

FOSS in Scientific Applications

© IOSN South Asia Node (CDAC Chennai) - 2007

This work is released under the Creative Commons Attribution 2.5 License. For full details of the license, please refer <http://creativecommons.org/licenses/by/2.5/legalcode>

1. Introduction

FOSS (Free/Open Source Software) programs are programs whose licenses give users the freedom to run the program for any purpose, to study and modify the program, and to redistribute copies of either the original or modified program without having to pay royalties to any previous developers. Thus open source is a development method for software that expresses the power of distributed peer review and transparency of process. The promise of open source is better quality, higher reliability, more flexibility, lower cost, and an end to predatory vendor lock-in [1]. The open standards and open contents are several other terms those are related to FOSS. The goal of this paper is to define these related terms in FOSS and show the use of FOSS in different scientific applications including construction of grid environment, solving bio-informatics problems, developing National Address Database (NAD) and so on.

2. What are FOSS, Open Standards and Open contents?

Normally we define the term “open source” as the software with open coding available. But truly it is not only the open codes but also open standards and open contents. According to OSI (Open Source Initiative) [2], the distribution term of open source software must fulfill several criteria. OSI also defines these criteria as follows:

I. Free Redistribution

Any party can not restrict the license of open source software by preventing the selling or giving away the software as a component of an aggregate software distribution containing programs from several different sources. The license shall not require a royalty or other fee for such sale.

II. Source Code

The program must include source code, and must allow distribution in source code as well as compiled form. Where some form of a product is not distributed

with source code, there must be a well-publicized means of obtaining the source code for no more than a reasonable reproduction cost preferably, downloading via the Internet without charge. The source code must be the preferred form in which a programmer would modify the program. Deliberately obfuscated source code is not allowed. Similarly, Intermediate forms such as the output of a preprocessor or translator are not allowed.

III. Derived Works

The license must allow modifications and derived works, and must allow them to be distributed under the same terms as the license of the original software.

IV. Integrity of The Author's Source Code

The license may restrict source-code from being distributed in modified form only if the license allows the distribution of "patch files" with the source code for the purpose of modifying the program at build time. The license must explicitly permit distribution of software built from modified source code. The license may require derived works to carry a different name or version number from the original software.

V. No Discrimination Against Persons or Groups

The license must not discriminate against any person or group of persons.

VI. No Discrimination Against Fields of Endeavor

The license must not restrict anyone from making use of the program in a specific field of endeavor. For example, it may not restrict the program from being used in a business, or from being used for genetic research.

VII. Distribution of License

The rights attached to the program must apply to all to whom the program is redistributed without the need for execution of an additional license by those parties.

VIII. License Must Not Be Specific to a Product

The rights attached to the program must not depend on the program's being part of a particular software distribution. If the program is extracted from that distribution and used or distributed within the terms of the program's license, all parties to whom the program is redistributed should have the same rights as those that are granted in conjunction with the original software distribution.

IX. License Must Not Restrict Other Software

The license must not place restrictions on other software that is distributed along with the licensed software. For example, the license must not insist that all other

programs distributed on the same medium must be open-source software.

X. License Must Be Technology-Neutral

No provision of the license may be predicated on any individual technology or style of interface.

Even with these broad properties and definition, I think I need to define the terms "open standards" and "open content" with little more effort. That's why here I have presented some few words about these two.

Open Standard

In a recent article Jonathan Schwartz of Sun Microsystems argues that open standards (i.e., open protocols) are more important than open source code. While he sets up an opposition between standards and source (that little word "versus" in his title), in reality there is no such thing--the two are mostly orthogonal to each other and, as we shall see, both are necessary.

We can define standards in some different way. Some people think that when the IETF or W3C approves a protocol or format, it thereby becomes a standard. But standardization is not a matter of approval; rather, it is a matter of acceptance in the market. Market is a complex stew of projects and organizations that develops and uses the emerging standard. In fact, it looks a lot like the ecosystem of developers and users, but written on a global scale. Not all standards are open (for example, MS Word and PowerPoint). However, when formats and protocols are open, then open implementations that are technically strong usually (but not always) tend to be accepted by the marketplace as standards [1].

Indeed, often a particular implementation of a certain protocol becomes not only a standard but also the dominant market maker. For example, Apache is the dominant web server and a protocol like HTTPng failed to catch on in large part because it lacked support in the Apache community. We see the same phenomenon in the Jabber world, where the jabberd server is the dominant player. So although diversity is a good thing, it's much better for the health of the ecosystem if the dominant implementation is open rather than closed. A strong open source "anchor" helps ensure the openness of the underlying technology [1].

Open Content

Open content, coined by analogy with "open source", describes any kind of creative work (including articles, pictures, audio, and video) or engineering work (i.e. open machine design) that is published in a format that explicitly allows the copying and the modifying of the information by anyone; not exclusively by a closed organization, firm or individual.

Technically, it is royalty free, share alike and may or may not allow commercial redistribution. Content can be either in the public domain or under an open license like one of the Creative Commons licenses.

It is possible that the first documented case of open content was with the Royal Society, where they aspired toward information sharing across the globe as a public enterprise. The commonality is difficult to dismiss.

3. FOSS in Scientific Application

Current world is continuously changing and progressing. Sharing of information, skills, and experiences is hardly necessary now. Open source offers us such options of sharing multiple skills on the same software. That improves the performance most of the cases. That's why people are eager to use Linux more than windows. Such practices of FOSS in scientific world are now very demanding.

From schools to business offices, all organizations are currently using open source software for their needs. The broad example of open source software is Unix operating system, like Linux Redhat, Ubuntu, Fedora etc. Most of the graduate schools in United States are currently using these open source Operating Systems in their developments and research works. Also due to different copyright conflicts, students and professors are encouraged to use the open source software in different research applications. Open source software provides diversity, which is necessary for not only for developed countries but also for the developing countries to come forward with better output. In accounting, business, education and even in medical applications, different open source tools are available. People in developing countries cannot afford Windows Vista, but they can afford Linux, they cannot afford Microsoft Word, but they can afford OpenOffice, they cannot afford Oracle database, but they can afford PostgreSQL. So people always have alternative resources, now they need only the awareness of using these open sources in their respective fields.

Here in this paper, I focus on some interesting areas where I have used open source software like Alchemi and Globus Toolkit with excellent outcomes. The next few sections contain these applications.

3.1 FOSS in Distributed Systems and Grid computing

I have divided this section into three parts. The first part includes the overview of Grid Computing; the second part contains details about Alchemi and the third

part is about the use of Alchemi in Grid.

3.1.1 Overview of Grid

According to Ian Foster, 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 global level popularity called as grid computing. This new paradigm has been dubbed as Internet computing, which is also called by several different names including enterprise/desktop grid computing, peer-to-peer (P2P) computing, and public distributed computing.

Theoretically, we can define Grid as follows---

“A computational grid is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities.” [3]

3.1.1.1 Characteristics of Grid

Of course, there are many big ideas behind the Grid and other systems. And of course, some of them have been around long before the name Grid appeared. Nevertheless, if we look at where the software engineers and developers who are building the Grid are spending their time and effort, then there are five big as described in grid cafe [3]. They define these areas as follows,

Sharing of Resources

The most important is the sharing of resources on a global scale. This is the very essence of the Grid. We enter the Grid to use remote resources, which allows us to do things that we cannot do with the computer we own or the computer center we normally use. This is more than simple file exchange: it is direct access to remote software, computers and data. It can even give us access and control of remote sensors, telescopes and other devices that do not belong to us.

A major challenge for the implementation of the Grid comes from this very simple fact: resources are owned by many different people. This means that they exist within different administrative domains, they run different software, and they are

subject to different security and access control policies.

Would we trust our car to a complete stranger? Probably only once! So the Grid is a bit like a car pool - we share our car with other people to be more efficient, and at other times, they share their car with us. These people could be strangers sometimes, but if they are part of the same car pool organization as us, we will generally trust them at some level, and likewise they will trust us. If one guy in the carpool is repeatedly late when it is his turn to drive, the others will complain. If that fails, they will eventually kick him out of the car pool. So there is trust, and there are mechanisms to deal with breach of trust [3].

This is the crux of the Grid philosophy - it's not about getting something for nothing, or giving computer resources to the world out of the goodness of our heart.

Security

Then, although it is hardly a novelty, security is a critical aspect of the Grid, since there must be a very high level of trust between resource providers and users, who will often never know who each other, are. Therefore, I have defined separately the security part of each open source software I use. Sharing resources is, fundamentally, in conflict with the ever more conservative security policies being applied at individual computer centers and on individual PCs. Security concerns about the following three aspects.

Access policy - resource providers and users must define clearly and carefully what is shared, who is allowed to share, and the conditions under which sharing occurs;

Authentication - we need a mechanism for establishing the identity of a user or resource;

Authorization - we need a mechanism for determining whether an operation is consistent with the defined sharing relationships.

Load Balancing

If the resources can be shared securely, then the Grid really starts to pay off when it can balance the load on the resources, so that computers everywhere are used more efficiently, and queues for access to advanced computing resources can be shortened.

This is where the Grid really starts to look interesting, even for someone blessed with a lot of computer resources. Because no matter how many resources we have, there will always be times when there is a queue of people waiting to use them. If we have a mechanism to allocate work efficiently and automatically among many resources, we can reduce the queues.

It is a lot like checkout counters in a supermarket. We know the story. Everybody goes to the shortest queue. Except, when we get in the shortest queue, of course, the lady in front of us has chosen an item that can't be read by the scanner, and it takes forever to find the right price... So ideally, we would like to know not only how many people are in each queue, but also exactly how long it is going to take them to check out.

On the Grid, in principle, we have the information about the different jobs being submitted, and since the whole thing is running on computers, we should be able to calculate the optimal allocation of resources. The development of the middleware, the software that performs this task and in general manages activity on the Grid, is the main purpose of many of the Grid projects going on today around the world

Death of the Distance

For this to work, however, communications networks have to ensure that distance no longer matters - doing a calculation on the other side of the globe, instead of just next door, should not result in any significant reduction in speed. What makes the Grid possible today is the impressive development of networking technology. Pushed by the Internet economy and the widespread penetration of optical fibers in telecommunications systems, the performance of wide area networks has been doubling every nine months or so over the last few years. Some wide area networks now operate at 155 megabits per second (Mbps), when in 1985 the US supercomputer centers were connected at 56 kilobits per second (Kbps) - that is a 3000x improvement in 15 years [3].

Open Standards

Finally, underlying much of the worldwide activity on Grids these days is the issue of open standards, which are needed in order to make sure that R&D worldwide can contribute in a constructive way to the development of the Grid, and that industry will be prepared to invest in developing commercial Grid services and infrastructure. The Global Grid Forum develops grid-specific standards. With more than 5000 individual researchers and practitioners on its list, this body is a significant force for setting standards and community developments. Currently, a standard known as OGSA (Open Grid Services Architecture), while still under development, is seen as the key reference for future Grid development projects.

3.1.1.2 Grid Architecture

The architecture of the Grid is often described in terms of "layers", each providing a specific function. In general, the higher layers are focused on the user (user-centric), whereas the lower layers are more focused on computers and networks (hardware-centric).

At the base of everything, the bottom layer is the network, which assures the connectivity for the resources in the Grid. On top of it lies the resource layer, made up of the actual resources that are part of the Grid, such as computers, storage systems, electronic data catalog, and even sensors such as telescopes or other instruments, which can be connected directly to the network.

The middleware layer provides the tools that enable the various elements (servers, storage, networks, etc.) to participate in a unified Grid environment. The middleware layer can be thought of as the intelligence that brings the various elements together - the "brain" of the Grid [3].

The highest layer of the structure is the application layer, which includes all different user applications (science, engineering, and business, financial), portals and development toolkits supporting the applications. This is the layer that users of the grid will actually see.

In most common Grid architectures, the application layer also provides the so-called serviceware, the sort of general management functions such as measuring the amount a particular user employs the Grid, billing for this use (assuming a commercial model), and generally keeping accounts of who is providing resources and who is using them - an important activity when sharing the resources of a variety of institutions amongst large numbers of different users.

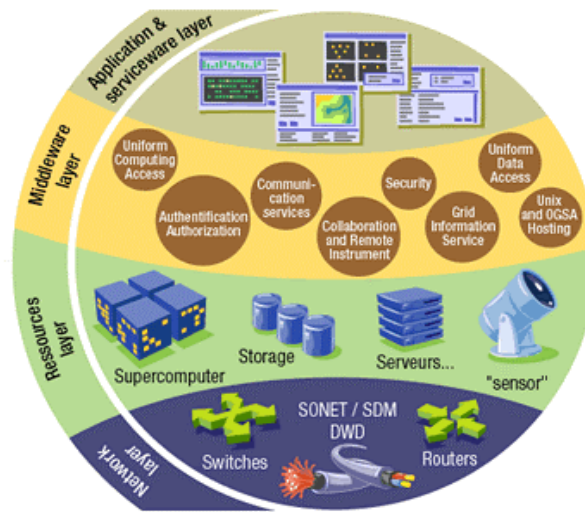


Figure.1 Grid Architecture

There are other ways to describe this layered structure. For example, experts like to use the term fabric for all the physical infrastructure of the Grid, including computers and the communication network. Within the middleware layer, distinctions can be made between a layer of resource and connectivity protocols, and a higher layer of collective services.

Resource and connectivity protocols handle all "Grid specific" network transactions between different computers and other resources on the Grid. We should remember that the network used by the Grid is the Internet, the same network used by the Web and by many other services such as e-mail. A myriad of transactions is going on at any instant on the Internet, and computers that are actively contributing to the Grid have to be able to recognize those messages that are relevant to them, and filter out the rest. This is done with communication protocols, which let the resources speak to each other, enabling exchange of data, and authentication protocols, which provide secure mechanisms for verifying the identity of both users and resources.

The collective services are also based on protocols: information protocols, which obtain information about the structure and state of the resources on the Grid, and management protocols, which negotiate access to resources in a uniform way. The services include:

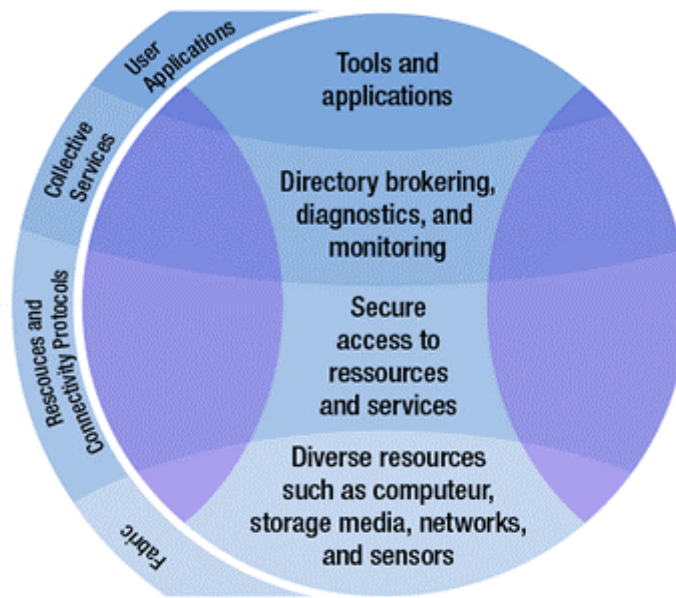


Figure.2 Layers of Grid Environment

- Keeping directories of available resources updated at all times,
- Brokering resources (which like stock broking, is about negotiating between those who want to "buy" resources and those who want to "sell")
- Monitoring and diagnosing problems on the Grid
- Replicating key data so that multiple copies are available at different locations for ease of use
- Providing membership/policy services for keeping track on the Grid of who is allowed to do what, when.

In all schemes, the topmost layer is the applications layer. Applications rely on all the other layers below them in order to run on the Grid. To take a fairly concrete example, consider a user application that needs to analyze data contained in several independent files. It will have to:

- Obtain the necessary authentication credentials to open the files (resource and connectivity protocols)
- Query an information system and replica catalog to determine where copies of the files in question can currently be found on the Grid, as well as where computational resources to do the data analysis are most conveniently located (collective services)
- Submit requests to the fabric - the appropriate computers, storage systems,

and networks - to extract the data, initiate computations, and provide the results (resource and connectivity protocols)

- Monitor the progress of the various computations and data transfers, notifying the user when the analysis is complete, and detecting and responding to failure conditions (collective services).

In order to do all of the above, it is clear that an application that a user may have written to run on a stand-alone PC will have to be adapted in order to invoke all the right services and use all the right protocols. Just like the "webifying" of applications - where users have to adapt a stand-alone application to run on a web browser, so too the Grid will require users to invest some effort into "gridifying" their applications [3]. We will see how we can do it for a Bio-Informatics problem later.

3.1.1.3 Available Open Sources for Grid

However, once a program is gridified, thousands of people will be able to use the same application and run it trouble-free on the Grid, using the middleware layers to adapt in a seamless way to the changing circumstances of the fabric. A number of open source software is available for implementing the grid environment. Some of these are:

CONDOR is a project begun already in 1988 at the University of Wisconsin, which started with the aim of pooling the computing resources of all computers in a University department. It performs "cycle scavenging", and includes ways to find appropriate resources as well as recover from faults, such as someone turning off a PC, which is running part of a Condor calculation.

CODINE (Computing for Distributed Network Environments) was a project started by a small German company, Genias Software, which later changed its name to Gridware, and was bought by Sun Microsystems in 2000. CODINE provides cycle scavenging similar to Condor, and a simple graphical user interface to view all available resources. CODINE continues as an open source project supported by Sun, and the next generation of CODINE has been renamed the Sun Grid Engine, and released free on the net.

LEGION was a project started at the University of Virginia in 1993, which took a rather purist computer science approach to building a metacomputing environment. It was based on an object-oriented approach, where everything (files, computers on the network etc.) is an object with specific access procedures, and fits into one giant virtual machine. This is elegant, but hard to

implement in practice, especially because a lot of applications for metacomputing environments are not "object oriented".

NIMROD was a project started at Monash University in Australia in 1994. It started with a very specific goal in mind: distributing very similar calculations, such as performance of an aircraft wing at many angles of attack, to many computers on a LAN.

UNICORE was a German initiative, started in 1997, to integrate supercomputing centers throughout Germany. The UNICORE middleware has elements of a Grid Toolkit, as well as a Grid Portal.

GLOBAL TOOLKIT is developed by University of Chicago, USA. It is a heavyweight open source tool built on JVM. It used ant server for providing web services. I have shown the use of Globus toolkit in Spam protection of mail server in this paper. Globus toolkit is a nice open source software for implementing the Grid environment as compared to the performance [24].

ALCHEMI is the user-friendliest software for implanting Grid due to its simple interface. The University of Melbourne, Australia developed it. I have used this Alchemi software in bio-informatics application and designing the National Address Database (NAD) of a country.

3.1.2 Overview of Alchemi

Alchemi is a dot net-based open source software for implementing Grid Environment. The latest version of Alchemi is available at www.alchemi.net [23]. While the notion of grid computing is simple enough, the practical realization of grids poses a number of challenges. Key issues that need to be dealt with are heterogeneity, reliability, application composition, scheduling, resource management and security [4][5]. The Microsoft .NET Framework [22] provides a powerful toolset that can be leveraged for all of these, in particular support for remote execution, multithreading, security, asynchronous programming, disconnected data access, managed execution and cross-language development, making it an ideal platform for grid computing middleware [6]. "Alchemi" [23] was conceived with the aim of making grid construction and development of grid software as easy as possible without sacrificing flexibility, scalability, reliability and extensibility [7][8].

3.1.2.1 Components of Alchemi

There are four types of distributed components (nodes) involved in the construction of Alchemi grids and execution of grid applications: Manager, Executor, and Owner & Cross-Platform Manager.

Manager

The Manager manages the execution of grid applications and provides services associated with managing thread execution [9]. 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 [9]. The Executors return completed threads to the Manager, which are subsequently passed on or collected by the respective Owner.

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 [9]. 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 deployment a non-dedicated 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.

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.

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 gridifying 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) [9].

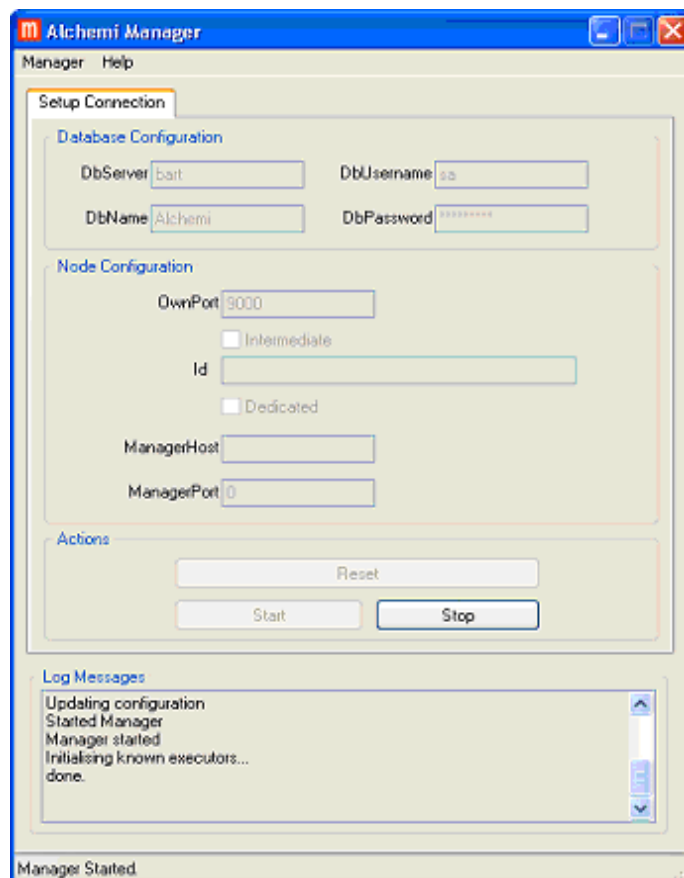


Figure.3 Alchemi Manager Started**3.1.2.2 Alchemi License Agreement**

Alchemi is open source software. The following components are licensed under the Lesser GNU General Public License (LGPL):

- Alchemi Core Library (used for developing applications on top of Alchemi): (Alchemi.Core.dll)

The following components are licensed under the GNU General Public License (GPL):

- Alchemi Manager
- Alchemi Executor
- Alchemi SDK (including Console, Job submitter, tutorial and all example programs)

3.1.3 How To Build A Grid Using Alchemi

Installing Executors on each machine that is to be part of the grid and linking them to a central Manager component create a grid. The Windows installer that comes with the Alchemi distribution and minimal configuration makes it very easy to set up a grid. After installation, we need to run the Manager in the entry portal at a specific port. The default port number is 9000. But we can change the port to whatever address we want (except the restricted ports).

Then we need to run Executors on different executor nodes and then we will tell each executor about the IP of Alchemi Manager. An Executor can be configured to be dedicated (meaning the Manager initiates thread execution directly) or non-dedicated (meaning that thread execution is initiated by the Executor.) Non-dedicated Executors can work through firewalls and NAT servers since there is only one-way communication between the Executor and Manager. Dedicated Executors are more suited to an intranet environment and non-dedicated Executors are more suited to the Internet environment. The screen shot for Alchemi Executor is in Figure 4. Thus we can build an Alchemi grid like Figure 5. Users can develop, execute and monitor grid applications using the .NET API and tools which are part of the Alchemi SDK.

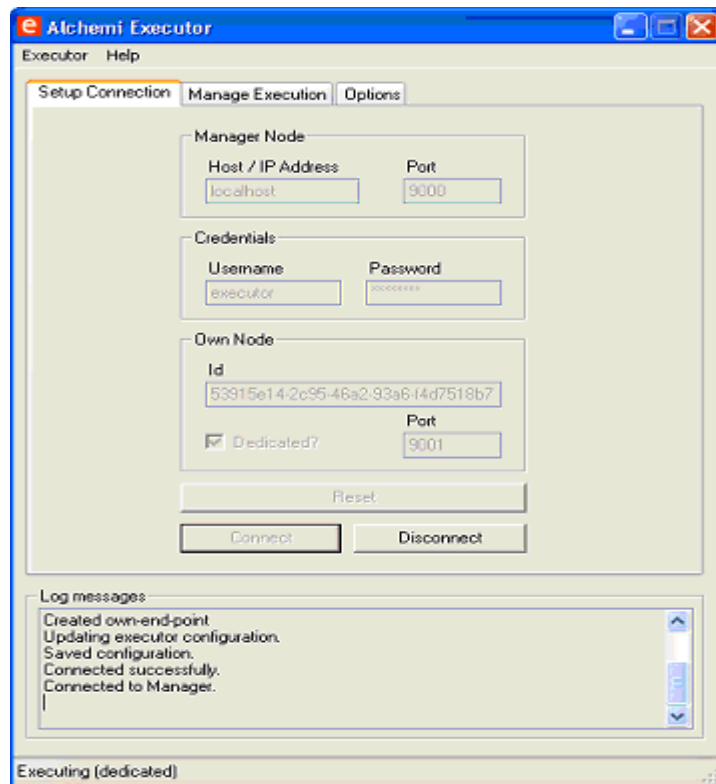


Figure.4 Alchemi Executor (dedicated) Connected to Manager

Alchemi also offers a powerful grid thread programming model, which makes it very easy to develop Grid applications and a grid job model for grid-enabling legacy or non-.NET applications.

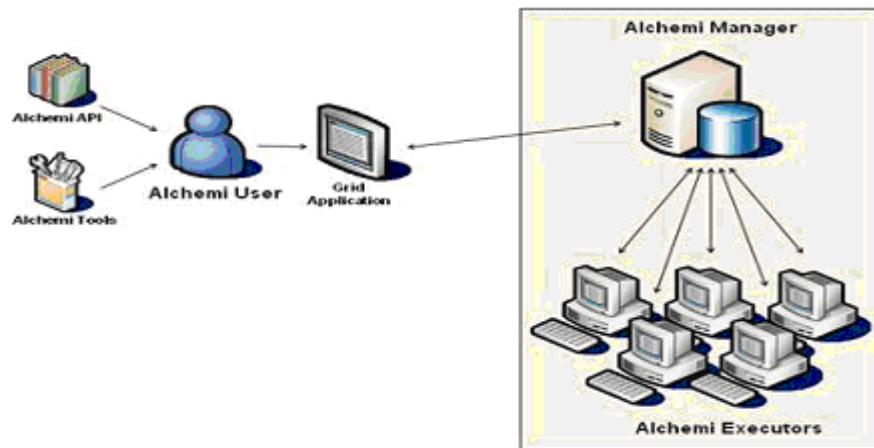


Figure.5 An Alchemi Grid

An optional component (not shown) is the Cross Platform Manager web service, which offers interoperability with custom non-.NET grid middleware.

3.1.3.1 Grid Thread Model

Minimizing the entry barrier to writing applications for a grid environment is one of Alchemi's key goals. This goal is served by an object-oriented programming environment via the Alchemi .NET API which can be used to write grid applications in any .NET-supported language.

```
using System;
using Alchemi.Core;

namespace NameSpace_Name
{
    [Serializable]
    class ClassName : GThread
    {
        //Class member variables
        public ClassName
        {
        }
        public override void Start()
        {
            //Remote Code here
            // The code that you want to execute remotely
        }
        //Other Methods
    }
}
```

Figure 6. Code for Start Method (Remote Code).

Remote code: code to be executed remotely i.e. on the grid (a grid thread and its dependencies), that is performing SQL query remotely on different Executors. And

Local code: code to be executed locally (code responsible for creating and executing grid threads). This code is implemented here on the Server that is responsible for creating Grid Threads.

A concrete grid thread is implemented by writing a class that derives from GThread, overriding the void Start() method, and marking the Serializable class with the Serializable attribute. Code to be executed remotely is defined in the implementation of the overridden void Start() method. The sample code for this is

in the Figure 6.

The application itself (local code) creates instances of the custom grid thread, executes them on the grid and consumes each thread's results. It makes use of an instance of the GApplication class, which represents a grid application. The code is shown in Figure 7.

```
class MainClassName
{
public static GApplication App = new GApplication();
[STAThread]
static void Main(string[] args)
{
    // Create grid Threads.
    // Tells them to execute the remote code in Figure 6

App.Threads.Add(new ClassName());
}
    // initialise application
Init();
    // start the application
App.Start();
    // stop the application
App.Stop()
}
```

Figure.7 Code for Creating Thread and Application (Local Code)

Instances of the GThread-derived class are asynchronously executed on the grid by adding them to the grid application. Upon completion of each thread, a 'thread finish' event is fired and a method subscribing to this event can consume the thread's results. Other events such as 'application finish' and 'thread failed' can also be subscribed to. This is shown in the Figure: 8.

```
private static void App_ThreadFinish(GThread thread)
{
    //Work that should be done on each thread finish
}

private static void App_ApplicationFinish()
{
    //Work that should be done on each Application Finish
}
```

```
// An application can contains many threads
}
```

Figure.8 Code for Finishing Thread and Application

```
<task>
<manifest>
  <embedded_file name = "Server.exe" location = "Server.exe">
</manifest>
<job id = 0>
  <input>
  //Specify Input to Each Thread
  </input>
  <output>
  //Specify where you want to place the output
  </output>
</job>
<job id =1> //Similarly.....
</task>
```

Fig. 9. Sample XML-Based Task Representation

3.1.3.2 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.

By specifying a command, input files and output files. In Alchemi, such a work unit is termed 'job' with many jobs constituting a 'task'. Tasks and their constituent jobs are represented as XML files conforming to the Alchemi task and job schema. Figure 9 shows a sample task representation that contains one job to execute the program against two input files. Before submitting the task to the Manager, references to the 'embedded' files are resolved and the files themselves are embedded into the task XML file as Base64-encoded text data. When finished jobs are retrieved from the Manager, the Base64-encoded contents of the 'embedded' files are decoded and written to disk. It should be noted that tasks and jobs are represented internally as grid applications and grid threads respectively.

3.1.3.3 Grid Security by Alchemi

Role based security is commonly used in Alchemi [10]. Every program connecting to the Manager must supply a valid username and password. Manager stores this username and password to a secured storage. Three default accounts are created during installation: executor (password: executor), user (password: user) and admin (password: admin) belonging to the 'Executors', 'Users' and 'Administrators' groups respectively.

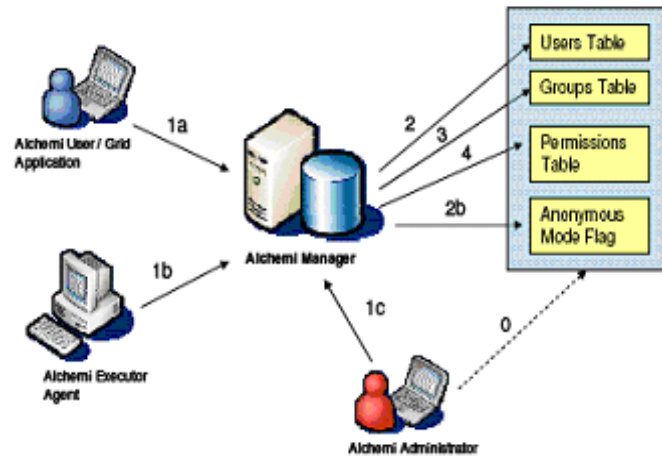


Figure.10 Role Based Security in Alchemi

Users are administered via the 'Users' tab of the Alchemi Console (located in the Alchemi SDK). Only Administrators have permissions to manage users; we must therefore initially log in with the default admin account. The Console lets us add users, modify their group membership and change passwords. The Users group (`grp_id = 3`) is meant for users executing grid applications. The Executors group (`grp_id = 2`) is meant for Alchemi Executors. By default, Executors attempting to connect to the Manager will use the executor account. If we do not wish Executors to connect anonymously, we can change the password for this account.

This grid environment can be used for any highly computational works. Some of this works may be related to Bio-informatics, Pattern Matching, Artificial Intelligence, Database Management System and so on. The next paragraph shows the use of such a Grid developed by Alchemi in Bio-Informatics.

3.2 FOSS in Bio-Informatics (Alchemi in Bio-Informatics)

One important application of Alchemi Open source Grid Technology can be in Bio-Informatics computation. Most scientific Computations in bio-Informatics require large processing time. Here I am focusing on one of these applications, called finding repeated patterns in a DNA sequence. Certain patterns on DNA and protein sequences are strongly conserved by evolution. That means, they likely have important biological functions. In DNA sequences, promoter-binding sites are typically marked by a pattern of 6-30 nucleotides shortly upstream of each co-

regulated gene. In protein sequences, highly conserved regions among similar proteins likely define the most important folds or binding sites [11]. Highly conserved does not mean completely conserved [12], and such short sequence matches may not stand out in any pair-wise alignment. Thus it is important to be able to identify possible repeats from a given sequences, and be able to test whether a given string contains a given motif.

Early statistical methods for finding repetition in biological sequences include a heuristic progressive alignment procedure These include algorithms that locate repeated substring, including tandem repeats, as well as programs for identifying known repeats, such as widely used RepeatMasker. A more rapid implementation of the same approach is MaskerAid, a wrapper for WU-BLAST that uses the BLAST engine instead of CrossMatch algorithm. Recently however, new systems based on suffix trees such as RepeatMatch and REPuter, have overcome the size limitation, at least biologically realistic input size [13]. Here I have proposed a new system for finding the repetition. I have only considered the forward pattern match [2], although the reverse and reverse complement patterns [14][15] can be searched using the same algorithm with a little modification. I have focused my attention to find a parallel algorithm for solving this problem so that this can be mapped to some Grid environment.

This part is one of the important parts of this paper, so I am presenting it here with some details. Here I have used a partitioning algorithm [16] to match the problem of Bio-Informatics with Grid and then apply Alchemi to solve it. If readers look carefully, then they can identify that any bio-informatics problem with high computational time can be mapped to such structure (gridifying) and can be solved with the reduced time delay.

I begin by defining an exact repeat as a subsequence that occurs in DNA sequence at least twice. A maximal repeat is a repeat that cannot be extended in either direction without incurring a mismatch [17][18]. Here I have considered the repeats of forward direction (a slight modification of the algorithm can be applied for reverse and reverse complement pattern matching) even with some insertions and deletions. In the initial repeat set, different repeats may be very close together and may even overlap. This event can be simplified by constructing a more general type of repeat, a 'merging repeat' that is defined as a sequence that can be found in the whole genome sequence not less than twice.

3.2.1 Algorithm

I have divided the whole process into a number of steps:

- Partitioning
- Finding Repeats
- Merging the Repeats
- Classification

The whole process can be represented by a flow chart, which is given in Figure 11.

3.2.1.1 Partitioning

For finding classification first I divide the sequence pattern into a number of different fixed length partitions. The length of each partition should be selected with tricks. Every time I want less number of partitions to improve performance of parallel computation. If the length of gene sequence is large, then the nucleotides in each partition should be increased. Lets assume, the length of each partition is L.

3.2.1.2 Finding Repeats

The technique for finding repeat is quite straightforward. Each partition of length L consists of A, T, C, and G. For A, T, C, G, I have 16 different two length patterns {AA, AT, AC, AG, TT, TA, TC, TG, CC, CA, CT, CH, GG, GA, GT, GC}. In the primary stage, I first check the repetition of these two length patterns among the partitions and inside the partitions. I keep the result information in the form of (Pattern, Partition#1, Coordinate list, partition#2, Coordinate list...).

Now in the second stage I have three length patterns. Not all the three length patterns need to be searched. I will use the two-length patterns information from the above list to find the three length patterns to be searched. These are formed by appending A, T, C, and G (after the pattern) with the previously selected two length patterns in the list. I will continue searching again with the new sequences for repetition as before. This will add several more entries to the above list:

Now the time for four length patterns and I will proceed this way until 30 length patterns. Each time for a pattern XXX will be searched only to the partitions where the XXXA pattern repeats and from the corresponding coordinates.

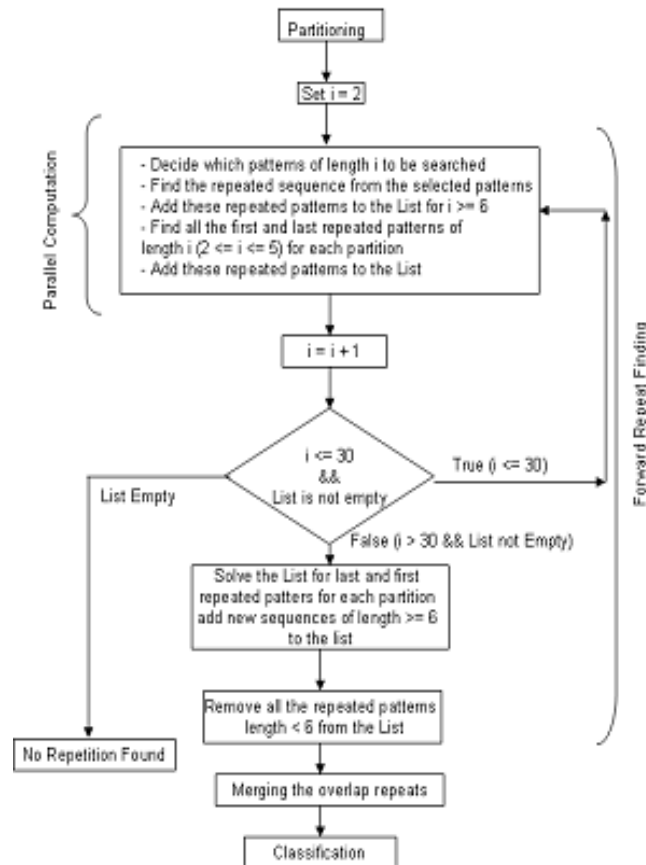


Figure.11 Flow Chart of the Algorithm.

This will minimize the searching overheads. And after the completion of whole process I will get a list of repeated patterns. Thus I impose two improvements of the brute force methods here. One is the number of patterns to be searched are determined incrementally from the lower length patterns. And second we don't need to search all the partition for repetition and even not all the coordinates of a partition.

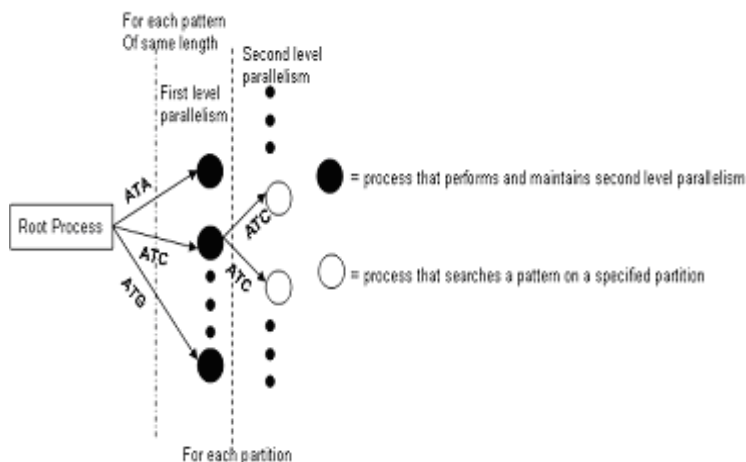


Figure.12 Two Levels of Parallelism

Searching for each sequence of same length can be performed in parallel manner. Here two levels of parallelism can be achieved. In the first level, tasks are distributed to different parallel processes for each pattern to be searched. In the second level, task for searching a pattern is distributed for each partition. The figure 12 best explains the situation of parallelism. For implementation on Grid, each process is built on separate computational nodes. Mathematical calculation here in section 3.2.5 assumes only one level of parallelism, that is, the processes at first level search for the associated pattern on the whole sequence and return the result to the root process.

I store all the minimum 6-length repeats and also the entire last and the first repeated patterns of length two to five of all partitions. This is because that repetition may occur where the partitioning happened, as my partitioning doesn't follow any prior rule.

Now, it is time to check the first and last patterns to find the more repetitive patterns those actually exist. Each time I take a last pattern (Say pattern P_1 from partition B_1). Then I check the repeated coordinates of this pattern P_1 . For each repeated coordinate, I move forward equal to the length of that pattern (P_1). If that coordinate is included in the repetition of the first pattern (P_2) of the next partition of B_1 (that is B_2) then it can be the choice of another repeated pattern provided the length is more than 5.

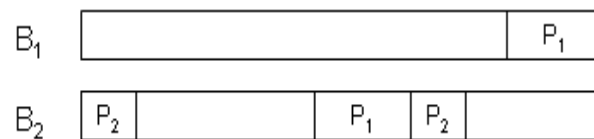


Figure.13 Checking for Repeats at Begin and End

This is then included in our list. Continuing this I can get the all repeated patterns in the conserved sequence. This can be shown in the Figure 13.

Then if the length of ($P_1 + P_2$) is greater or equal to 6, then can be treated as a repeated sequence and added in the list. This is done for all the repeated pattern

coordinates. As I am not saving patterns for only one length, so when a pattern of length 5 occurs at the beginning or at the end of the partition, then they require a special attention as follows:

- If repeated pattern of length 5 occurs at the end of any partition P_i , then the next one length pattern after the other repetition is checked with the first one length pattern of the partition P_{i+1} .
- If the repeated pattern of length 5 occurs at the beginning of partition P_i , then the previous one length pattern before the other repetition is checked with the last one length pattern of partition P_{i-1} .

3.2.1.3 Merging the Repeats

Using the list of operation points, I begin merging the exact repeats using the following criteria. Given two partition points $p1 = (A1, A2, l_A)$ and $p2 = (B1, B2, l_B)$, [here, $A1, A2, B1, B2$ are coordinates of DNA sequence and l_A and l_B are length of the respective patterns where $A1 < B1$]. I compute the distance between the non overlapping repeats as

$$d(p1, p2) = \max(0, B1 - A1 - l_A + 1)$$

Next, given a maximum gap size $G > 0$, the merging with gap protocol uses the rule that sequences corresponding to $p1$ and $p2$ are merged if

$$d(p1, p2) < G.$$

Now, the merging with overlap protocol only merges the sequences that overlap one another, that is, they are at least partially identical. We can denote the overlap of two sequences as

$$O(p1, p2) = \max(0, A1 + l_A - B1 + 1) \text{ for } A1 < B1$$

Then the criterion for 'merging with overlap' is as follows:

Given a minimum overlap proportion op , where $0 \leq op \leq 1$, then the repeat points $(A1, A2, l_A)$ and $(B1, B2, l_B)$ are merged if the following condition is

satisfied

$$O(p1, p2) > op \min (l_A, l_B)$$

The parameter op is interpreted as a fraction of the shorter of the two repeats. Thus for $op = 0.75$, I will merge two overlap sequences if the length of the overlap is at least 75% of the length of the shorter sequence.

Using either merging procedure, if two sequences are merged then the new sequence will be defined as a merging repeat with starting position $M = A1$ and with length $l_M = \text{Max}(A1+l_A, B1 + l_B)$. The merging procedure is not permitted to merge pairs of partition points of the form $(B1, B2, l_B)$ and $(B2, B1, l_B)$. This condition avoids repetitiveness within the merging repeats.

3.2.1.4 Classification

If a merging repeat has at least one reference in common with another, then they belong to the same class. Thus after solving the classification I get a list of patterns of different classes.

3.2.2 An Illustrative Example

To better understand the situation I have represented a DNA sequence here and show different steps of finding the repeats. This is shown in the Table 1. For the length of 180 nucleotides of DNA, we partition the sequence into 5 parts each containing 40 nucleotides and the last partition is padded with X for uniformity in size. The result shows three classes of repetition from the sequence.

3.2.3 Designs and Implementation

The System is implemented considering the demand of faster response and processing time for any genome sequence. At Section 3.2.5 of this paper, I have presented a performance evaluation graph that shows how the performance of the parallel algorithm in the Grid System varies from the simple application without the Grid technology and also how the performance increases with the increase of node number. Different parts of this system are shown next.

3.2.3.1 Components of the system

The components of the system include Manager nodes, Executor nodes and Grid Server nodes. In my implementation I use the same node as the Manager and

Grid Server node, though their activities are different. The duties and work associated to each component are described next.

Table. 1 Example for Repeat Finding Algorithm

| | |
|--|--|
| Sequence | GGCGGTCATTGCGGTCAGTAAGTGACTGGGATCGAACCTGAAGAGCTGAAGAGCTATGTTCGGTAGTGGTG GACCGACGTAATTGGTTGCCGGCCAGGCATAGGCGCGTTTACTACGTAATGCTCAGTAATGGATACTGACTGCATCGTATTGGTGGCATTGTCATCGCACTCACCTG |
| Partitioning | 1)GGCGGTCATTGCGGTCAGTAAGTGACTGGGATCGAACCTG 2)AAGAGCCTGAAGAGCTATGTTCGGTAGTGGTGGACCGACGT 3)AATTGGTTGCCCGGCCAGGCATAGGCGCGTTTACTACGTA 4)ATGCTCAGTAATGGATACTGACTGCATCGTATTGGTGGCA 5)TTTGTTCATCGCACTCACCTGXXXXXXXXXXXXXXXXXXXXXXX |
| Complete Match (length >= 6) | <i>CAGTAA</i> , P1, {16}, P4, {6} <i>TGACTG</i> , P1, {23}, P4, {19} <i>TGGTGG</i> , P2, {27}, P4, {33} <i>ATTGGT</i> , P3, {2}, P4, {31} |
| First Pattern Match (2 <= Length <= 5) | [GG , P1, {1, 4, 13, 28, 29}, P2, {22, 28, 31}, P3, {5, 13, 18, 24}, P4, {13, 34, 37}] [GGC , P1, {1}, P3, {13, 18, 24}, P4, {37}] [GGCG , P1, {1}, P3, {24}] [AA , P1, {20, 35}, P2, {1, 10}, P3, {1}, P4, {10}] [AAG , P1, {20}, P2, {1, 10}] [AAGA , P2, {1, 10}] [AAT , P3, {1}, P4, {10}] [TTT , P3, {30}, P5, {1}] [AT , P1, {8, 31}, P2, {17}, P3, {2, 21}, P4, {1, 11, 15, 26, 31}, P5, {7}] [ATG , P2, {17}, P4, {1, 11}] [TT , P1, {9}, P3, {3, 7, 30}, P4, {32}, P5, {1, 2}] |
| Last Pattern Match (2 <= Length <= 5) | [TG , P1, {10, 23, 27, 39}, P2, {8, 18, 27, 30}, P3, {4, 8}, P4, {2, 12, 19, 23, 33, 36}, P5, {3, 19}] [CTG , P1, {26, 38}, P2, {7}, P4, {18, 22}, P5, {18}] [CCTG , P1, {37}, P2, {6}, P5, {17}] [GT , P1, {5, 14, 18, 22}, P2, {19, 23, 26, 29, 39}, P3, {6, 29, 38}, P4, {8, 29, 35}, P5, {4}] [CGT , P2, {38}, P3, {28, 37}, P4, {28}] [ACGT , P2, {37}, P3, {36}] |

| | <p>[TA, P1, {19}, P2, {16, 24}, P3, {22, 32, 35, 39}, P4, {9, 16, 30}] [GTA, P1, {18}, P2, {23}, P3, {38}, P4, {8, 29}] [CGTA, P3, {37}, P4, {28}] [CA, P1, {7, 16}, P3, {16, 20}, P4, {6, 25, 39}, P5, {6, 11, 15}] [ACCTG, P1, {36}, P5, {16}] [GCA, P3, {19}, P4, {24, 38}, P5, {10}] [GGCA, P3, {18}, P4, {37}]</p> | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----------------------------|---|----------------------|----------------------|----------------------|-----------|-----------|-----------|--------|---|---------|----|---------|---|---------|---|----------|----|---|-------------|---|---------|---|---------|---|--------|---|--------|---|---------|---|---------|---|---------|---|---------|---|----------|---|---------|---|----------|---|---------|---|---------|---|---------|---|
| Repeated Patterns | <p>[Cord {16, 126}, Length 6], <i>CAGTAA</i> [Cord {23, 139}, Length 6], <i>TGACTG</i> [Cord {38, 47,}, Length 8], <i>CTGAAGAG</i> [Cord {67, 153}, Length 6], <i>TGGTGG</i> [Cord {77, 116}, Length 7], <i>ACGTAAT</i> [Cord {82, 151}, Length 6], <i>ATTGGT</i> [Cord {118, 128}, Length 6] <i>GTAATG</i></p> | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Pre-processing the Repeats | <table border="1"> <thead> <tr> <th>Coordinates</th> <th>Length</th> </tr> </thead> <tbody> <tr><td>16, 126</td><td>6</td></tr> <tr><td>23, 139</td><td>6</td></tr> <tr><td>38, 47</td><td>8</td></tr> <tr><td>67, 153</td><td>6</td></tr> <tr><td>77, 116</td><td>7</td></tr> <tr><td>82, 151</td><td>6</td></tr> <tr><td>118, 128</td><td>6</td></tr> </tbody> </table> | Coordinates | Length | 16, 126 | 6 | 23, 139 | 6 | 38, 47 | 8 | 67, 153 | 6 | 77, 116 | 7 | 82, 151 | 6 | 118, 128 | 6 | <table border="1"> <thead> <tr> <th>Coordinates</th> <th>Length</th> </tr> </thead> <tbody> <tr><td>16, 126</td><td>6</td></tr> <tr><td>23, 139</td><td>6</td></tr> <tr><td>38, 47</td><td>8</td></tr> <tr><td>47, 38</td><td>8</td></tr> <tr><td>67, 153</td><td>6</td></tr> <tr><td>77, 116</td><td>7</td></tr> <tr><td>82, 151</td><td>6</td></tr> <tr><td>116, 77</td><td>7</td></tr> <tr><td>118, 128</td><td>6</td></tr> <tr><td>126, 16</td><td>6</td></tr> <tr><td>128, 118</td><td>6</td></tr> <tr><td>139, 23</td><td>6</td></tr> <tr><td>151, 82</td><td>6</td></tr> <tr><td>153, 67</td><td>6</td></tr> </tbody> </table> | Coordinates | Length | 16, 126 | 6 | 23, 139 | 6 | 38, 47 | 8 | 47, 38 | 8 | 67, 153 | 6 | 77, 116 | 7 | 82, 151 | 6 | 116, 77 | 7 | 118, 128 | 6 | 126, 16 | 6 | 128, 118 | 6 | 139, 23 | 6 | 151, 82 | 6 | 153, 67 | 6 |
| Coordinates | Length | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 16, 126 | 6 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 23, 139 | 6 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 38, 47 | 8 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 67, 153 | 6 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 77, 116 | 7 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 82, 151 | 6 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 118, 128 | 6 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Coordinates | Length | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 16, 126 | 6 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 23, 139 | 6 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 38, 47 | 8 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 47, 38 | 8 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 67, 153 | 6 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 77, 116 | 7 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 82, 151 | 6 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 116, 77 | 7 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 118, 128 | 6 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 126, 16 | 6 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 128, 118 | 6 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 139, 23 | 6 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 151, 82 | 6 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 153, 67 | 6 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Merging the Repeated Map | <table border="1"> <thead> <tr> <th>M</th> <th>l_M</th> <th>n_M</th> <th>R1</th> <th>R2</th> <th>R3</th> </tr> </thead> <tbody> <tr> <td>16</td> <td>6</td> <td>1</td> <td>16</td> <td></td> <td></td> </tr> <tr> <td>23</td> <td>6</td> <td>1</td> <td>23</td> <td></td> <td></td> </tr> </tbody> </table> | M | l_M | n_M | R1 | R2 | R3 | 16 | 6 | 1 | 16 | | | 23 | 6 | 1 | 23 | | | <p>M = Starting point of pattern l_M = Length of the Pattern</p> | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| M | l_M | n_M | R1 | R2 | R3 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 16 | 6 | 1 | 16 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 23 | 6 | 1 | 23 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

| | 47 6 1 38 47 67 6 1 67 77 11 2 77 116 8 2 77 116 126 8 2 16 118 126 139 6 1 23 139 151 8 2 67 77 151 | n_M = Number of Repetition R_i = Repeated Reference | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----------------|---|--|--------------------|--|-------|------------|--------|----------|---|----|---|---------------|---|----|---|---------------|---|----|----|--------------------|---|-----|---|-----------------|---|-----|---|-----------------|---|-----|---|-----------------|---|----|---|---------------|---|-----|---|---------------|
| Classification | <p>Initial Classification</p> <p>Class 1 16 Class 2 23 Class 3 38 47 Class 4 67 Class 5 77 116</p> <p>This can be merged into as follows:</p> <p>Class 1 16 77 116 128 Class 2 23 Class 3 38 47 Class 4 67</p> <p>Final Classification</p> <p>Class 1 16 67 77 116 126 151 Class 2 23 139 Class 3 38 47</p> | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Result | <table border="1"> <thead> <tr> <th>Class</th> <th>Coordinate</th> <th>Length</th> <th>Sequence</th> </tr> </thead> <tbody> <tr> <td>1</td> <td>16</td> <td>6</td> <td><i>CAGTAA</i></td> </tr> <tr> <td>1</td> <td>67</td> <td>6</td> <td><i>TGGTGG</i></td> </tr> <tr> <td>1</td> <td>77</td> <td>11</td> <td><i>ACGTAATTGGT</i></td> </tr> <tr> <td>1</td> <td>116</td> <td>8</td> <td><i>ACGTAATG</i></td> </tr> <tr> <td>1</td> <td>126</td> <td>8</td> <td><i>CAGTAATG</i></td> </tr> <tr> <td>1</td> <td>151</td> <td>8</td> <td><i>ATTGGTGG</i></td> </tr> <tr> <td>2</td> <td>23</td> <td>6</td> <td><i>TGACTG</i></td> </tr> <tr> <td>2</td> <td>139</td> <td>6</td> <td><i>TGACTG</i></td> </tr> </tbody> </table> | | | | Class | Coordinate | Length | Sequence | 1 | 16 | 6 | <i>CAGTAA</i> | 1 | 67 | 6 | <i>TGGTGG</i> | 1 | 77 | 11 | <i>ACGTAATTGGT</i> | 1 | 116 | 8 | <i>ACGTAATG</i> | 1 | 126 | 8 | <i>CAGTAATG</i> | 1 | 151 | 8 | <i>ATTGGTGG</i> | 2 | 23 | 6 | <i>TGACTG</i> | 2 | 139 | 6 | <i>TGACTG</i> |
| Class | Coordinate | Length | Sequence | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 1 | 16 | 6 | <i>CAGTAA</i> | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 1 | 67 | 6 | <i>TGGTGG</i> | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 1 | 77 | 11 | <i>ACGTAATTGGT</i> | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 1 | 116 | 8 | <i>ACGTAATG</i> | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 1 | 126 | 8 | <i>CAGTAATG</i> | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 1 | 151 | 8 | <i>ATTGGTGG</i> | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 2 | 23 | 6 | <i>TGACTG</i> | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 2 | 139 | 6 | <i>TGACTG</i> | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

| | | | |
|---|----|---|----------|
| 3 | 38 | 8 | CTGAAGAG |
| 3 | 47 | 8 | CTGAAGAG |

**this table is taken from the author's another research paper [16]*

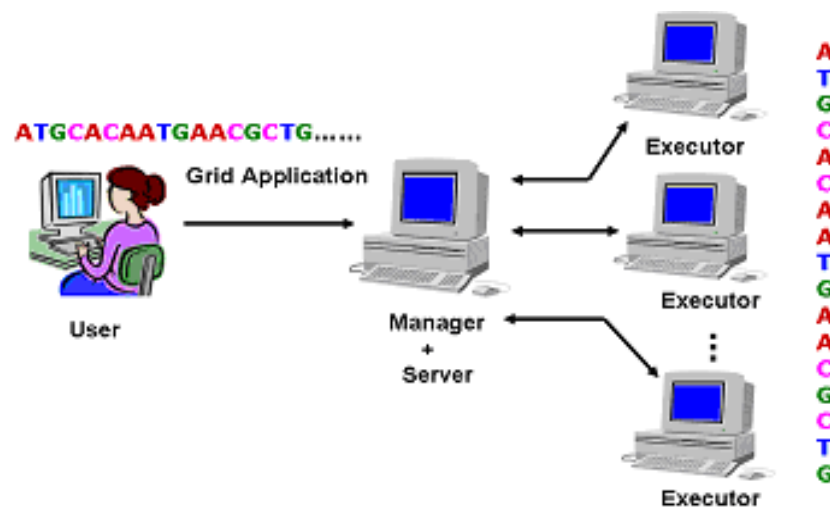


Figure.14 Components of Grid System

Manager

The Manager provides services associated with managing execution of grid applications (Bio-Informatics processing) and their constituent threads [9][7]. Executors register themselves with the Manager, which in turn monitors their status.

Executor

The Executor accepts thread from the Manager and executes them. Each thread performs the searching of a pattern in the partitions [9]. For large sequence of pattern, the scheduling can be performed in two steps. In the first step, the patterns to be searched are distributed to the some Executor nodes.

In the second step, these Executors perform another scheduling as Manager and thus each pattern is searched on a single partition on a single executor. Here I use only one step scheduling for implementation. Alchemi also sets up executors.

Grid Server

The Server is a user written program that uses an API provided by the Alchemi developers. We used C# for our research application; also C++ API is available

for such work. The activities of server also includes –

- 1) Partitioning depending on the length
- 2) Decision making about the patterns of different length
- 3) Checking terminating conditions
- 4) Saving repeated patterns of interests.
- 5) Tells each Executor which partitions to search for the pattern.

Here I use the same node as Manager and Grid Server.

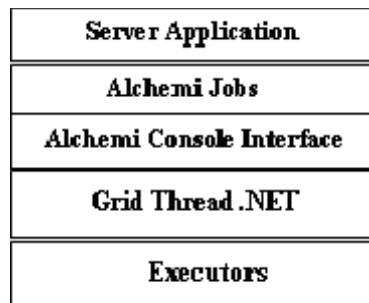


Figure.15 Structure of Operation

3.2.4 How the System Works

Server takes the input sequence and handles the partitioning depending on the length of the sequence. The queries of all same length patterns are performed in parallel manner on different nodes called Alchemi Executors.

This distribution is the responsibility of Manager. After all the query of n length pattern, the results are returned to the server for decision about the (n+1) length patterns. This continues until the termination conditions for searching the repetition occur. The structure of figure 16 can best explain the operation.

3.2.5 Parallel Computing and Performance Comparison

As the algorithm I have implemented here can run parallelly on a system, the computational time complexity can be reduced drastically depending on the number of nodes on the Grid (or heterogeneous Distributed) environment. A simple mathematical calculation of parallelism shows that the execution time will be improved by a factor n for n number of nodes. Here I consider only a single level of parallelism. This is as follows:

Table. 2 Number of Executors Vs Execution Time

| No of Executors | Total CPU (GHz) | Execution Time (Sec) |
|-----------------|-----------------|----------------------|
| 1 | 1.7 | 356.906 |
| 2 | 3.41 | 186.797 |
| 3 | 4.905 | 146.047 |
| 4 | 6.408 | 134.578 |
| 5 | 8.102 | 125.407 |

Let, for the matching of Patten of length I,

No of Nodes = n (all are same powerful)

Number of Patterns to be Searched = 4^I (Worst case, all pattern of length I to be Searched)

Time for each matching of length I = t

Now, if only one CPU/computational node is available then

The Total time required for finding repetition, $T = 4^I * t$;

But, if n Computational Nodes are available in a Grid environment then,

Total time required = Ceiling $[4^I / n] * t$;

This is obviously a great improvement and my implementation result shows the similar improvement curve. For different number of Executors the searching time for the finding repeated patterns for the sequence described in the example (table 1) is shown in the table 2 and the related graph is on Fig. 16. Thus the implementation result follows the mathematical calculation and the curve shows the similar slope.

Also I have introduced two improvements over the brute force methods here. One is that the number of patterns to be searched is determined incrementally from the lower length patterns.

And secondly, I don't need to search all the partition for repetition and even not all the coordinates of a partition. This even improves the computational complexity on a single node.

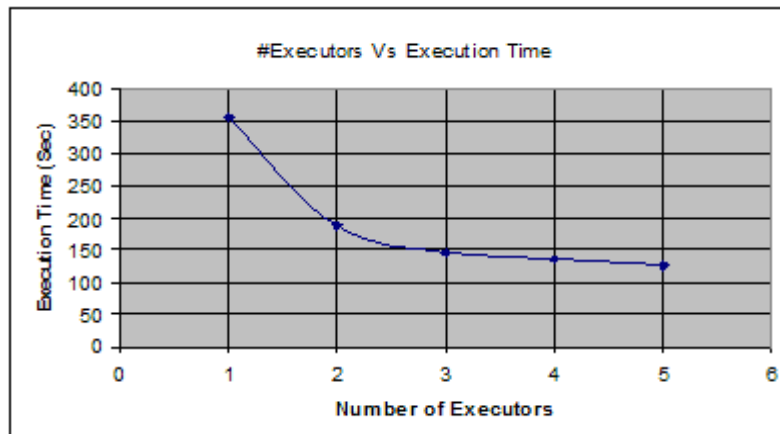


Figure.16 Number of Executors Vs Execution time

The next paragraph is about my recent research output. Here I have used Alchemi in building National Address Database (NAD) on the Grid for a country. The output of this research work shows the stronger power of Alchemi in GRID. Many developing countries can think of such improvement in country's database. Such data then can be available to Map, Mobile, PDA, GPS. I have built a testbed and shown the performance of improved NAD using Grid. Lets talk about the details of this scheme.

3.3 NAD on the Grid by Alchemi

Address has immense importance to every citizen. This has led to the creation of National Address Database (NAD), which will allow every place to have a valid and verifiable address and also allow diversified services to be provided to every citizen. Traditionally, national address databases have been built and maintained at a single central location. Centralized systems of this sort have inherent drawbacks: single point of failure, congestion and low scalability. To overcome these flaws my research focus is to develop a system with NAD in a grid environment. I have implemented NAD in both centralized and grid environment and done a comparative study between them. And to build the Grid, I have used Alchemi as before. Also in this research, I have used Intiempo Server as the Database node developed by SDSL (Structured Data System Limited) with their prior permission.

3.3.1 Proposed System Architecture

I have built the system using 3-tier architecture. The figure 17 shows the pictorial description of this 3-tier architecture:

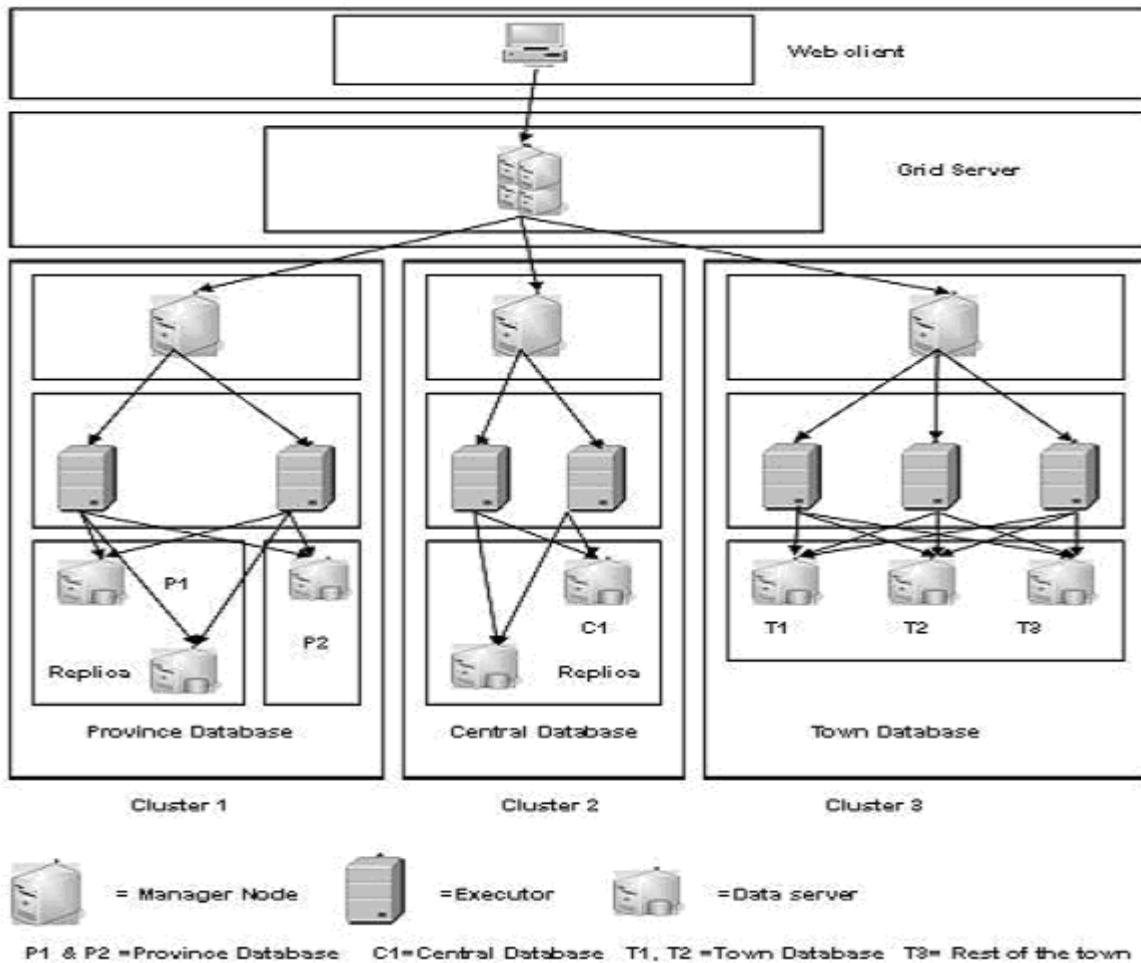


Figure.17 Proposed System Architecture

Tier-1: Consists of two portions. One is Grid Server or entry portal another is manager node, though it is possible to place them on different workstations.

Tier-2: Consists of several executor nodes. These PCs can be located any part of the world even can be users connected to Internet. Thus we can use the unused processing time of worldwide Internet users. or better performance, I have defined the executor architecture using Cluster Grid.

Tier -3: Contains data server or database node. Here the database is Intiendoo server, developed by SDSL (AfriGIS Bangladesh). This Server reads Data from Generic Hierarchy database of National Address.

3.3.2 Components Of System Architecture

Different types of nodes (or hosts) take part in desktop grid construction and application execution. Deploying a Manager node and deploying one or more Executor nodes configured to connect to the Manager construct an Alchemi desktop grid. I have defined Alchemi Manager and Executors in the following ways.

3.3.2.1 Manager

Here my Client sends request to the Manager through a middleware (Grid Server) using a web based user interface (Web Client) [5]. This node then distributes the jobs among the working executors.

3.3.2.2 Executor

The Executors/Working Nodes accept threads and execute them They are actually responsible for requesting the data to the DataServer nodes and processing the data after getting them from the specified database [9].

3.3.2.3 Data Server Node

The DataServer node actually accesses the database. In the system I have used Intiando server as a DataServer node, which access a generic hierarchy database to retrieve data.

3.3.2.4 Web Client

Web client is a web page, which provides the user interface from where the client can send the address verification request (Match Address). The "Match Address" request in the web client contains five fields (province, Town, Suburb, Street name, Street no.). By filling any of these fields the client can send his/her request for address verification. Upon receiving the request the system searches through the database to find a match with the field specified in the request. The system displays all the possible matched addresses with their matching percentage as a response. The data provided by the client can be miss-spelled, wrong or unformatted.

3.3.2.5 Grid Server

The request from the client will be directed to the grid server. I have developed this grid server as a user defined middleware that is responsible for creating threads for each of the request sent by the client. Depending on the type of the request grid server will select the manager from a particular cluster. Then the threads created by the grid server will be sent to that manager to make scheduling, so that each thread (user request) will be assigned to a specific executor.

3.3.3 Cluster Specification

Here I have introduced several clusters on this Grid Environment. The request type from the end users differentiates these clusters.

Cluster 1- If a user sends request by specifying the province name then the grid server will select manager of this cluster which in turn schedule the threads for each request to a specific executor to retrieve data from the data server node. Here I have assumed that there are two provinces P1 and P2 and P1 is mostly searched database. So I have created a replica of P1 to reduce the load.

Cluster 2- If province and town name is not specified in the request then this cluster will be selected, which has central (whole database) NAD and also a replica of the database to reduce the load.

Cluster 3- If town name is specified in the request then this cluster will be selected. Here I have assumed that T1 and T2 are the town databases, which are mostly searched. If town name in the request matches with T1 or T2, data will be retrieved from the matched database. Otherwise data will be retrieved from T3, which is a database for rest of the towns.

3.3.4 System Operation

Clients will send request through web client to the Grid Server. Grid Server will create threads for each of the requests, selects manager of a particular cluster depending on the request and instruct manager to generate the response. The manager will handle these requests by scheduling the threads. Each of these threads will be assigned to an executor. Executor will retrieve data from Intiendo server. So the steps are the following:

1. Parameters for the request will be converted into CSV (comma separated value) format.
2. Manager will send this request format to executor.
3. Grid Server will provide Manager Node with a table for the location of the DataServer node and their database content.
4. The table along with the request will be provided to the executor by manger node.
5. Now executor will decide which database will be required for this request and request will be sent to that data server.

6. Data server will retrieve response from the database.
7. The path through which the response will be sent to the client is as same as the request.

3.3.5 Functional Specification Of The Testbed

For the clear understanding, here I am introducing the functional specification of the testbed. This specification explains the physical distribution and configuration of each node. The following nodes will form part of our NAD on the Grid.

I have defined three levels for the nodes.

Level 1 Node: This type of node is up and available 100% of the time.

Level 2 Node: This type of node is up and available most of the time, say 85% up-time.

Level 3 Node: This type of node only has dial-up access to the grid.

Node 1

Level: 1

Type: Web client

Operating System: Windows

Grid software: N/A

Street Address Data: None

Data Format: CSV (Comma Separated Value) format for Intiempo Server

IP Address: 203.208.189.69(Internet)

Node 2

Level: 1

Type: Grid Server (User defined middleware)

Operating System: Windows

Grid software: Alchemi

Street Address Data: None

Data Format: CSV (Comma Separated Value) format for Intiempo Server

IP Address: 172.16.24.3(LAN)

Node 3, 4, 5

Level: 1

Type: Manager Node of cluster 1, 2 and 3

Operating System: Windows

Grid software: Alchemi

Street Address Data: None

Data Format: N/A

IP Address: 172.16.24.5(LAN), 172.16.24.122(LAN),
172.16.24.129(LAN)

Node 6,7,8,9,10,11,12

Level: 1 or 2 or 3

Type: Executor Nodes of cluster 1, 2 and 3 connected to corresponding Managers.

Operating System: Windows

Grid software: Alchemi

Street Address Data: None

Data Format: N/A

IP Address: 172.16.24.5(LAN), 172.16.24.122(LAN),
172.16.24.129(LAN), 172.16.24.3(LAN),
172.16.24.161(LAN), 172.16.24.162(LAN)
172.16.24.149(LAN)

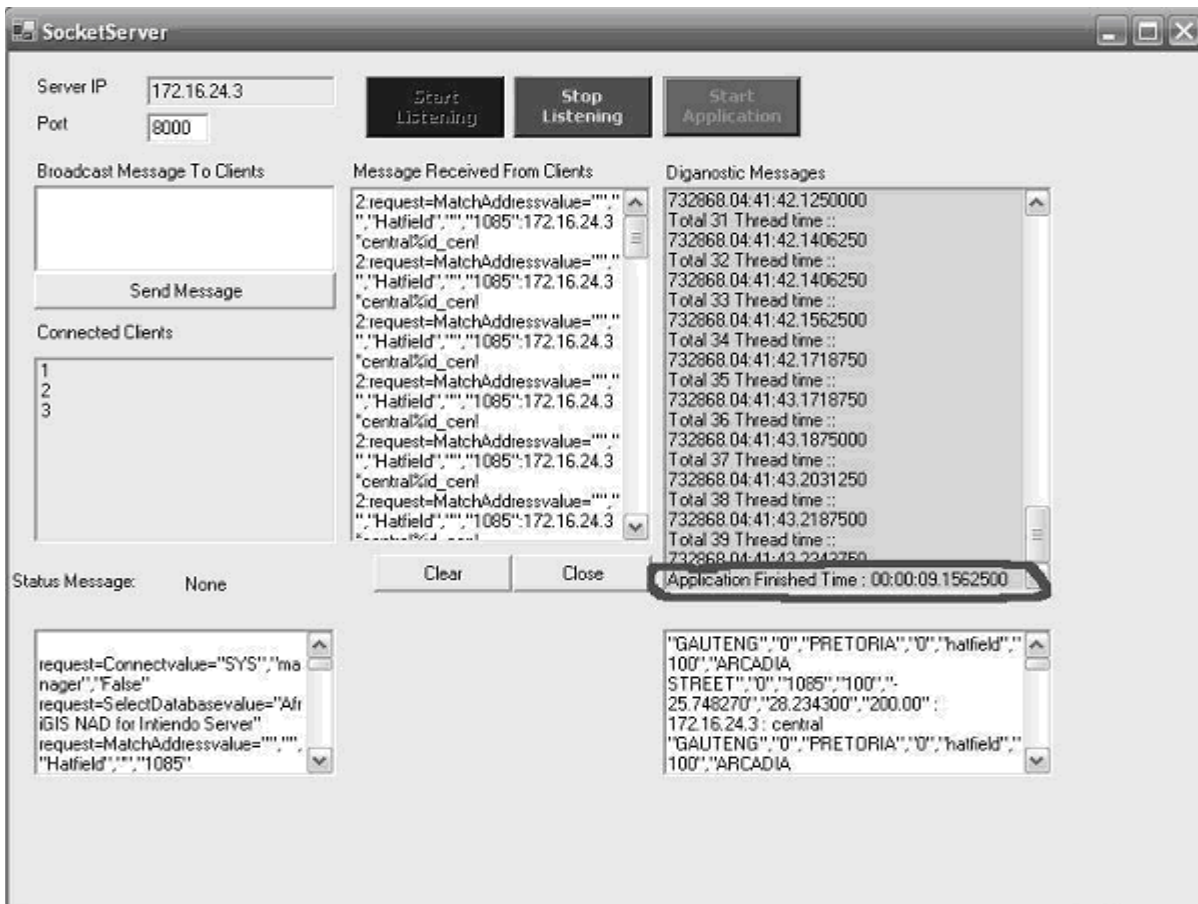


Figure.18 Snap shot of Grid Server during execution on Cluster Grid with 40 requests.

Node 13, 14, 15, 16, 17, 18, 19, 20

Level: 1

Type: DataServer Nodes of cluster 1, 2 and 3

Operating System: Windows

Grid software: Alchemi

Street Address Data: Street Address database of South Africa is used (Demo)

P1=Gauteng (province), P2=Western Cape (province)

C1= Demo version of the whole South Africa Street Address db

T1=Pretoria(Town), T2=Lawley(Town), T3=Rest of the town db

Data Format: Intiendo Hierarchy

IP Address: 172.16.24.5(LAN), 172.16.24.122(LAN)
 172.16.24.129(LAN), 172.16.24.3(LAN),
 172.16.24.161(LAN), 172.16.24.162(LAN),
 172.16.24.149(LAN), 172.16.24.150(LAN).

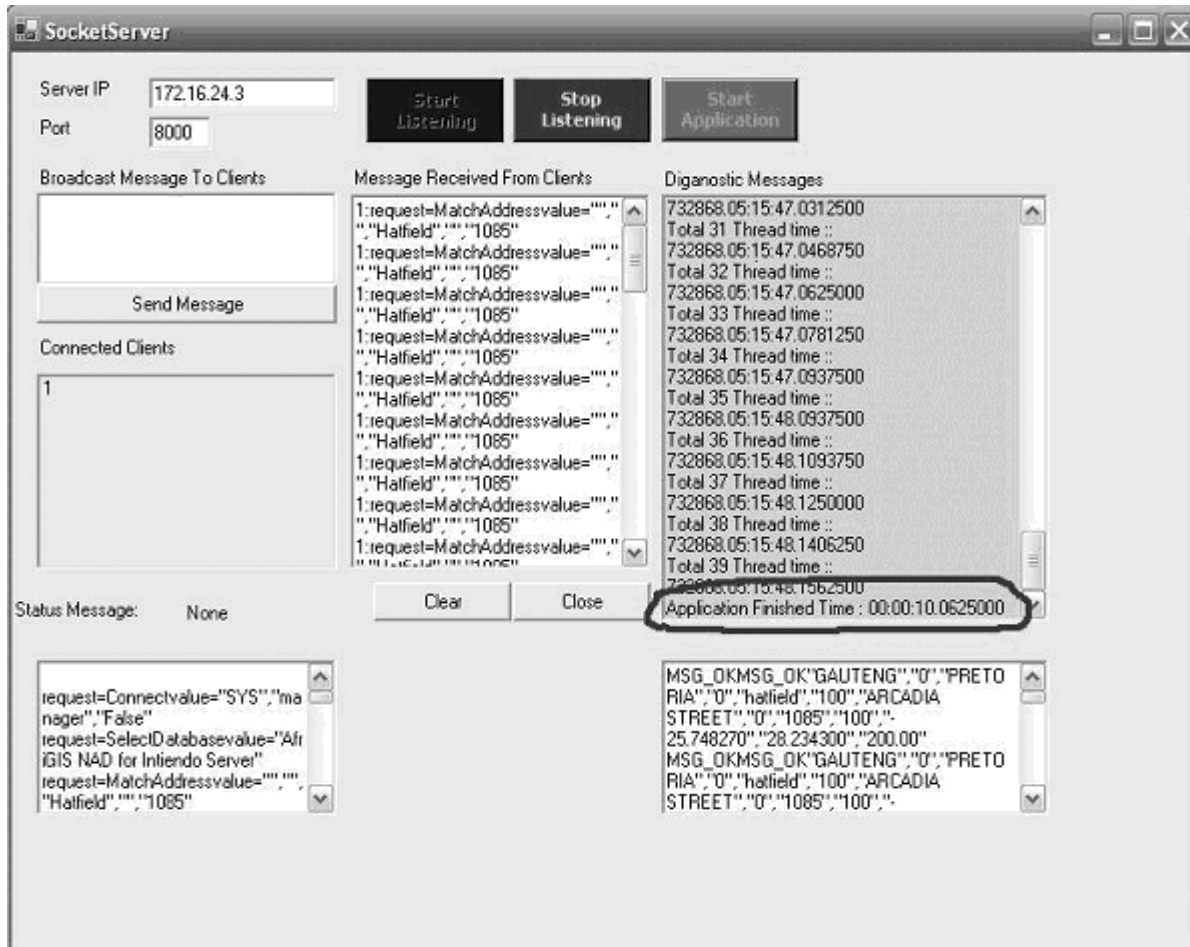


Figure.19 Snap shot of Grid Server during execution on Centralized System with 40 requests.

3.3.6 Results and Performance Evaluation

The test bed for grid environment is prepared and then to measure the performance for Match Address service I have created specific set of requests and sent those requests simultaneously to both centralized system and our proposed system in grid environment. Here I have sent 20 different requests to both centralized and grid system and documented the response time. This procedure has done three times and from these data average response time is calculated. Analogously for 30 and 40 requests the same procedure is followed. The snap shot of out output is given at figure 18 and figure 19. The following table contains the response times for different requests.

Table.3 Response Time Table

| No of Request (REQ) | Response Time in centralized Environment | | | | Response Time in Cluster Grid Environment | | | |
|---------------------|--|--------|--------|---------|---|--------|--------|---------|
| | TEST 1 | TEST 2 | TEST 3 | Average | TEST 1 | TEST 2 | TEST 3 | Average |
| 20 | 5.6562 | 6.6718 | 5.6406 | 5.9895 | 5.6093 | 5.3437 | 5.3593 | 5.4375 |
| 30 | 6.75 | 6.59 | 6.43 | 6.59 | 5.8281 | 5.9001 | 5.7561 | 5.8281 |
| 40 | 10.071 | 10.033 | 10.084 | 10.063 | 9.1562 | 9.1562 | 9.1561 | 9.1562 |

From this table by plotting average response time and no of request, a graph (figure 20) is constructed to visualize the performance.

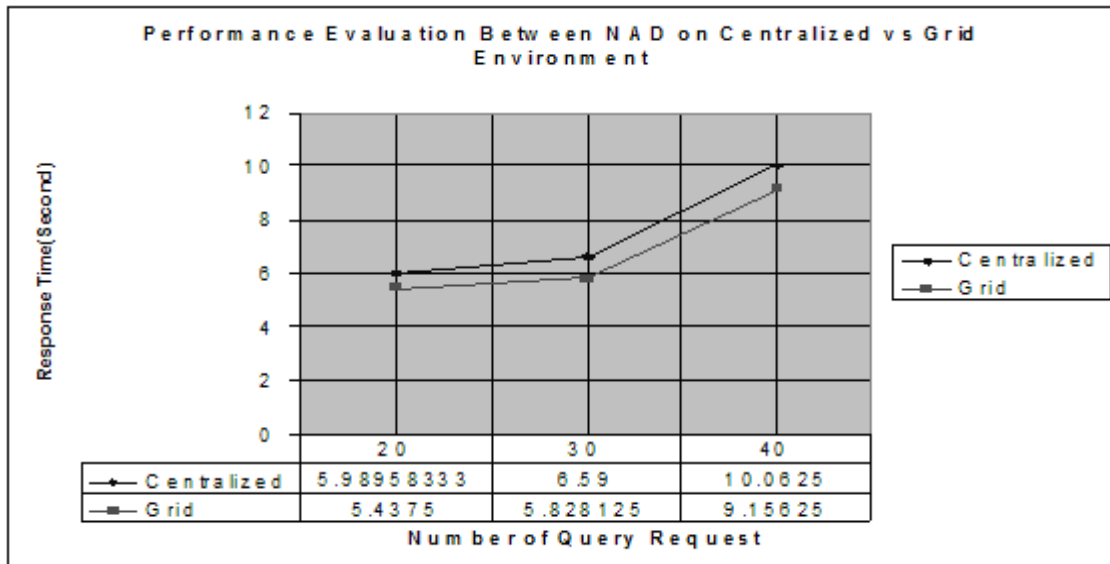


Figure.20 Performance Graph (Centralized vs. Grid)

From the figure, it is clearly understandable that my NAD on the Grid can provide much better performance than any available NAD systems in any country. The curve for Cluster Grid is always better than the curve for centralized NAD implementation.

I can conclude from this experimentation result that the performance of the grid system is better than the centralized system. Though many countries are currently using NAD on various ways, they can ensure better performance if they use this architecture. For the limitation of resources I have performed the experiments with small number of requests and small number of nodes. This

experiment can infer that for larger number of requests the reduction of the response time in grid system will be much more impressive.

Next, I am going to reference another common problem in today's Internet world – "Spam in email server". To solve this problem, I have used another open source software- "Globus Toolkit", which is widely used in Worldwide Grid Environment.

3.4 Globus in Spam Protection

The other important area where the grid software can be used is in spam protection. Here, I have just provided the architecture of such design and defined how it works.

E-mail being one of the most important applications available via the Internet is also one of the most vulnerable one. The flood of spam brings about a severe problem to the effectiveness and performance of this service. According to a statistic made by Brightmail Probe Network [19], spam accounted for 62% of all Internet e-mail in February 2005. Lately the effect of spam is getting more and more annoying up to the point where it affects people's everyday lives. The need for a solution to eliminate spam was obvious and thus many people started thinking and implementing ideas to fight junk e-mails. Although some of the methods have been successful in diminishing the amounts of e-mails, none of them has managed to eliminate them completely and still assure no loss of legitimate e-mails.

There are different anti-spam techniques available, some of these are assigning mail-server blacklist, signature based filtering, filtering, rule based filtering, challenge response filtering. We use similar to Bayesian filtering techniques in our Globus environment.

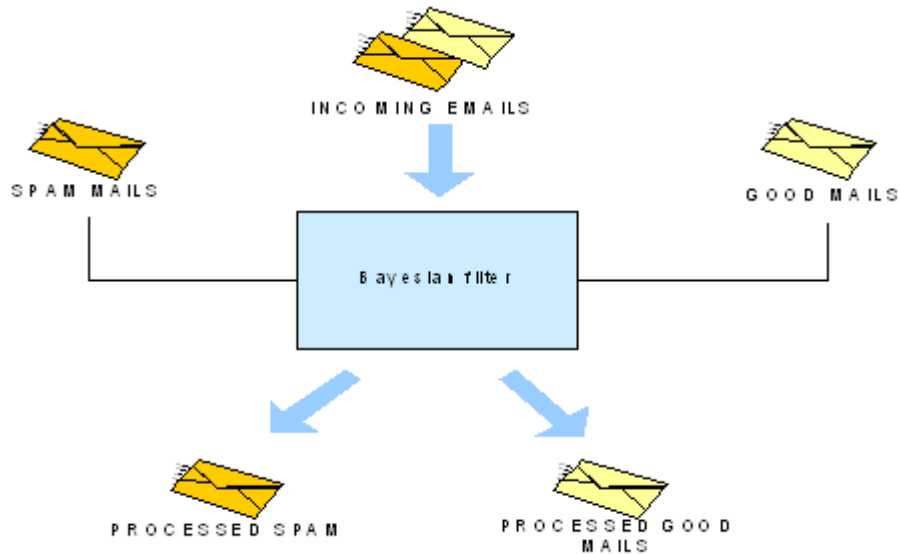


Figure.21 Bayesian spam filtering

Bayesian spam filters calculate the probability of a message being spam based on its contents. Unlike simple content-based filters, Bayesian spam filtering learns from spam and from good mail, resulting in a very robust, adapting and efficient anti-spam approach that returns hardly any false positives.

Since the weakness of scoring filters is in manually building the list of characteristics, Bayesian spam filters build the lists themselves. I start with a big bunch of e-mails that I have classified as spam, and another bunch of good mails as shown in the figure 1, below. The filters look at both, and analyze the legitimate mail as well as the spam to calculate the probability of various characteristics appearing in spam, and in good mail.

The characteristics of a Bayesian spam filter can look at can be

- The words in the body of the message
- Mail headers
- Included HTML codes (colors)
- Word pairs, phrases
- Meta information (where a particular phrase appears)

3.4.1 Overview of Globus Toolkit

The open source Globus Toolkit is a fundamental enabling technology for the "Grid," letting people share computing power, databases, and other tools securely online across corporate, institutional, and geographic boundaries without sacrificing local autonomy [www.globus.org]. The toolkit includes software services and libraries for resource monitoring, discovery, and management, plus security and file management. In addition to being a central part of science and engineering projects that total nearly a half-billion dollars internationally, the Globus Toolkit is a substrate on which leading IT companies are building significant commercial Grid products [ww.globus.org].

The toolkit includes software for security, information infrastructure, resource management, data management, communication, fault detection, and portability. It is packaged as a set of components that can be used either independently or together to develop applications. Every organization has unique modes of operation, and collaboration between multiple organizations is hindered by incompatibility of resources such as data archives, computers, and networks. The Globus Toolkit was conceived to remove obstacles that prevent seamless collaboration. Its core services, interfaces and protocols allow users to access remote resources as if they were located within their own machine room while simultaneously preserving local control over who can use resources and when.

The Globus Toolkit has grown through an open-source strategy similar to the Linux operating system's, and distinct from proprietary attempts at resource-sharing software. This encourages broader, more rapid adoption and leads to greater technical innovation, as the open-source community provides continual enhancements to the product.

There are several pre-requisite software for Globus. These are Java 1.4.2 or higher, Ant 1.5+, C compiler, Gnu make/tar, jdbc complaint database (postgres), and junit. First we need to install java, Ant, and Junit. Then a globus account should be created in Linux (as I use Linux, but globus also works in windows with limited services). We have to specify GLOBUS_HOME, JAVA_HOME and ANT_HOME environment variables. To install Globus, next we need to run `./configure - prefix=/usr/local/gt4.0.1` and then run the make file.

Communication among different globus nodes are performed using Virtual Organization (VO). Several nodes can be defined as upstream nodes and several are downstream. Index Service is used to circulate the services of one node to another using VO.



Figure.22 Architectur of Globus Tolkit

3.4.2 System Security Offered by Globus

The basic security components within the Globus Toolkit provide the mechanisms for authentication, authorization, and confidentiality of communication between grid computers. Without this functionality, the integrity and confidentiality of the data processed within the grid would be at risk.

There are many different tools and technologies available to secure grid environment in Globus [11, 13]. Grid Security Infrastructure (GSI) of the Globus Toolkit maintains the security issue. Along with the different responsibilities

associated with securing a grid environment, there are many risks involved. While the Grid Security Infrastructure (GSI) components make it easy to install and configure [9].

3.4.3 Licensing of Globus Toolkit

The Globus Alliance is committed to maintaining a liberal, open source license. The Globus Toolkit Public License (GTPL) allows software to be used by anyone and for any purpose, without restriction. They believe that this is the best way to ensure that Grid technologies gain wide spread acceptance and benefit from a large developer community.

3.4.4 How The System Works

The Grid technology provides the features and services to implement the solution to fight against spam. The following facts about spam and current spammer behaviour suggest us to affirm this belief.

- Spam is delivered globally
- A central control system to identify and eliminate spam will not be sufficient
- The nature of the spam is unpredictable (when, how, quantity)
- An infrastructure where we can attain dynamic computing resources is required

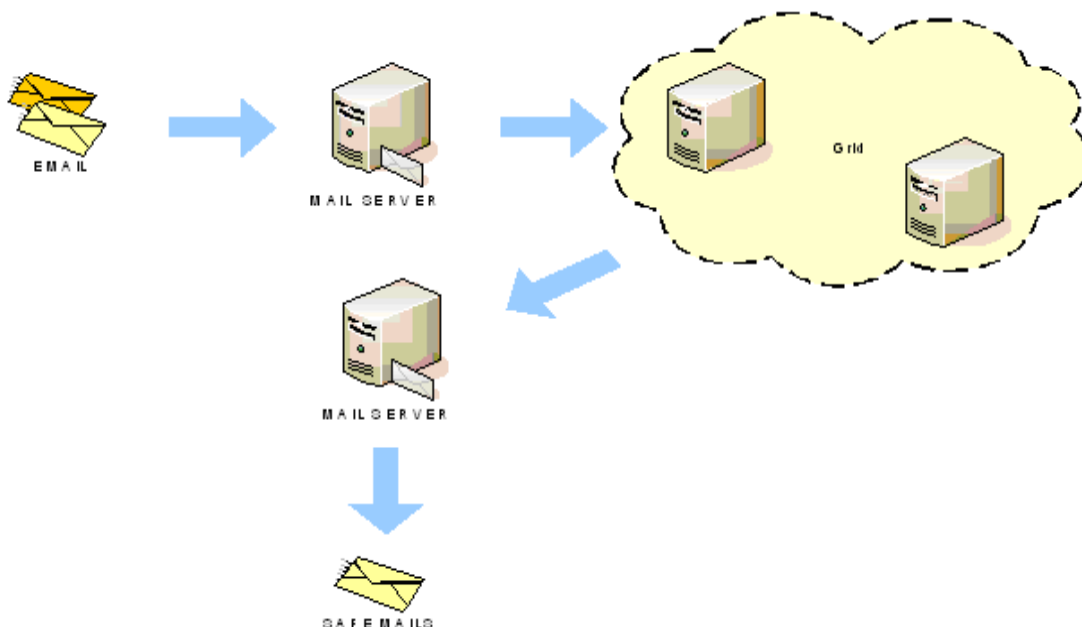


Figure.23 Design goal for Spam Filtering with Globus

An implementation overview of my design middleware is presented in Figure 23. It consists of mail servers, which process normal Internet e-mail as usual, which makes use of the Grid environment to process the incoming spam. In this design the main advantage point is that I am fighting against spam messages as soon as it is fed into the network by the spammer. As soon as the spammer has sent the mails, the spam identification and elimination process can be started, rather than having to wait for the clients to download the spam messages. The client side filters, used by much mail application software, allow the user to download all the mails, and starts identification of spam at the client side. From our design the user benefits from the fact that he does not have to download the spam mails along with his legitimate mails.

By Globus, Grid environment will run Grid Resource Information Service (GRIS) and publish their services in the Grid Information Index Service (GIIS) that runs on Monitoring and Discovering Service (MDS) server.

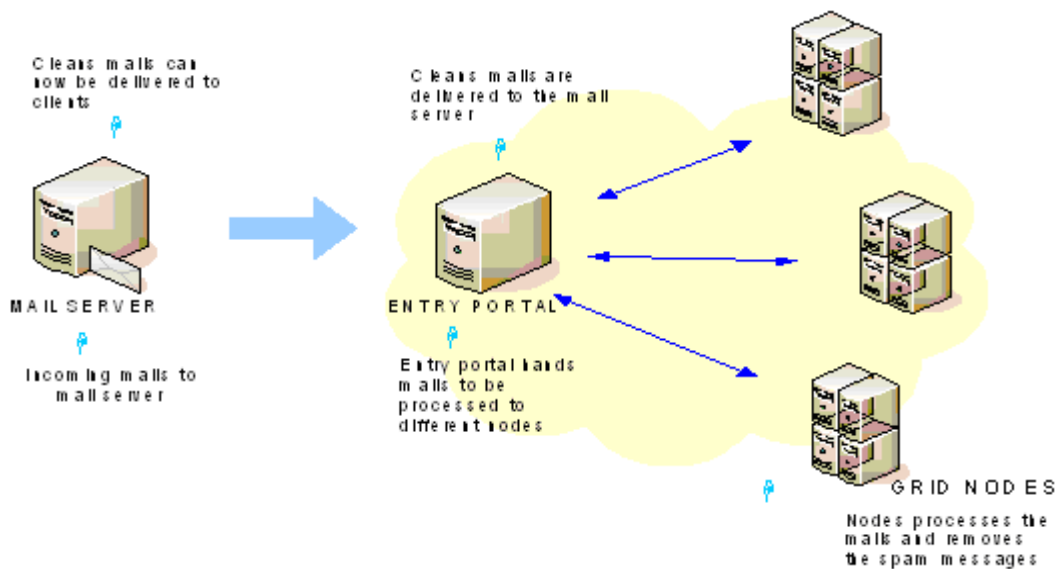


Figure.24 Design architecture for Spam Filtering using Globus

When a mail server receives e-mail to be delivered, it will access the Grid Resource Access and Management (GRAM) server in Globus node, which will query the MDS server and then choose a server near the client (mail server is the client here) and inform it. The query replies to the mail server indicating that it is running the Spam Check Service. Portions of the mail that needs to be processed for spam then will be transferred to the available nodes via GriFTP. Individual grid nodes will be processing portions of the mails but they all will follow the same algorithms and codes. Grid nodes will process and identify the spam mails,

which we can choose to eliminate, based on the success of our filtering mechanism. After processing, the nodes will transfer the clean mails to the mail server. Now the mail server can deliver the mails in the usual way. The overall process is shown in the figure 24.

4. Future Works in FOSS

With the demand of FOSS in scientific applications, world leaders are now realising their needs for developing countries. Most of the people and organizations of developing countries cannot afford highly expensive software. But current technological world cannot go faster without the use of modern technology and software. That's why we need to introduce FOSS especially in developing countries. Recently, UNDP and Internet Society of Bulgaria have launched a project to help municipal government in South-eastern Europe use the Internet to better respond to citizen's needs. This is the first e-government project in the region of use Free/Open Source Software application to enhance government transparency and people's access to municipal services. Also University of Illinois at Urbana Champaign is developing Lightweight OS for Sensor Network. It is also promising that some organizations are working on FOSS in medical science which will reduce the treatment cost and improve medical research with worldwide contributions. Recently Sun Microsystems inc. has released their open source Hardware, which is a great initiative. According to them, they want worldwide programmers to contribute on the testing, bug identification and modifications. It will be a great steps for the world if other big companies come forward with their tools as open source for different scientific applications.

5. Acknowledgement

Some parts of this paper are the research output with my different colleagues and students. I would like to thank Abu Wasif, Abdullah Ibney Anwar for the research of using FOSS in Bio-Informatics Application. Also my students Yamin, Nafiz, Sadim and Hussain have helped me for setting of the Grid enviroment and gathering results for NAD project and Spam Filtering project. The materials of Alchemi are available at www.alchemi.net and the materials of Globus Toolkit are available at www.globus.org. All the descriptions of these open source software are taken from the respective sites.

6. Conclusion

All the designs, implementations and results those are described in this paper are completely based on the open source software that I have used for different scientific applications. The design and results are so promising that people are now interested on the use of such Open Source in their scientific applications. Developing countries can use such NAD on their country's database. Also the use of FOSS in Bio-informatics and Spam protection are good examples of scientific success using low cost techniques. My main success here is to build a parallel environment using different Open Source Software and gridifying different applications suitable for such environment. These are small areas and still we have more scientific areas where we can introduce the use of different such Open Source Software. In fact, FOSS can be used in our every part of technological/scientific movements and applications.

References

- [1] Open Source Initiative, "The Open Source Definition, " *Open Source Initiative*, July, 07, 2006. [Online]. Available at <http://www.opensource.org/>. [Accessed: Aug, 29, 2007].
- [2] M. Y. Leung. B. E. Blaisdell, & C. K. Burfe, "An Efficient Algorithm for Identifying Matches with Error in Multiple Long Molecular Sequence". *J Mol Bio*, vol. 221, pp.1367 – 1378 ,1991.
- [3] Gridcafe, "The place for everybody to learn about Grid," *Gridcafe*. [Online]. <http://gridcafe.web.cern.ch/gridcafe/>. [Accessed: May, 07, 2007].
- [4] L. Smarr and C. Catlett, "*Metacomputing and Communications*, ACM Press, pp.44-52,1992.
- [5] P. Cappello, B. Christiansen, M. F. Ionescu, M. O. Neary, K. E. Schauer, and D. W. Javelin, *ACM Workshop on Java for Science and Engineering Computation*, pp.323-328,1997.
- [6] A. Luther, R. Buyya, R. Ranjan, and S. Venugopal, "Peer-to-Peer Grid Computing and a .NET-based Alchemi Framework", *Conference on High Performance Computing: Paradigm and Infrastructure*, pp.403-429

,New Jersey, USA ,Jun. 2005.

- [7] I. Foster & C. Kesselman, "The Grid: Blueprint for a Future Computing Infrastructure" (Mogan Kaufmann Publishers, 1999).
- [8] I. Foster, C. Kesselman, and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations", *International Journal of Supercomputer Applications, Sage Publications*, vol.15(3), pp.112-116, 2001.
- [9] Ian Foster and Carl Kesselman, "Globus: A Metacomputing Infrastructure Toolkit", *International Journal of Supercomputer Applications*, vol.11(2), pp.15-128, 1997.
- [10] W. Li, "New Stopping Criteria for Segmenting DNA Sequences", *Physical Review Letters*, vol.86, pp.5815—5818, jun. 2001.
- [11] G. D. Stormo, "DNA Binding Sites: Representation and Discovery", *Bioinformatics*, vol.16, pp.16-23, 2000.
- [12] M. Gupta and J. S. Liu, "Discovery of Conserved Sequence Patterns Using Stochastic Dictionary Model", *J Am Stat Assoc*, vol.98, pp.55-66,2003.
- [13] N. Volfovsky, B. J. Haas and S. L. Salzberg, "A Clustering Method for Repeat Analysis in DNA Sequences", *Journal of the Institute of Genomic Research*, vol.2(8), pp.27.1 – 27.11, 2001.
- [14] M. A. Arefin, and M. S. Sadik, "Implementation of Server on Grid System: A Super Computer Approach," *15th International Conference on Information System and Development. (ISD, 06)*, Budapest, Hungary, Aug, 2006. Published in SpringerLink Book "*Advances in Information Systems Development*," ISBN: 978-0-387-70760-0. [Print] 978-0-387-70761-7.[Online], pages: 225-236, USA, 2007.
- [15] C.J. Bult, O White, G.L.Olsen, L. Zhou, R.D. Fleischmann, G.G.Sutton, A.J.Blake, R.A.Clayton, and J.D.Gocayne, "Complete Genome Sequence of the Methanogenic Archaeon, *Methanococcus Jannachii*", pp.1058-1073. 1998.
- [16] M. A. Arefin, A. I. Anwar, and A. Wasif, "Finding Repeated Pattern in DNA

Sequences with Parallel Algorithm on Grid," *18th IASTED conference of Parallel and Distributed Computing and Systems (PDCS, 06)*, UTDallas, Texas, USA, Nov, 2006.

- [17] S.K. Kannan, and E.W. Myers, "An Algorithm for Locating Nonoverlapping Regions of Maximal Alignment Score", *SIAM J Comput*, vol.25, pp.648-662, 1996.
- [18] R. Sharan, S. Suthram, R. M. Kelly, T. Kuhn, S. McCuine, P. Uetz, T. Sittler, R. M. Karp, and T. Idker, "Conserved Patterns of Protein Interaction in Multiple Species", *Conference of National Academy of Science*, pp. 1974-1979, 2003.
- [19] SpamAssassin, "The Apache SpamAssassin Project," *SpamAssassin* .[Online]. <http://spamassassin.apache.org>. [Accessed: May, 07, 2007].
- [20] S. O. Kurtz, C. Schleiermacher, & J. G. Stoye, "Computation and Visualization of Degenerate Repeats in Complete Genome", *8th International Conference on Intelligent Systems for Molecular Biology*, pp.228-238, 2000.
- [21] Y. Amir, B. Awerbuch, and R. S. Borgstrom, "The Java Market: Transforming the Internet into a Metacomputer", *Technical Report CNDS-98-1*, Johns Hopkins University, 1998.
- [22] Microsoft Corporation, "Microsoft Dot Net Framework," *Microsoft Corporation*. [Online]. Available at <http://msdn.microsoft.com/netframework/>. [Accessed: July, 07, 2007].
- [23] The University of Melbourne, "Alchemi-.NET based Grid Computing Software," *The University of Melbourne*. [Online]. Available at <http://www.alchemi.net>. [Accessed: July, 07, 2007]
- [24] Md. Ahsan Arefin, Md. Shiblee Sadik, Serena Coetzee, Judith Bishop, "Alchemi Vs Globus: A Performance Comparison", *4th International Conference on Electrical and Computer Engineering (ICECE,06)*, BUET, Dhaka, Bangladesh, Dec, 2006.