Grid Architecture Storage - Utilising Grid Computing for Dynamic Data Storage

P. Koszek, K. Sandrasegaran
Institute of Information Sciences and Technology
University of Technology Sydney (UTS)
PO Box 123, BROADWAY, NSW 2007, Australia
Email: phoebe.e.koszek@student.uts.edu.au ; kumbes@eng.uts.edu.au

Abstract

Grid Architecture Storage (GAS) is based on the concept that the Grid Computing Architecture is able to be manipulated in order to facilitate data storage on the Grid. This paper describes the software that has been developed to prove this concept. The software utilises volatile resources on distributed grid nodes – primarily random access memory – to store data dynamically by shifting the data from node to node without transferring the data to persistent storage. A suite of applications that exercise this theory has been developed using the Alchemi .NET Grid Computing framework, showing that the Dynamic Storage Grid is a feasible grid computing application. Users are able to distribute files from their desktop computers across the Internet, benefiting from the unused resources that are available on computers all over the world.

1 Introduction

Grid computing is the utilisation of the "unused" capabilities of a large, ad-hoc collection of compute nodes distributed across a network. The basic premise of a grid is that compute nodes work together to achieve a common goal. There exists in every grid a grid master or a number of grid masters, which is a central point of management for the grid. The grid master does not control the compute nodes within its grid; rather it coordinates the resource usage on each node as demand for those resources arise.

The common notion of grid computing is a means of network computing that harnesses the unused processing cycles of numerous computers to perform useful work; often solving intensive problems that are too large for a single computer to handle. This is referred to as a 'computational grid'. An alternative way of accessing and employing distributed resources on a network is the 'data grid,' which uses a network of potentially geographically separated nodes for the large scale storage of data.

This project investigates and implements a completely new way of viewing network resources. Rather than extending existing grid usage models (that is, distributing computation or storage needs), it uses the network to implement a dynamic store. In this new archetype, data that is "stored" on the Grid is constantly in motion: there is no point at which one can state that the data is resident on any particular node. Data traverses the network nodes, existing temporarily in volatile primary storage (random access memory) before being moved to the next node – memory that would have otherwise been unused.

To elaborate, dynamic data storage does not introduce new hardware technologies, nor any of the costs that would be associated with the acquisition of new hardware. Instead, it is a way of exploiting primary, volatile memory for the retention of persistent data. In the same way that the computational grid paradigm enables large amounts of work to be broken up into smaller tasks and distributed across a network for work to be completed by many computers with spare processing power, GAS involves the fragmentation of data into traceable packets and distributing the fragments across the grid. The grid master will dictate the course of the data movements based on the continually changing availability of grid resources.

2 Background

Although originally intended for advanced science and engineering applications, grid computing has emerged as a paradigm for coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organisations in industry and business. Today, Grids can be used as effective infrastructures for distributed high-performance computing and data processing.

Accompanying the emergence of grid computing applications is the arrival of peer to peer (P2P) networks for sharing distributed resources. Whilst the grid community is generally focused on the aggregation of distributed high-end machines, the P2P community is looking into sharing low-end systems such as PCs connected to the Internet for sharing compute power and contents.

Table 1 – Inefficient Technology Infrastructure (Source: Server World Magazine 2002)

IT Resource	Daytime Utilisation
Windows Servers	< 5%
UNIX Servers	15-20%
Desktops	< 5%

To give an indication of the potential resource availability of computers connected to a network, Table 1 shows the average daytime utilisation of workplace technology infrastructure. The GAS project is focused at desktop computers and it is estimated that the utilisation of these computers could be boosted by 30% (Server World Magazine, 2002).

Proceedings of the Third International Conference on Information Technology and Applications (ICITA'05)
0-7695-2316-1/05 $20.00 © 2005 IEEE

3 System Technical Description

Each of the applications developed as a part of the GAS project (Owner, Manager, Executor and Monitor) has been built using Microsoft Visual Studio 2003, specifically the c# language. The operational requirements for the application suite are the Microsoft .NET Framework Version 1.1 and network connectivity. The Manager requires access to a Microsoft SQL Server database; the database may be local or remote.

Remote communications are all implemented using .NET Remoting over the TCP/IP protocol, which requires the use of two dedicated configurable ports on the Owner and Executor, and one port on each of the Owner and Monitor.

Much of the Grid Computing functionality exhibited by the applications has been developed using the Alchemi .NET Grid Computing Software Development Kit, a project that is under development at the University of Melbourne (Luther et al, 2004).

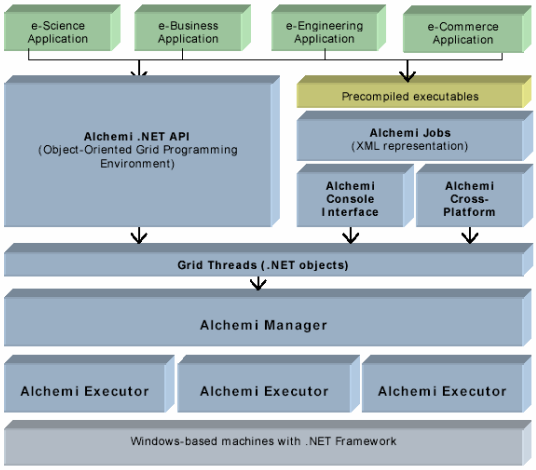

Figure 1 - Architecture of Alchemi SDK

Figure 1 shows Alchemi's layered architecture, which follows the master-worker parallel computing paradigm in which a central component dispatches independent units of parallel execution to workers (compute nodes) and manages them. This unit of parallel execution is referred to as a 'grid thread' and contains the instructions to be executed by the node (Executors). The Manager is the central resource manager on the grid.

The GAS software may be deployed in a variety of networking environments, ranging from local area network, to wide area network, to the Internet.

4 Architecture

The chosen implementation of the Dynamic Storage Grid theory allows a user (typically an Internet user) to export files from their local hard drive to the grid where they are stored dynamically until the user chooses to import the files from the grid and they are restored to the local file system.

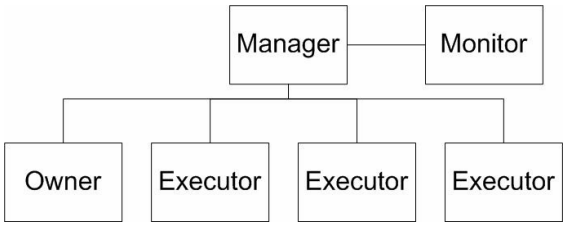

Figure 2 – Basic GAS Grid Configuration

This is facilitated by a suite of four applications as shown in Figure 2:

- Manager – an application that manages the work units (resources) on the grid in terms of accepting requests from Owner applications and distributing work to Executors.
- Owner – an application that runs on the user's work station and enables the exporting of files onto the grid and the import of previously exported files. The Owner application need not be running whilst files are on the grid; it must only be active during file import or export activities.
- Executor – an application that runs on the grid nodes, allowing the node to contribute resources. When resources are available, the Executor connects to the Manager and requests work. If work is available the Manager responds with a grid thread containing the work to be completed.
- Monitor – an application that runs on any machine that is able to connect to the Manager and provides a real time view of grid activity.

When a user chooses to export file(s), they are broken into fragments of a predefined size and allocated unique file fragment identifiers. The Owner connects to the Manager to request the removal of the files and the Manager records the request in the SQL server database.

The next time that an Executor has available resources, it will connect to the Manager and request work. The work is provided in the form of a serialised grid thread which contains information about where the Executor is to retrieve the file from.

The Executor deserialises and executes the grid thread, logging onto the remote Owner using custom FTP libraries and downloading a file fragment. The file fragment is stored in RAM on the Executor while the Executor connects to the Manager to request the creation of a new grid thread so that it may pass the file fragment on to the next Executor.

As another machine running the Executor connects to the Manager to volunteer storage resources, it is allocated the work and connects to the first Executor to download the file. The file is transferred to the second Executor and the RAM on the first is flushed.

Files continue to traverse the grid in this manner until the Owner logs onto the Manager with a request to import the previously exported file. The Manager saves the request and each time an Executor connects to request that another Executor remove the file fragment that it is in possession of, the Manager checks the unique identifier of the Executor's file. If the unique identifier matches that of the request from the

Owner, instead of transferring the file to another Executor, the next Executor to be in possession of the file connects to the Owner and delivers the file using the GAS FTP libraries.

Figure 3 shows the component architecture of the GAS system.

The manager system boundary encapsulates the components that provide services associated with managing execution of grid applications and their constituent threads, and thus the management of resources on the data storage grid. To understand the low-level activities taking place on the grid, the following paragraphs discuss the Manager component in further detail.

Threads received from the Owner and Executors are placed in a pool and scheduled to be executed on available Executors. There are two types of thread that the Manager may receive: export or import.

The file export thread describes the work to be done by an Executor when either the Owner is exporting a file, or an Executor is attempting to pass a file onto another Executor. The file import thread contains information about a file import request and can only originate from the GAS Owner application.

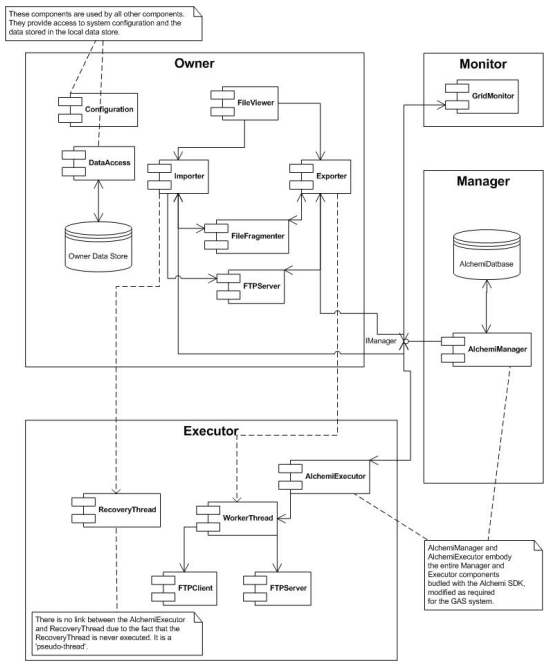

Figure 3 - GAS Component Architecture

Unlike the file export thread, the file import thread is never executed. It should be viewed as a 'pseudo-thread' that is used to facilitate two-way communication between the grid Manager and Owner that would not have been possible using the standard Alchemi grid thread model. The file import thread is saved to a separate application pool which each file fragment is validated against every time work is allocated to an Executor.

The Manager attempts to track possible failures within the grid by monitoring Executors constantly. Upon disconnection of an Executor, any thread that running at the time of disconnection is rescheduled. All data is immediately persisted to disk (SQL server database) to prevent loss of data in the event of problems with the Manager module.

5 Performance

The basic testing environment involved 20 desktop PCs – primarily 2.4GHz Pentium 4 512MB RAM running Windows XP Professional – to serve as Executors and two dual processor servers running Windows Server 2003 to host the Manager and database.

The majority of system components were able to run without failure for periods stretching over a number of days, handling 5 files fragmented into 10=10MB fragments. The exception to this is the Executor application which exhibited a memory leak in version 1.0 of the GAS software. The memory leak does not prevent the Executor from performing work as required but does result in performance degradations over time.

Figure 4 shows typical performance results during Executor operation over a period of 55 minutes, hosting 1 file fragment (for simplicity). The number of Executors connected to the grid was varied over the test duration.

A 1MB file was exported to the grid by an Owner application at 2:00pm with 2 active executors. At 2:10pm a third executor was added. At 2:20pm a fourth executor was added. At 2:38pm a fifth executor was added, and at 2:48pm the file was imported by the Owner.

The graph shows that the rate at which Executors are receiving work is consistent with the number of Executors on the grid – the relationship is linear. The graph is best viewed from the perspective that each spike on the 'megabytes received per second' line is the Executor importing a file, and that each corresponding spike on the 'megabytes sent per second' line is the Executor allowing the file to be downloaded by another Executor on the grid.

The patch of no activity at 2:23pm shows a network dropout on the test network, resulting in the loss of a file fragment. To recover from this, the tester re-exported the same file to the grid (as network errors have not been handled yet in the software design; this has been identified as future work for the project).

The aforementioned memory leak can be observed in the available Mbytes line on the graph in Figure 4. At approximately 2:30pm, the available memory nears zero and the operating system takes action to make more memory available to the running applications (the Executor application). From this point forward, memory usage remains fairly consistent as it is managed by the OS. The Pages/sec trend line shows the operating system activity as it maintains the memory availability.

Over the duration of the test, the Manager application performed extremely well, proving to be very robust and reliable. The Owner and Monitor met system functional and behavioural requirements. Performance evaluation of these applications is not

discussed here as their involvement in overall activity conducted on the grid is minimal.

The key system characteristics to note as an output from performance evaluation are:

- The number of Executors limits the number of file fragments that may be exported to the grid due to the 'handover' nature of the thread model used. There must be two executors for each file fragment. This limitation is to be removed as a part of future work.
- The amount of RAM on a grid node determines the size of the fragment that a node can traffic. The current design requires that the entire file fragment be loaded into memory before a request can be placed to transfer the file to another Executor.
- The file fragment size and available network bandwidth together determine the overall throughput of file fragments. That is, the rate that an Executor is able to process grid threads.

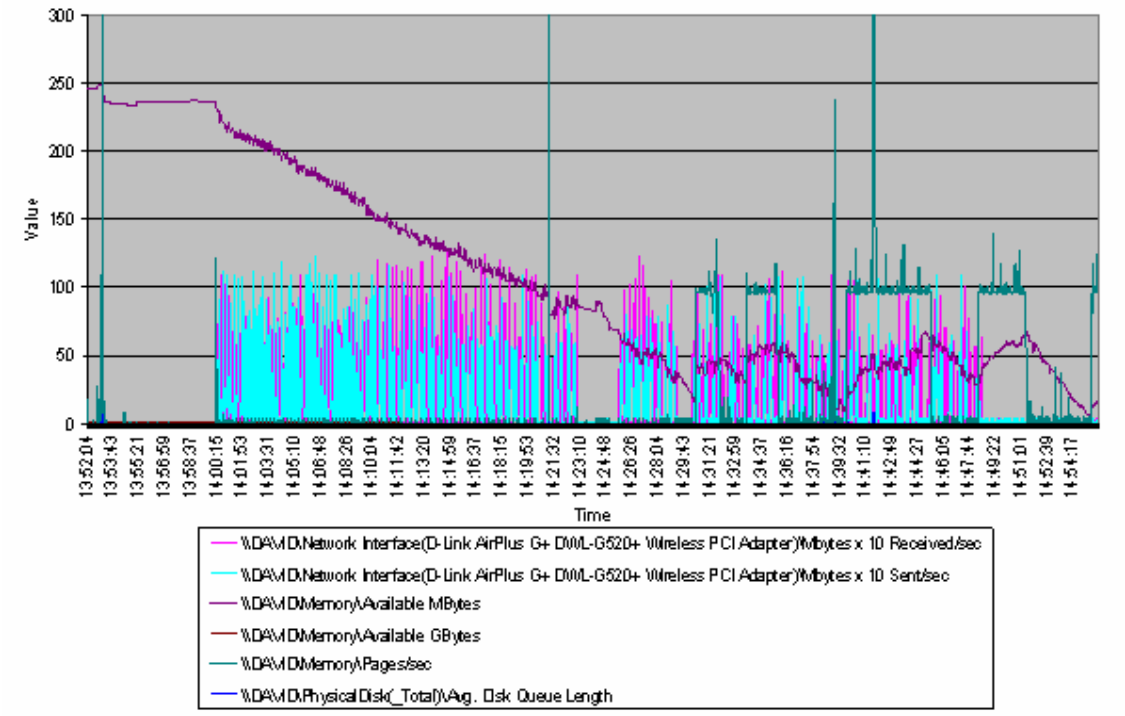

Figure 4 - Performance Testing Results

6 Conclusion

The overall objectives of the project were primarily focused on proving the validity of the dynamic data storage theory upon which the project was based by designing and implementing software that demonstrated the concept. This goal was successfully met, and proof of the effectiveness of the design lies in the system testing results. It has been shown that it is possible to apply the grid computing architecture to means other than common applications such as Compute and Data Grids; the grid computing architecture may be manipulated in order to facilitate dynamic data storage using the under utilised RAM and network resources of grid nodes.

The basic product suite that was developed has been targeted at desktop PC users who may have an interest in taking part in peer to peer computing activities and accept the challenge of experimenting with new ways to store their growing collections of data. As the application framework advanced it became evident that dynamic data storage has potential to be used for a number of purposes, including secure file distribution with the intent of hiding files on the grid, and peer file sharing to name but a few.

7 References

[1] Abbas, A., 2004, Grid Computing: A Practical Guide to Technology and Applications, Charles River Media, Massachusetts

[2] Akshay Luther, Rajkumar Buyya, Rajiv Ranjan, and Srikumar Venugopal, 2004, *Peer-to-Peer Grid Computing and a .NET-based Alchemi Framework*

[3] *Alchemi .NET Grid Computing Framework*, http://www.alchemi.net/, Accessed 26/07/2004.

[4] Developer Works, 2003a, http://www-106.ibm.com/developerworks/grid/library/gr-design.html

[5] Douglas Thain, Jim Basney, Se-Chang Son, and Miron Livny, 2001, *The Kangaroo Approach to Data Movement on the Grid*, University of Wisconsin-Madison

[6] Thomas, M., 2003, *Overview of Grid Computing Environments*, Global Grid Forum.