIN network to the South Asian sub-region. To this end a feasibility study (SAFS) was undertaken in conjunction with the prospective South Asian partners in Afghanistan, Bangladesh, Bhutan, India, Nepal, Pakistan and Sri Lanka.
The study took place between May and December 2008 and assessed:

• The status and prospects for research and education networking in the target countries

• The role that TEIN3 could play to assist the countries, and

• The obligations of the target countries of participation in TEIN3 Thus this link is a pioneering cooperation between European and North American programmes, a model which is well suited to build on in extending research and education networking to other countries in South Asia.

In addition to networking the Asian-Pacific countries together, the links to GÉANT2 also provide connections to:

• Research and education networks in over 30 European countries

• The major research networks in North America (Abilene, CANARIE, ESnet)

• The South African research network, TENET

• The Indian research network, ERNET

• Research networks in Latin America via GÉANT2's connection to the ALICE project

• Research networks in the Mediterranean via GÉANT2's connection to EUMEDCONNECT2

THE TEIN3 FACTS

• Supports a community of more than 30 million users

• Regional connectivity extends from three network hubs in Beijing, Singapore and Hong Kong to connect ten countries across the Asia-Pacific region

• Direct links to GÉANT2 create a potential user base of more than 60 million

• Via GÉANT2, it enables world-class EU-Asian research collaboration

• Operates at speeds of up to 2.5 Gbps
Managed by research networking expert DANTE (Delivery of Advanced Network Technology to Europe).

- Aims to transfer project management to Asian ownership, develop funding models for long-term sustainability and to extend geographical coverage of the network

7.3 NSU GRID-LAB: RESEARCH SCOPE FOR FURTHER DEVELOPMENT

As high performance computing is an emerging research area where several geographically distributed computing resources can be utilized to solve a scientific or technical problem. To have, a powerful and cost effective computing infrastructure server is the idea. Around the globe, universities, research labs and companies are looking forward to install and use Grid instead of buying expensive servers and resources. In NSU, there are number of computers in the LABs and offices, which remains idle time to time. A local grid can be created where idle CPU cycles can be effectively used to achieve a powerful computational system that can be used for research and educational purpose.

There will be a great scope for having a grid lab where development of the grid can be continued and able to be utilized for further research works. Already, the developed system with user friendly portal is able to solve high performance computing for parallel processing jobs, data mining and resource allocation areas. If there is a Grid-Lab, we can utilize the Grid for different research works and further developments. Thus this lab is very beneficial for research projects, implementation of grid scheduling algorithms and high performance computation.