Non-intrusive and Incremental Evolution of Grid Programming Systems

by

Xin Zuo

A Thesis submitted to the
National University of Ireland, Dublin
for the degree of Ph. D.
in the
College of Engineering, Mathematical and Physical Sciences

March 2011

School of Computer Science and Informatics
Under the supervision of
Dr. Alexey Lastovetsky

ACKNOWLEDGEMENTS

With no doubt, the work on this thesis has been the most challenging and interesting project undertaken so far. Generally, I would like to thank all the people involved in this thesis, to have at the same time provided me the means to work and enough freedom to do a “real” PhD.

I am thankful to my supervisor Dr. Alexey Lastovetsky for his guidance, patience, and encouragement. Without him this thesis will never come to an existence.

I would like to thank the University College Dublin (UCD), Computer Science and Informatics School for providing me with the opportunity to learn, facilities to perform my research, and a motivating environment that carried me forward through my course work. I would like to express my special gratitude to the Computer Science Administrative Department for their resourcefulness and invaluable help throughout the course of my research.

I must also thank the Heterogeneous Computing Laboratory group members. My special gratitude goes to Ravi Reddy, Brett Becker, and Thomas Brady for their collaboration and invaluable discussions and for their understanding and encouragement.

Finally, I thank my parents and my wife Ying Liu to have supported me in some difficult moments.

Contents

List of Figures

List of Tables

Part I

Approach for Evolution of Grid Programming Systems

Chapter 1

Introduction

1.1 Software Features between Industry and Community

Both universities and businesses began to produce programs to do certain computing tasks when the first computers were sold out in the early 1960s. Since then the computer markets grow fast in both hardware and software areas. In the 21st century the IT industry now involves about thousands of companies and communities with millions of developers and researchers. In the area of software development, lots of successful products have been developed by both industries and communities. One of most famous software is Windows Operating System [Mic] developed by the company Microsoft. By contrast, community developers have produced Linux OS [Acc86], one of the most prominent examples of free and open source [Osi] software collaboration. Although the functions are quite similar between Windows and Linux, there are differences in many aspects. This is also considered as the general differences of software development between industries and communities. These differences are described from followings aspects:

Usage & Driven: The purpose of software development in industry is to achieve financial success and is mostly driven by investors. It provides services to companies and personals in many different areas such as financial, entertainment, scientific, etc. For example, Windows is widely accepted everywhere and has millions of software packages that run under it. While the software development running by communities normally don't enjoy nearly the same marketing or development budget that industry

have. In fact, the developers from communities and academic are commonly credited as founding the Open Source Software movement, which is the idea that software can be made better through the free sharing of its source code. In this philosophy, programmers often volunteer their time to develop software for free. The software developed by communities are normally used by personals or communities, but sometimes are used for commercial purpose or extended by industry developers. Besides these, there are also some projects that are driven by government or organization, and both community and industry take part in the development of software, or one side is mainly responsible for the development.

Ownership: Generally, the software developed by industries is much stricter in governing the usage or redistribution of software than the software developed by communities. Ownership of most Industry software remains with the software publisher and the end-user must pay fees and accept the software license to use the software at all. Also, software companies often make special agreements with large businesses and government entities that include support contracts and specially drafted warranties, including an extensive list of activities which are restricted such as: reverse engineering simultaneous use of the software by multiple users, and publication of benchmarks or performance tests. On the other side, software developed by communities are mostly with a free license, which means the ownership of a particular copy of the software does not remain with the software publisher. Instead, ownership of the copy is transferred to the end-user. As a result, the end-user is, by default, afforded all rights granted by copyright law to the copy owner. Additionally, a free software license typically grants to the end-user extra rights, which would otherwise be reserved by the software publisher. An example of Free Software license is the GNU General Public License (GPL) [GNU] which gives the end-user significant permission but not entirely free of obligations for the end-user. For instance, any modifications made and redistributed by the end-user must include the source code for these, and the end-user is not allowed to re-assert the removed copyright restrictions back over their derivative work. Another example of free software licenses are the BSD license [BSD], which essentially grant the end-user permission to do anything they wish with the source code in question, including the right to take the code and use it as part of closed-source software or software released under a proprietary software license.

Development: Although both Industry and Community are targeting to make software development predictable, controllable, and manageable, software isn't usually developed that way. But it can be improved to get very closed in that way. Compared to communities, industry companies have put huge resources into it such as staff recruitment, software testing, etc. The industries always try to get the best fit developers through agent or their own human resources apartment, while the communities are usually teamed up by different level of developers and researchers from different networks. The commercial products developed by industries are relatively more stable and secure than the software developed by community. There are many software testers who have certain qualifications and certificates involves in quality assurance of commercial products, while for the software developed by communities most testers are volunteers or end-users who is doing testing only for their own purpose, which is not covering the test of all aspects for software. However, it doesn't mean that the qualities of all community software are worse, for example GNOME [GNO] is considered as one of best graphical user interface that runs on top of a computer operating system. Commercial software products usually have very good and quick support compared to the software developed by communities for some obviously reason someone is paid to do so. Although there are lots of help available on the internet and there are many self-motivated forums that can help to use the software developed by community, there are few qualified support available. For example, you have to figure out on your own how to install and use applications without sabotaging your data and hardware. The most case would be that you may find an annoying bug in an application that you need assistance with, but you may not get it for months or without paying someone to fix it.

Management: On the industry side, in most IT company software development process may involve many departments, including marketing, engineering, research and development, and general management. Usually there is a certain methodology used to structure, plan, and control the process of software development. Each of the available methodologies is best suited to specific kinds of projects based on various technical, organizational, project and team considerations. On the community side, management of software development varies depending on the project as well. Some small projects just simply have a development team only, and developers within the

team may do a job of management at the same time. For some big projects which involve many community developers they have a good strategy to manage the software development. GNOME is as an example again they have a regular release cycle and a disciplined community structure. It gives freedom to developers free to develop any modules or component for GNOME, but for integration into next release of GNOME the new add-ins have to be voted by community members of GNOME. There are a few projects developed by community having as strong management as industry companies have. For example, MPI [GL][SOW95] - A Message Passing Interface Standard which establish a practical, portable, efficient, and flexible standard for message passing. The project was led by University of Tennessee, Knoxville, Tennessee with participation from over 40 organizations. The project was well managed by the group from University of Tennessee and was developed by researchers from different communities. In the area of high performance computing, it is widely used as a standard and extended into many different implementations by both industry and community.

1.2 Software Development of High Performance Computing

1.2.1 Context

High Performance Computing (HPC) is most commonly associated with computing used for scientific research, such as problems involving quantum physics, weather forecasting, molecular modelling, space technology, etc. Both Universities and Companies have made significant contribution to the development of hardware and software for High Performance Computing for more than 50 years. Today companies and researchers are using the hardware mostly designed and produced by industries, while the software running on top of the machines are implemented using the approaches mostly from the effort of community's research. Regarding hardware development for High performance computing, the power of super computing system doubles about every 1.2 years while the performance of single computer doubles approximately every two years according to Moore's Law [Com03]. Moreover, different architectures are designed for super computers to achieve performance and stability, such as a) massively parallel computers and clusters running proprietary software, (b) proprietary clusters running standard software, and (c) Grid clusters built from commodity hardware and software. Today, a list of the most powerful

high-performance computers can be found on the TOP500 [Top500] list measured by the High Performance Linpack (HPL) benchmark [DLP03] published by community. As a consequence, hardware became less and less costly, but software became more and more complicated and multifarious. Regarding software development for high performance computing, community have made main contribution for software development of high performance computing. The followings are techniques widely used in the past:

- High Performance Fortran (HPF) [KR][Lov93]
- Message Passing Interface (MPI)
- Open Multi-Processing (OpenMP) [OMP]
- Grid Computing [Gra05]

High Performance Fortran (HPF) is an extension of Fortran 90 with constructs that support parallel computing. Its original FORTRAN was the first high level programming language (software) invented by John Backus for IBM in 1954, and released commercially in 1957 which is still used today for programming scientific and mathematical applications. The primary design goal of HPF was 1) to support the data parallel programming style, 2) to achieve top performance on MIMD and SIMD machines with non-uniform memory access costs, 3) to support code tuning on various architectures. The first discussions on a High Performance Fortran standard were started at Super Computing ’91 and the detailed work began in March 1992 with a group of approximately 40 people. The HPF was officially published by the High Performance Fortran Forum (HPFF) as the HPF Specification v1.0 in May 1993. The development group contained members from industry, universities and United States government laboratories. There is official connection with recognized standards bodies for HPFF, but work on FORTRAN standardization is undertaken by the ANSI X3J3 committee which is effectively under directions from an international group WG5. The development work on HPF was quite active then, soon the FORTRAN 95 was published incorporated several HPF capabilities. In response, the HPFF again convened and published the HPF 2.0 Report. But while some vendors did incorporate HPF into their compilers in the 1990s, some aspects proved difficult to implement and of questionable use. Since then, most vendors and users have moved to

OpenMP-based parallel processing. However HPF continues to have influence and remains the top language in scientific and industrial programming, of course it has constantly been updated. Like other software developed by communities the software license of HPF is like BSD license, it's free to use and quite flexible to be extended.

Message Passing Interface (MPI) is a specification for an API that allows many computers to communicate with one another. It is a language-independent communications protocol used to program parallel computers. The goal of the Message Passing Interface simply stated is to develop a widely used standard for writing message-passing programs. The main advantages of establishing a message passing standard are portability and ease of use. Its main features generally includes: 1) Allow efficient communication, 2) Allow for implementations that can be used in a heterogeneous environment, 3) can be implemented on many vendor's platforms, 4) Semantics of the interface should be language independent. The design of MPI sought to make use of the most attractive features of a number of existing message-passing systems rather than selecting one of them and adopting it as the standard. Thus, MPI has been strongly influenced by work at the IBM T. J. Watson Research Center [BK92][BKR92], Intel's NX/2 [Pie88], Express [PCP92], nCUBE's Vertex [Ncu92], etc. A preliminary draft proposal known as MPI1 was put forward by Dongarra, Hempel, Hey, and Walker in November 1992. The draft MPI standard was presented at the Supercomputing 93 conference in November 1993. The development of MPI is organized by MPI forum, involving about 60 people from 40 organizations mainly from the United States and Europe. Most of the major vendors of concurrent computers were involved in MPI, along with researchers from universities, government laboratories, and industry. MPI forum was not officially set up, it is constituted together with meetings and emails during the development of MPI. The membership of MPI forum is open to all members of the high performance computing community. The MPI project is well managed by MPI forum all the time, at the beginning to achieve the goal the MPI working group met every 6 weeks for two days throughout the first 9 months until the releasing of the alpha version. Also, annual conference EuroMPI [EM10] has been held since 1994 related to the research and development of MPI standard. The implementation language for MPI is different in general from the language or languages it seeks to support at runtime. Most MPI implementations are done in a combination of C, C++ and assembly language. Both

industries and communities have implemented software based on MPI standard. The initial implementation was MPICH [MPICH], from Argonne National Laboratory and Mississippi State University. After that, most supercomputer companies of the early 1990s like IBM had built commercialized implementations just based on MPICH, instead of building their own version from the start to the end. Other implementations developed by community are such as LAM/MPI [SL03], OpenMPI [DM98]. So far, a couple of versions for MPI standard have been built. MPI-1 is the original version, where single group operations are most prevalent. MPI-1 emphasizes message passing and has a static runtime environment. MPI-2.1 (MPI-2, completed in 1996), which includes features such as parallel I/O, dynamic process management and remote memory operations. MPI-2 specifies over 500 functions and provides language bindings for ANSI C, ANSI Fortran (Fortran90), and ANSI C++ [Ian95]. The MPI-3 [MPI3] standard is currently under development and it is scheduled to be ratified by the end of 2010. The new standard will tackle a variety of issues, including 1) remote memory access for one processor to write to another processor's memory, 2) fault tolerance to respond to a problem without crashing the application, and 3) non-blocking collectives for simplifying and improving communication within an application. Today after many years' development MPI is widely accepted by both industry and community, and remains the dominant model used in high-performance computing.

OpenMP (Open Multi-Processing) is an application programming interface that supports multi-platform shared memory multiprocessing programming. It specifies a collection of compiler directives, library routines, and environment variables that can be used to specify shared-memory parallelism in C, C++ and FORTRAN programs. This functionality collectively defines the specification of the OpenMP Application Program Interface (OpenMP API). This specification provides a model for parallel programming that is portable across shared memory architectures from different vendors. The goal of OpenMP includes 1) Standardization: Provide a standard among a variety of shared memory architectures/platforms, 2) Lean and Mean: Establish a simple and limited set of directives for programming shared memory machines, 3) Ease of Use: Provide capability to incrementally parallelize a serial program, unlike message-passing libraries which typically require an all or nothing approach, 4) Portability: Supports Fortran (77, 90, and 95), C, and C++. The first OpenMP API

specifications were published by the OpenMP Architecture Review Board (ARB) for Fortran 1.0 in October 1997 [CDK01]. One year later, ARB released the C/C++ standard. In the year of 2000 version 2.0 of the FORTRAN specifications were released. In 2002, version 2.0 of the C/C++ specifications was released. Until 2005, a combined C/C++/Fortran specification was released in 2005. The latest version 3.0 was released in May, 2008, which is the current version of the API specifications including the concept of tasks and the task construct. OpenMP has been implemented in compilers by both industry and community. For example, the commercial implementations are like Sun Studio compilers, Intel Fortran and C/C++ versions Compilers, IBM XL C/C++ Compiler. The open-source compilers are like The Fortran, C and C++ compilers and GCC (GNU Compiler Collection) [GCC] compiler. GCC is one of most successful implementation based on OpenMP. It has been adopted as the standard compiler by most other modern Unix-like computer operating systems, including GNU/Linux, the BSD family and Mac OS X. Nowadays it is ported to a wide variety of processor architectures and is widely deployed as a tool in commercial, proprietary and closed source software development environments. There is often some confusion between OpenMP and MPI. From a programmer's standpoint, MPI is a library that contains message passing routines. OpenMP, on the other hand, is a set of compiler directives that tell an OpenMP enabled compiler what portions of the program can be run as threads. Thus the main difference can be considered as threads against messages. OpenMP is developed for use by the high-performance computing community and it works best in programming styles that have loop-heavy code working on shared arrays of data.

Grid Computing originated in the early 1990s as a metaphor for making computer power as easy to access as an electric power grid in Ian Foster's and Carl Kesselman's seminal work, "The Grid: Blueprint for a new computing infrastructure" [FK04]. In contrast to the traditional notion of a supercomputer, which has many processors connected by a local high-speed computer bus, Grid Computing is a special type of parallel computing that relies on complete computers connected to a network by a conventional network interface, such as Ethernet. The definition of Grids is environments that enable software applications to integrate instruments, displays, computational and information resources that are managed by diverse organizations in widespread locations. Grid computing appears to be a promising trend for three

reasons 1) make more cost-effective use of a given amount of computer resources, 2) solve problems that can't be approached without an enormous amount of computing power, 3) the resources of many computers can be cooperatively and perhaps synergistically managed as a collaboration toward a common objective. So far, there are many Grid projects worldwide, which are hierarchically categorized as core middleware, integrated Grid systems, user-level middleware, and applications driven efforts [Buy02]. (a) There are different kinds of Grid core middleware such as Globus Toolkit [FK97], gLite [Gli], and UNICORE [Uni], which provide uniform and secure environment for accessing remote computational and storage resources. (b) Integrated Grid systems are like GridSolve/NetSolve [ICL][CD97][ACD02][ICLG] and NINF [TNS03], which are programming and runtime system for accessing high-performance libraries and resources transparently. (c) User-level middleware includes Schedulers and Programming Environments, ex: Condor-G [FTF01], a grid job processing system; MPICH-G [KTF03], MPI implementation on Globus. (d) Applications and application-driven Grid efforts are these examples: European Data Grid [EDG] used for high energy physics, earth observation, biology; IPG implemented by NASA for aerospace [IPG]; Earth System Grid by LLNL, ANL &NCAR for climate modelling [ESG]; etc. At the moment wide-area high performance distributed computing has been a popular application of the Grid. Besides this, there are also a large number of other applications that can benefit from the Grid.

1.2.2 Motivation of Research and General Objectives

In this thesis, a further analysis respecting the software development of High Performance Grid Computing is given for following reasons:

- In past ten years, the grid computing phenomenon is a hot topic that has captured the interest of many technology organizations, from academic researchers to community developers. This technique radically changes the economics of High Performance computing. There are many scientific research and real-world applications have been benefit from the Grid.
- In the area of Grid Computing, communities have done a plenty research and development works which made main contribution in this area. Compared to

the other High performance computing techniques such as HPF, MPI, OpenMP, industries only show interests instead of putting resources and efforts into it. As a result, the projects development related to grid computing are totally driven by community and most grid computing software are developed by community.

- Although the size of Grid computing community is quite big, within community the groups and organisations are loosely connected. It causes that the software development of the Grid Computing projects commonly have a few issues, which are generally the same to the issues of software development on the community side in the aspects of development, management, etc.

Nowadays, High Performance Grid Programming Systems have reached a certain level of maturity. It allows scientific programmers to develop reliable Grid applications. The systems are quite easy to install and use. They also demonstrate high level of stability and reliability achieved over years of testing and maintenance. On the other hand, the constantly growing number of users and applications results in the need of further development of such systems in terms of functionality and quality. Traditionally, addition of a new feature to a Grid programming system is achieved by changing the code of the system and producing its new version. This new version of the system has to replace the previous one in order to enable Grid applications with new features. This original approach to the evolution of Grid programming systems has two serious disadvantages. Firstly, the change of the system's code may introduce bugs resulting in the situation when some applications, which have been developed, tested and successfully executed with the previous version of the system, will not run properly with the new one. Secondly, the new version of the system has to replace the old version systems on all computers of the Grid in order to support the development and execution of applications enabled with the new feature. Such simultaneous and total replacement can have very high organizational overhead and sometimes be simply unrealistic as different computers on the Grid are managed and administered by independent and, very often, loosely connected users. Hence, our research focuses on establishing a new approach to address these issues. The approach we propose and study in this thesis is to introduce a new feature by implementing a supplementary software component running on top of the existing grid programming system. In this

way, the new feature is added to the grid programming system non-intrusively and incrementally. Non-intrusiveness means that the original system does not change and the new feature is provided by a supplementary software component working on the top of the system. Increment means that the software component does not have to be installed on all computers to enable applications with the new feature. It can be done incrementally, and the new feature will be enabled in part.

The main objective of the presented research is to prove this concept by picking a typical Grid programming system and a particular feature, which this system is lacking, and adding this feature to the system in the described way. As this project is performed in the context of high performance computing, both the system and the feature should be relevant to high performance computing in Grid environments. To achieve our research target, particularly we worked on a case that present software component enabling NetSolve applications with direct communications between remote tasks. NetSolve is positioned as a one of most popular programming system for scientific researchers to develop reliable Grid applications on global networks. The grid feature adding into NetSolve is enabling direct communications between remote servers, which is also widely recognized as a major issue in Grid Programming Systems by community. In our research work, the software component NI-Connect has been designed and implemented in a non-intrusive and incremental way. Generic design principles are also proposed. It can be re-used for developing non-intrusive and incremental software component for other Grid programming system to enable new features. At last based on the implementation of enabling direction communications between remote servers in NetSolve, we present real-world applications with different communication structures and demonstrate the performance improvement achieved due to the use the software component for elimination of bridge communications on both local network and large-scale Grid environment. This thesis also reviews the use of approach in heterogeneous communication networks, which are typical for real-life Grid environments, than in artificially designed homogeneous ones.

1.3 Outline of the thesis

Chapter 2 describes the approach for Evolution of Grid Programming System in details. Chapter 3 studies on the selected grid programming system and the feature that is used to test the approach. Chapter 4 gives the implementation of software component NI-Connect for the targeted system and the selected feature. Chapter 5 proposes generic design principles of software component for general grid programming systems. Chapter 6 presents the experiment results of using NI-connect to conduct the real-world applications. Chapter 7 presents large-scale experiments on heterogeneous grid networks. Chapter 8 concludes the thesis and presents ideas for future work.

Chapter 2

Analysis and Approach Statement

2.1 Analysis of GridPRC Related Programming System

To demonstrate the approach for evolution of Grid Programming System we choose one of High Performance Grid Programming Systems named NeSolve. There are three reasons why we select NetSolve system in our research. First, it is considered to be one of best programming systems in the grid environment to perform high performance computing tasks. Second, NetSolve allows scientific programmers to develop reliable Grid applications and gain non-performance-related benefits, like ease of development and control of the application. Third, NetSolve has most influential in the appearance of GridRPC [SNM02] model, which represents to be a basic and standard mechanism for grid computing. This is also the most important reason why we select NetSolve as our targeted Grid Programming System. In our research, we choose to investigate the Non-intrusive and Incremental approach for evolution of NetSolve system with new grid feature. As a result, the approach can be applied to any Grid Programming Systems related to GridRPC model, such as GridSolve, Ninf-G and DIET [INR] [CD06] because these Grid Programming Systems share as same as design principles of NetSolve and actually just have slightly different APIs. In this chapter we start to introduce our research work by describing GridRPC and investigating the proposed approach for evolution of GridRPC software in different aspects.

2.1.1 GridRPC

In a general application, a called procedure runs in the same computing resources and results are returned to the calling procedure. When the client side calls a procedure which runs on the different servers and the results are sent back to the client, we consider it as a distributed environment with a client and server running on different computing resources. This is called a remote procedure call (RPC) [BN84] and forms the basis for RPC programming. The GridRPC API is an extension of RPC to standardize and implement a portable and simple remote procedure call (RPC) mechanism in the grid environment. It is a standard promoted by the Open Grid Forum which extends the traditional RPC for the Grid environment. A GridRPC system processes each GridRPC task call by first performing dynamic resource and task discovery, then mapping the task to a server and then executing the task on the mapped server. Since each GridRPC task call consists of these operations (discovery, mapping and execution) and each GridRPC task is processed individually, the GridRPC model imposes the restriction that these three operations are atomic and cannot be separated. As a result, each task has to be mapped separately and independently of other tasks of the application. . Recent results of GridRPC system performance analysis [ML09] indicate that GridRPC model achieves better performance for those data parallel applications are not representative. Figure 2.1 shows the structure of GridRPC model.

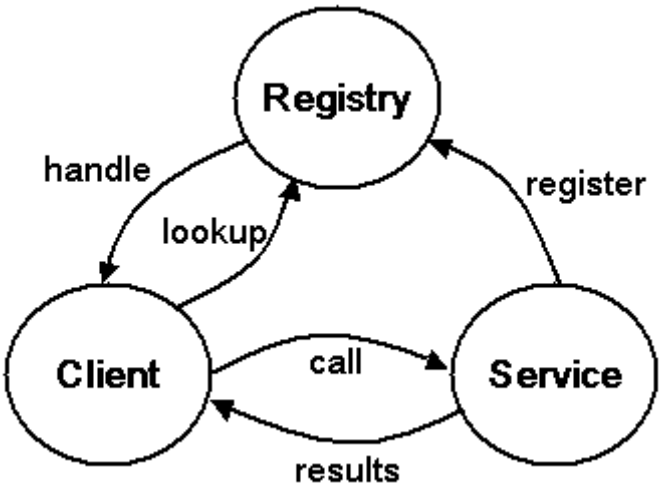


Figure 2.1 GridRPC model

The research work related to GridRPC model such as GridSolve and NINF-G shows that client access to existing grid computing systems can be unified via a common API, and computational tasks are proven to be solved. The GridRPC programming model is easier than the RPC one since the programmers do not need to specify the server to execute the task and the stub for each remote task. Furthermore, GridRPC extends RPC since it adds asynchronous remote task calls. Currently, various Grid middleware systems implement the GridRPC model, such as GridSolve, Ninf-G and DIET.

2.1.2 Evolution of GridRPC Related Software

Nowadays, high performance Grid programming systems have reached a certain level of maturity. Scientific researchers and programmers can develop reliable Grid applications by using systems such as NetSolve/GridSolve and Ninf-G, DIET. The software developed by using GridRPC model is quite easy to install and use. They also demonstrate high level of stability and reliability achieved over years of testing and maintenance. On the other hand, the constantly growing number of users and applications results in the need of further development of such systems in terms of functionality and quality.

Traditionally, addition of a new feature to a GridRPC system is achieved by changing the code of the system and producing its new version. For example, NetSolve/GirdSolve provide new features and enhancements by releasing new version of software [News]. The new version of the system has to replace the previous one in order to enable Grid applications with the new feature. This original approach to the evolution of GridRPC systems are considered to have three serious disadvantages.

- The change of the system's code may introduce bugs resulting in the situation when some applications, which have been developed, tested and successfully executed with the previous version of the system, will not run properly with the new one.
- The new version of the system has to replace the old version on all computers of the Grid in order to support the development and execution of applications

enabled with the new feature. Such simultaneous and total replacement can have very high organizational overhead and sometimes be simply unrealistic as different computers on the Grid are managed and administered by independent and, very often, loosely connected users.

- Since the research and development of GridRPC System are mainly carried out by community, the support and update of most GridRPC software are not as professional as commercial software products usually are. It may take months for users to get a new version with bug fixed and new feature added, or even longer time. It heavily depends on the management of each GridRPC projects and users can do few things unless they find another approach to satisfy their requirements, ex: a light-weighted external component.

Thus, our research focuses on developing a new approach, that in a way without these issues, to enable new features in an existing Grid programming system.

2.2 Non-intrusive and Incremental Approach

In chapter 2.1, we have analyzed the evolution of GridRPC related software. The result shows that the traditional approach to enabling the existing RPC-based Grid Programming System with new features is considered to have two disadvantages:

- **Intrusive**, that is, the code of the system has to be changed in order to add the feature; Re-configuration and re-compilation are needed for original system.
- **Non-incremental**, that is, to make the system functional with the new feature, the modified system has to be installed on all the computers that are supposed to participate in the execution of applications.

Correspondingly, we formulated a new approach to enable an existing Grid system with a new feature. Its main difference from traditional ones is that it is **Non-intrusive** and **Incremental**.

2.2.1 Definition of Features

In this section, we formally define the key features of our approach.

Definition of ***Non-intrusiveness*:**

- The original RPC-based grid programming system does not change, including client-side APIs and library, server-side programs and all existing server procedures.
- The new features are provided by a supplementary software component working on the top of the system. The software component on both client-side and server-side are running actually as external processes to original grid programming system.
- Original system does not need re-compilation or re-installation. This applies to all grid nodes. If the re-compilation and re-installation of original grid programming systems are performed, it does not impact with supplementary software component.
- Supplementary software component does not affect original system and make no changes to grid environment. It can be enabled or disabled at any time. Any changes made to the supplementary software components have nothing to do with original grid programming system.

Definition of ***Increment*:**

- The supplementary software component does not have to be installed on all computers to enable applications with the new features. It means the new features can be enabled partly in the grid environment.
- To enable applications with the new features. It can be done incrementally, step by step. All original grid programming systems are working as usual for all grid applications at the same time.
- The new features can be enabled in part, with the completeness dependent on how many nodes participating in the execution of the application have been upgraded with the supplementary software component.

Next, we will compare the difference between proposed approach and original approach for evolution of grid programming systems.

2.2.2 Approach Comparison

Compared to original approach, the non-intrusive and incremental approach is designed to achieve three general benefits. First, to establish a more robust way for evolution of existing grid programming system, this reduces the risk of crashing other grid resources. Second, minimize the development effort involved so that programmers can easily add new features to existing grid programming system. Third, new features can be easily extended to all grid nodes depends on how many nodes participating in the execution of the application which have enabled software component. Following paragraphs will describe the differences between original approach and non-intrusive and incremental approach in details.

To add new functionalities or features, Grid Programming Systems traditionally release a new version of the software to achieve the changes. The whole process includes changing of software code, linked library, documentation, and other support materials, etc. The testing and quality assurances of new version also need to be carried out in the grid environment. The bugs produced by updating current grid programming system can be from many different aspects. For example, the existing grid application is not compatible with new API/IDL [IDL] or some applications may need to be tweaked to take full advantage of the new model. Even the client-server interaction functions may not be working on gird nodes which are installed different versions of Grid programming system. There are also various reasons for not fixing bugs such as the developers often don't have time or it is not economical to fix all non-severe bugs. Or sometimes the changes to the code required to fix the bug could be large, expensive, or delay finishing the project. Like any other part of engineering management, bug management must be conducted carefully and intelligently. It could have unintended consequences, especially in grid environment. While in our research, the non-intrusive and incremental approach we proposed is able to eliminate all these disadvantages. It is done as simple as that the original system is not changed at all if we add new features non-intrusively. Programmers do not need to worry about bugs produced by original system when they develop supplementary software component.

The testing works are also relatively less. As a fact in this case evolution of software can be considered as a separate project which is related to the original system, including design, develop and testing. The difference is that the developers have to be familiar with original grid programming systems and have knowledge with certain level of technical details.

By analyzing the original approach to enabling new features to existing grid programming system, we also discovered that there are large amount of programming effort. If one of components of Grid Programming System switches to new technology, there are many related works such as re-deployment, re-configuration on all grid nodes. And testing/QA for new version of Grid Programming System normally cost a huge amount of resources. For external linked API/Library, there are certain extension works involved. Because technologies evolve quickly, the development cost will be kept increasing based on the current model of upgrading Grid programming system. Beside this, there are also some other issues to consider when determine technical feasibility such as performance, ease of learning, Scalability, third-party support for related products and relationships to Grid Middleware [LHP04]. Since our research is focusing on the grid programming systems in the area of high performance computing, researchers would like to add new features to existing system as easy as they can and what they really care about is the amount of development work involved. If both intrusive and non-intrusive approach works, for their own research purpose they normally choose the easier one. As a fact, the non-intrusive and incremental approach can reduce the development effort involved compared with original approach. The design is relatively simple compared with changing original system. Because it is realistic that external developers develop a supplementary software component rather than starting a project together with the developers who developed original grid programming systems. In this case the management of the project is easier. And the testing will only be for supplementary software component, not including every part of original systems. And to apply the changes to grid environments by using the non-intrusive and incremental approach, the amount of works are quite flexible as it is not necessary to change every grid systems for adding new feature. While the original approach have to do so, which involves certain amount of works.

Since one of key properties of grid environment is heterogeneous, the original method to add new features to existing Grid Programming System appears to raise issues in two aspects. Firstly, management of grid resources are not compatible between different versions of grid applications, which means all old version system on the grid node must be replaced with new version to make the whole grid functional. Secondly, the communications between grid nodes which installed different versions of grid programming systems are disabled. This will affect the performance of grids as for computational task not all grid resources are able to be part of computations. Different versions of grid programming systems may have different APIs and functions, this does not allow the evolution of grid programming system to be done one by one, or partly deployed. But with using the non-intrusive and incremental approach, all grid programming systems are able to perform computation for the tasks and the features can be enabled partly. The difference is that for those grid nodes who have not added new feature, they can still work perfectly just without new feature enabled because every grid programming system is not changed at all. Actually, non-intrusive and incremental software component can be installed on any accessible grid nodes by system administrators. In this case, non-intrusive and incremental approach suits grid environments well compared with original approach.

As we have described, the non-intrusive and incremental approach for evolution of grid programming systems we proposed has many advantages compared with original approach. It is more reasonable to provide this new approach for evolution of Grid Programming System.

2.3 Summary of Research

Based on proposed approach concept, our research is to prove the feasibility of Non-intrusive and Incremental approach, and to demonstrate the performance by enabling new features on existing Grid Programming System. As a result, our research including following works:

- Investigate selected GridRPC related grid programming system and particular features which is added to the grid programming system. In particular, we analysis one of widely used Grid Programming System which

is NetSolve, and a specific feature that is to enable direct communications between remote tasks.

- Design supplementary software component for selected grid programming system with added grid feature.
- Based on the design, implement software component for NetSolve to enable direct communications and present the details of the implementation.
- Propose generic principles and APIs for implementation of non-intrusive and incremental software component, which is used to add new grid features to any existing Grid Programming Systems.
- Present experiments for real-world applications with different communication structures.
- Present experiments on large-scale grid environments with heterogeneous networks.

The next two parts of this thesis will present the details of above summaries.

Part II

Case Study and Generic Implementation of Non-intrusive and Incremental Approach

Chapter 3

Enabling Direct Communication in NetSolve

3.1 Introduction

Our research work is to demonstrate the feasibility of non-intrusive and incremental approach for evolution of Grid Programming System. To prove the concept, we pick a particular Grid Programming system which is NetSolve, and a particular feature adding to target system is enabling direct communication between remote tasks. As this project is performed in the context of high performance computing, both the system and the feature we select are relevant to high performance computing in Grid environments. In this chapter, the design of software component for NetSolve is described so that direct communications can be enabled between servers in a non-intrusive and incremental way [LZZ06].

3.1.1 NetSolve

The NetSolve project, is developed at the University of Tennessee and Oak Ridge National Laboratory, allows users to access computational resources, such as hardware and software, distributed across the network. There are two reasons why we select NetSolve as targeted Grid Programming System. Firstly, NetSolve is positioned as a Grid Programming system related to GridRPC model for grid environment.

Secondly, NetSolve is one of most popular systems which are used in the area of High Performance Grid Computing. Besides, at the end of year 2005 GridSolve was released by the Innovative Computing Laboratory (ICL) of the University of Tennessee who developed NetSolve as well. Because GridSolve and NetSolve are quite similar and in fact GridSolve is highly evolved from NetSolve, there is no difference to prove the concept of our non-intrusive and incremental approach by using any of these two Grid Programming Systems. While our research projects started in 2004, the design and implementation of non-intrusive and incremental software component for NetSolve have already been developed. In this thesis we still select NetSolve as targeted research Grid Programming System for our research work.

Generally, NetSolve deals with the situation when some components of the application cannot be provided by the user and are only available on remote computers. To program a NetSolve application, the user writes a NetSolve client program, which is any program (say, in C or Fortran [For]) with calls to the NetSolve client interface. Each call specifies the name of the remote task to be performed, pointers to the data on the user's computer required by the task, and pointers to locations on the user's computer where the results will be stored. When the program runs, a NetSolve call will result in a task to be executed on a remote computer. The NetSolve programming system is responsible for selection of the remote computer to perform the task, transferring input data from the user's computer to the remote computer, and delivering output data from the remote computer to the user's one.

NetSolve is designed to solve computational science problems in a distributed environment. The Netsolve system integrates network resources and provides a desktop application interface. The intent of Netsolve is to hide parallel processing complexity from user applications and deliver parallel processing power to desktop users. A Netsolve server can use any scientific package to provide its computational software. All component communications use TCP/IP. Netsolve provides resource discovery, fault tolerance, and load balancing. The Netsolve system follows the service Grid model with hierarchical cell-based machine organization. The NetSolve-agents act as an information repository and maintain the record of resources available in the network. As a new node comes up, information such as its location

and its services are sent to the NetSolve agent. The Figure 3.1 shows the overview of NetSolve System.

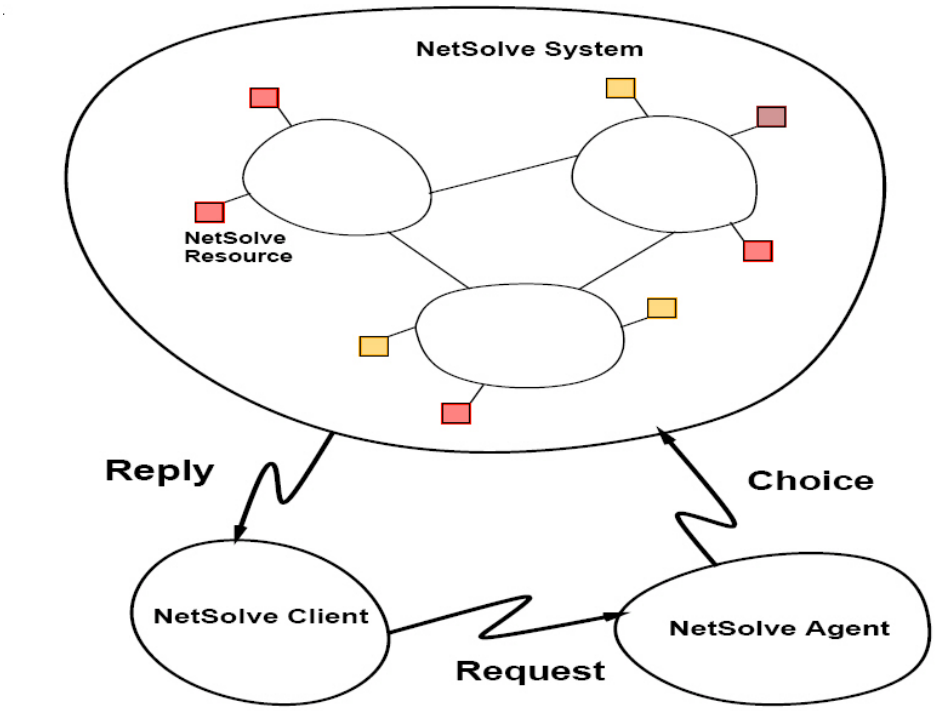


Figure 3.1 Overview of NetSolve System

3.1.2 Enable direct communications between remote servers

In this thesis, we have selected NetSolve as our targeted Grid Programming System to investigate non-intrusive and incremental approach. For the grid feature adding into NetSolve, we choose enabling direct communications between remote servers. This feature is widely recognized as a major issue in GridRPC systems by community. As a result, our case study will demonstrate both feasibility of our approach and performance of selected grid feature added into Grid Programming System. Compared to direct communication, the definition of unnecessary bridge communications is that output data of remote tasks are typically sent back to the client upon completion of each remote task even if the data are only needed as input for some other remote tasks. In this thesis, our approach is using a lightweight supplementary software component that enables direct communication between remote tasks in NetSolve in a non-intrusive and incremental way, without recompilation or reinstallation of the

original NetSolve programming system. The Figure 3.2 shows how to enable direct communication in NetSolve.

In the figure, we have two tasks, A and B, to be performed on remote nodes. The output of task A is the input of task B. In NetSolve, the output of procedure A has to be sent back to the client machine and stored there, and then transferred together with input B to another server. This causes unnecessary communications. To enable direct communication between procedure A and procedure B, the data are just transferred between the remote servers. The implementation of the NI-Connect will introduce handler to parse the parameters which will be presented in next Chapter. So upon invocation of NI-Connect, The use of handler tells procedure B to get the input from the server where Output A is located directly, instead of getting data from the client's machine. Thus in this way, the direct communication is enabled between remotes servers in NetSolve.

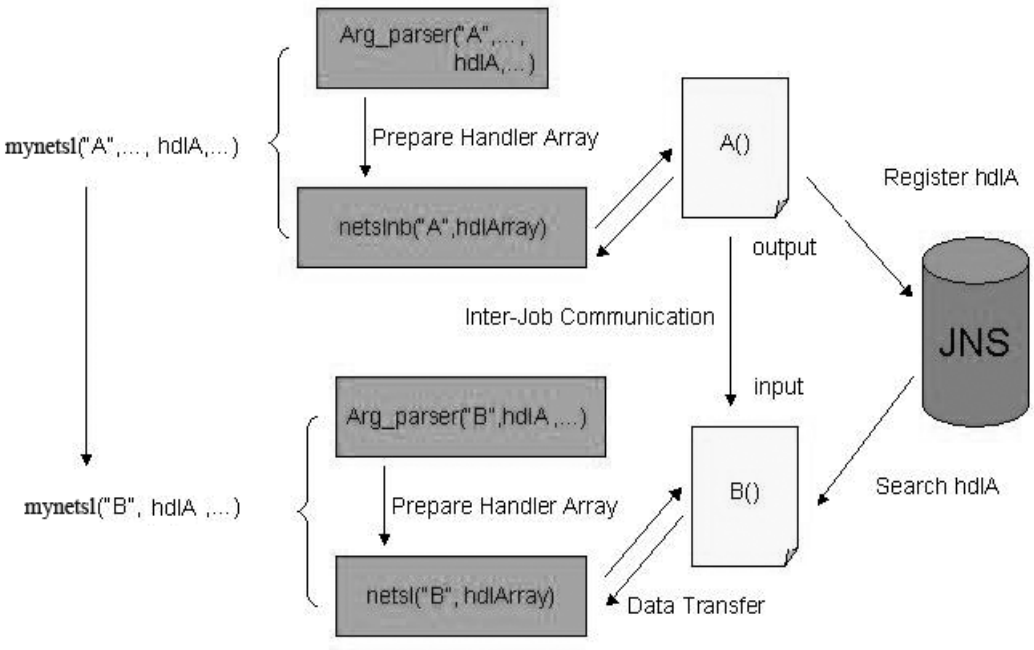


Figure 3.2 Enabling Direct Communication between remote servers

3.2 Related Works

To enable direct communications between remote servers, a few projects have implemented the function to solve the problem. We will describe the method they use and compare to our non-intrusive and incremental method one by one.

NINF is a remote computing infrastructure built as a component of the Asia Pacific Grid [Apg]. The latest release of NINF is an implementation of Grid RPC using Globus. These two systems share the similar motivation, design and protocol, as well as the similar problems, one of which is Bridge Communication. Both systems allow the user to invoke a procedure on the remote server. Ninf employs the Grid RPC specification. NetSolve uses its own API and also provides the port for other systems (Grid RPC, Matlab, Octave [Oct], Mathematica, etc.). Being constructed on the similar principles, NINF and NetSolve have a lot of similarities in their APIs. To alleviate the problem of bridge communication, NetSolve introduces the mechanism called Request Sequencing. The mechanism imposes a number of restrictions on the sequence of remotely called tasks, the most restrictive of which is that all the tasks have to be performed on the same computing node [AAB02].

Another effort to enable direct communications between remote servers is the Logistical Computing and Internetworking (LoCI) [BAB02]. LoCI provides facility to schedule the data storage at a place ‘close’ to the receiver. Although it is sufficient, the goal of building a complete network storage system makes LoCI over-heavy for our purpose.

One similar work to improve communication features for Grid environments is an extension of ProActive Groups [BB05], which provides a mechanism to optimize Grid environments. Based on IP multicast, changes are in internal behaviours and thus earn high performance communication for grid computing systems.

The REDGRID project [DJ04] is the closest to our approach sharing the similar idea behind its design. The main difference is that REDGRID is built into NetSolve. It requires re-compilation and re-installation of the modified NetSolve on all involved computing nodes to enable direct communication. Also the REDGRID’s design is not

extendable and relies on the NetSolve architecture, and it looks like that a certain amount of work is needed to port REDGRID to other GridRPC-based systems.

SmartGridSolve [BKL06][BG08][BDG09] is one of recent works of mapping tasks in Grid Programming System for high performance computing, which is an extension of GridSolve. In GridSolve each task in an application is mapped individually and independently of other tasks of the application. This model supports the minimization of the execution time of each individual task of the application rather than the minimization of the execution time of the whole application. SmartGridSolve allows tasks to be mapped collectively and therefore supports the minimization of the execution time of a group of tasks collectively. There are a number of advantages of mapping tasks collectively. When mapping tasks individually, the communication load and computation of a single task are only considered. However when tasks are mapped collectively the communication load and computation load of multiple tasks can be considered together. In addition the relationships between tasks can be considered such as the order of tasks and data dependencies between tasks. The SmartGridSolve model is designed to be both incremental and interoperable with the underlying GridRPC middleware. It is incremental to the GridRPC model which means a SmartGridSolve client application can be executed on a standard GridRPC network where all servers can only communicate with the client. Furthermore, a SmartGridSolve application can be executed on a network consisting of both GridRPC enabled servers and SmartGridSolve enabled servers (Smart Servers) that can communicate with each other. It is interoperable with GridRPC model which means that if the SmartGridSolve extension is installed, the application programmer has the option to implement their application using the GridRPC model or the SmartGridSolve model. As a result of these interoperability and incremental features SmartGridSolve is both upward and downward compatible with the GridRPC model.

3.3 Design of software component

Our research is to formulate a new approach to enable an existing Grid system with a new feature. The key features of our approach for evolution of Grid Programming System are Non-intrusiveness and Increment, the supplementary software component

to enable direct communications between remote servers in NetSolve needs to be designed and functioning in a non-intrusive and incremental way. Hence in our research, the design of software component is following two major design principles which are non-intrusiveness and increment. In this section, we will describe the design of non-intrusive and incremental software component for NetSolve to enable direct communications between remote servers in details.

3.3.1 Structure of Software Component

The proposed software component consists of three parts: *Client API & Argument Parser, Server Connector and Job Name Service (JNS)*. Client API is for client-side programmers to use, it provides a uniform interface for the client to make remote procedure calls. Server Connector is on the server side, which is a proxy program responsible for interacting with clients and other Server Connectors to enable direct communications. Job Name Service (JNS) is responsible for registration of a procedure upon its invocation during RPC call. Figure 3.3 depicts the architecture of the non-intrusive and incremental software component designed for NetSolve to enable direct communications between remote servers.

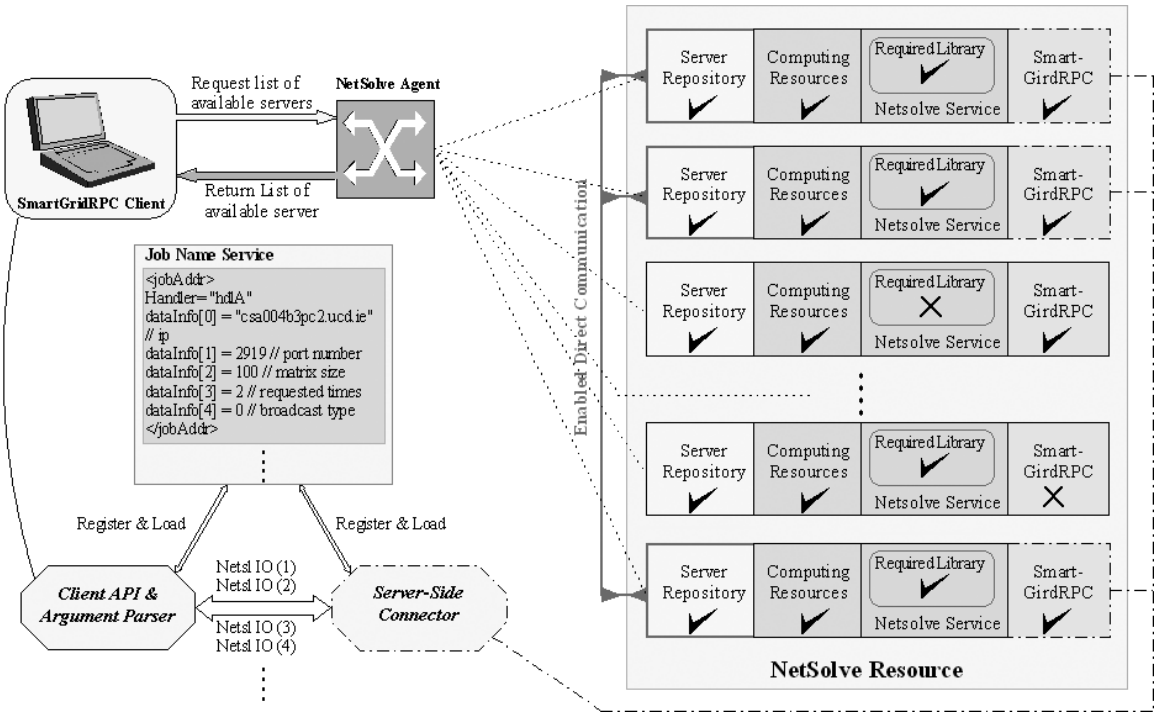


Figure 3.3 Architecture of the supplementary software component

Following steps shows how a grid application is performed in a grid environment where non-intrusive and incremental software component are installed on several grid nodes.

Step 1: Scientific researchers use Client API to program the computing task and specify the data resources which are going to be transferred to other grid resources. It's same to the way of programming in the original grid programming system. But programmers can specify the data dependencies for the computational tasks.

Step 2: Client API passes the description of the task to the Argument Parser, which communicate with grid resources to collect the resource information to run the task. The information includes the deployment of the grid applications in the grid which perform the task, the list of servers which have new features enabled, the performance of all available servers. Then Job Name Server generates the job list for each allocated server.

Step 3: Before the task is started to execute, Argument Parser register the information collected from grid resource to the Job Name Server. Then client API uses the information on the Job Name Server to invoke the Server Connector of NI-Connect to function.

Step 4: All Server Connectors invoked start to connect to the Job Name Server to retrieve the jobs list of their own. Based on the jobs list retrieved, Server Connectors communicate with client or other servers to require the data source to perform the scheduled task on each server. Direct communications can be enabled between remote servers through server connectors.

Step 5: With completion of data transaction, grid computing servers perform the execution of the task submitted by Server Connector and return the result to the Server Connector.

Step 6: All Server Connectors repeat the process until all of their jobs on the Job Name Server are completed. And the final result will be return to the client.

As we can see in the above steps, a remote task can be executed on the remote servers by using non-intrusive and incremental software component to enable direct communications. Next, based on the design of software component, we will present more details about functionalities of software component enabling direct communications in NetSolve on both client-side and server-side.

3.3.2 Client-side and Server-side Programs

On the client side,

- Client API provides a uniform interface for the client to make remote procedure calls. Despite the modification on the remote side, the wrapper API allows the calls to be made in the same manner. The only difference is in the arguments. We parse the list of argument to construct the handler array. For each argument, the relevant communication information is generated. For each input argument, which is a variable storing real data, the local IP address and the port number are used as such communication info. If this input argument is a handler, then a request is sent to the JNS to get the IP address and the port number of the remote resource and this information is used as communication info for this handler.
- For each output argument, which is a variable storing real data, the client wrapper function will set up a socket to download output data from computational servers. If this output argument is a handler, the re-turned result information from computational servers is sent to JNS and registered. In the future, other computational tasks can require the data source information from JNS and use the obtained information to get real data. A handler contains the data source/target's IP address and the port number, which will be used to send/receive data. In this sense, upon making a call to NetSolve, this is actually only a handler array which is transferred to the remote server. For example, in the wrapper function for mynetsl(), if the client cannot find any server, which has both Server Connector and the requested remote procedure, it will still run properly by using calls to the original netsl() functions.

On the server side,

- A proxy program called Server Connector is responsible for interacting with clients and other Server Connectors to enable direct communications. The Server Connector has two main functions. The first one is to pass handler information between clients and servers. This allows servers to know how to get the data by setting up direct communications. The second function is the extraction of the handlers' information and using it to download needed data through direct communication. After all needed data have been acquired, the Server Connector calls the procedure to re-submit to the local host to perform computations that the user exactly requested for. There is no difference in the way the client and computational servers download the result of the computations. The Server Connector firstly returns the result's communication information to the client. Then it sets up a socket waiting for the client or the server to connect in to download the result of computations.
- All the other I/O data transfer is managed by the Server Connector. In particular, the data transfer between the client and the server is performed with help of the client-side JNS. The algorithm of selection of the fastest server among all available servers is the same as the one implemented in the NetSolve agent program.

3.3.3 Analysis of Designed Software Component

To investigate how the designed software component to enable direct communications in NetSolve in a non-intrusive and incremental way, an analysis of principles based on the design is presented in following two parts.

Non-intrusiveness analysis:

- Original NetSolve programming system is not changed at all. Developers needn't to make any changes to the original grid programming system when they deploy supplementary software component on both client-side and server-side. It does *NOT* need re-compilation or re-installation of the original NetSolve servers. The server connector on the server-side is running as a new process individually which can be easily stopped by server administrator.
- All server-side procedures which are already deployed on the NetSolve server do not need to be changed as well. This allows other procedure

developers to develop their own applications as usual. After the supplementary software component is enabled on the server-side, any new procedure can be added to NetSolve in the way as same as before.

- Client API & Argument Parser, Server Connector and Job Name Service are all built on top of original NetSolve systems. If any of these three parts stop working, what happened is that the new features of supplementary software component are not available anymore. But it will not affect the original NetSolve systems in all aspects. Rather than vice versa, if NetSolve process is stopped, the supplementary software component no longer function anymore.
- Client programmer can either use designed Client API & Argument Parser or original NetSolve API to write his/her own client program. But with designed Client API & Argument Parser, client can specify the data dependency of all input in order to enable direct communications between remote servers. If client programmers use original NetSolve API to design tasks, there is no issue. In this case, supplementary software component is not invoked at all and as a result the new feature is not enabled. For compilation of programs, there is no difference between two choices.
- The relationship between Server Connector and Original NetSolve system is that the Server Connector re-submits the task to the local NetSolve server after all the necessary data are successfully received. NetSolve system regards Server Connector as a NetSolve procedure, just like all other procedures deployed on the server side. Regarding the interaction between Server Connector and NetSolve system, Server Connector can be considered as a client to the original NetSolve server.
- Server Connectors receive input data from and send output data to other servers to reduce un-necessary communications. Original NetSolve systems are not involved in the process at all. Server Connector is designed as a part of software component to enable new features, without Server Connector the original NetSolve systems are not able to enable direct communications between remote servers.
- Job Name Service is set up on the client side. Compared with NetSolve agent, the design and functionalities are quite similar in Job Name Server. It

contains all information about every handler. Handler registrations are done during the execution of the wrapper functions implementing the Client programs on the client side. There is no communication between JNS and Original NetSolve systems. Job Name Server can only be accessed by internal functions within the supplementary software component.

Increment analysis:

- The supplementary software component does not have to be installed on all servers. As what shows in Figure 3.3, not all NetSolve servers have installed the Server Connector. All Server Connectors can communicate with each other to enable direct communications between NetSolve Servers where supplementary software component has installed and enabled. Server Connector is the only part of supplementary software components on the server-side, both installation and un-installation of Server Connector can be easily done by server administrators.
- After installation of software component on the server-side, server administrators can enable Server Connectors incrementally, step by step. Server Connectors can be easily disabled after enabled if users want to cancel the new features which added to NetSolve system. On the client side, programmers only need to install and use the Client API & Argument Parser to develop computational task. There is a chance that if there is no Server Connectors enabled but client programmers still use designed Client API and Argument Parser to program remote tasks, in this case the remote tasks can still be executed in the way which is same to the process of performing remote tasks in original NetSolve system.
- To perform a remote task on the NetSolve Servers, required library has to be installed on the NetSolve Servers. This is the same in the original NetSolve system. But direct communications can only be enabled between servers which have Server Connectors installed. It doesn't mean that the servers whose have not Server Connectors installed won't be able to perform the remote task. Actually they are still able to execute remote tasks if they have required library, just not be able to enable direct communications.

- Direct communications can be enabled not only between remote servers, but also on the same server. If the input data is already on the computational NetSolve server, Server Connector can simply pass the data to the NetSolve procedure to perform the computation instead of sending data back to the client and get the same data back again. In this case sometimes even there is only one NetSolve server with supplementary software component installed and enabled, the direct communications can still happen.

The analysis of Non-intrusiveness and Increment shows the design of software component theoretically is able to enable direct communications between NetSolve servers. Next chapter we will present the implementation of software component enabling direct communication in NetSolve in details.

Chapter 4

NI-Connect: Implementation of Software Component

4.1 Overview

In last chapter, we select NetSolve as targeted Grid Programming System and enabling direct communications as new feature adding to NetSolve. We also presented the design and principles of supplementary software component which is used to add new features to existing Grid Programming system. As a result, we have implemented the software component NI-Connect for NetSolve to enable direct communications between remote servers. In this chapter, we will present the implementation of NI-Connect in details together with a case study. Installation and deployment of NI-Connect on both client and server side are also discussed.

4.2 NI-Connect Modules

The software component NI-Connect consists of three modules: Client API & Argument Parser, Server Connector and Job Name Service (JNS). In Figure 4.1 it shows the structure and communication model of NI-Connect:

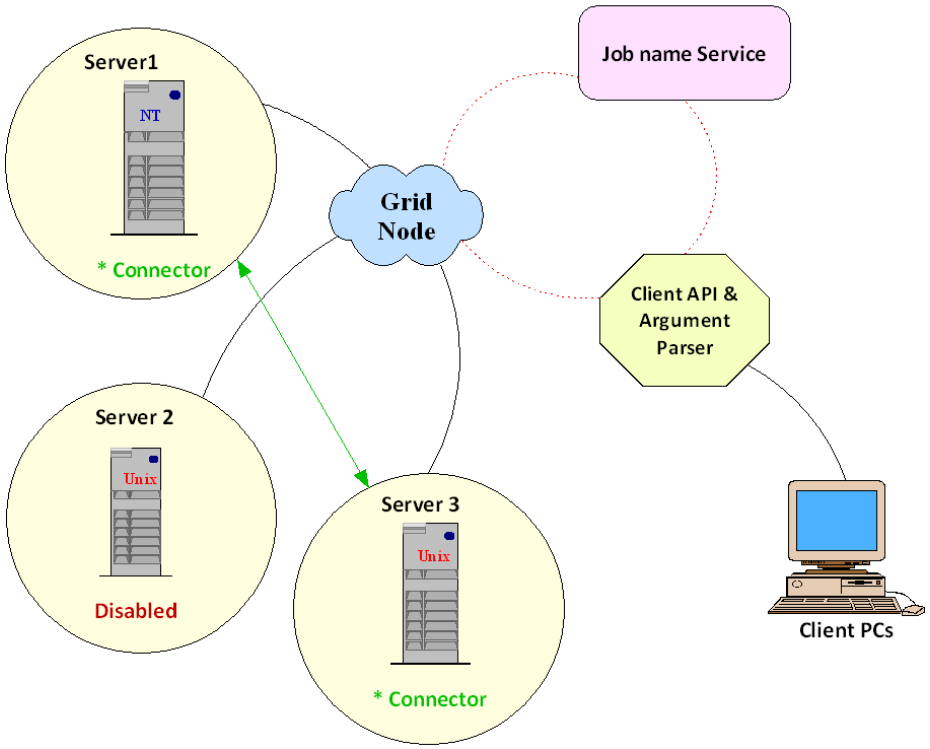


Figure 4.1 Structure of software component NI-Connect

Following sections will describe the different modules of NI-Connect in details.

4.2.1 Client API & Argument Parser

Client API provides a uniform interface for the client to make remote procedure calls. Despite the modification on the remote side, the wrapper API allows calls to be made in the same manner. The only difference is in the arguments. Same as the technique employed in NetSolve, we parse the calling parameter to construct the handler array and perform the data transfer. By checking each Arguments with a loop, communication information for each input and output data is generated. When input Arguments are variables which store real data, local ip and port number are given as data's communication info. Else if they are handlers, then requests are sent to JNS to get resource's ip and port number used for data's communication info. When output Arguments are variables which store real data, then client will set up sockets to download result data from computational servers. Else if they are handlers, the returned result information from computational servers are sent to JNS and registered there. So that in the future other computational tasks can request data source

information from JNS then use the requested info from JNS to acquire real data. A handler contains the data source/target's address and port number, which will be used to send/receive data. In this sense, upon making a call to NetSolve, this is actually only a handler array which is transferred to the remote server. All the other I/O data transfer is managed by the procedure itself. The API *mynetsl()* is summarized as follows:

```
Int mynetsl(ProcName,ArgList) {

// get list of Netsolve servers
server_list = my_NS_config();
for (i=0;i<number_of_servers;i++) {
   server_info[i]=get_info(servers_list);
}

// get list of problems for each server
for (i=0;i<number_of_servers;i++) {
     prob_list[i]=my_NS_problmes(server_info[i]);
}

// select servers which has installed
// our component and the problem "ProcName"
// is in the list of that server's problems list
servers_available= myselect(server_info, prob_list);

// if there is at least one server satisfy the two
// conditions described above
if (servers_available != NULL)
     {
   // select fastest server from available servers
   server_best=select_fastest(servers_available);

   // generate communication information
   // for all Arglist by checking input
   // arguments' type
   if LOCAL RESOURCE {
        // allocate local ip and port number
      local ip and port -> ArgList_info;
      }
   else if HANDLER {
      // get ip and port number from JNS
       ArgList_info= myRequest(handler);;
   }

   // make Netsolve non-blocking assignment call
   // and invoke server-side wrapper
```

```
   err=netslnb_assignment("server_best:connector", ProcName,
                                ArgList_info);

    // set up socket waiting for computational
    // server to connect in to download local
    // input data described by Arglist_info.
    for (i=0;i<ArgNum;i++) {
       mysocket_wait( data_input[Arg_num] );
       }

    // waiting until result info is returned
    result_info = mysocket_wait();

    // receive result data from server or
    // submit its info to JNS
    if LOCAL RESULT {
       result = mysocket_get(result_info);
       }
    else if HANDLER {
       myRegister(result_info);
       }
    }
// else if there is no available server satisfy the two
// conditions described above
else
     {
  // connect to JNS to request new variables
  // (corresponding to handlers) information which
  // is created by previous computation.
   hdl_info=myRequest(ArgList);

   // create a new ArgList by checking ArgList,
  // hdl_info and PDF, also new memory allocated
  // variables are created instead of handler to
   // store the data in the function.
     new_ArgList = mycreate(ArgList, getPDF());

     // use original netslnb function to submit task
   err=netsl(ProcName, new_ArgList);

   // register new variables' info to JNS
  myRegister(new_ArgList);
  }
}
```

In *mynetsl()*, if the client can't find any server which both has installed our component and has the request function, it will still run properly by using original *netsl* calling

functions. In particular the data transfer between the client and the server is performed as well with the help of client-side JNS. To select a fastest server among all available servers, the algorithms we use is absolutely same to the one implemented in the NetSolve agent program.

For the client side to run a remote task, originally the servers are selected by the NetSolve agent who is located on one of remote servers. The default setting is to use *netsolve.cs.utk.edu*. System administrator can re-configure it by editing the file "*server_config*" to replace *netsolve.cs.utk.edu* with specified agent (say my.machine.net). The commands are as follows:

UNIX> setenv NETSOLVE_AGENT my.machine.net
UNIX> $NETSOLVE_ROOT/bin/$NETSOLVE_ARCH/NS_agent

In the wrapper library for Client API & Argument Parser, to enable direct communications between remote servers a NetSolve agent will be picked from one of remote servers which have Server Connector installed. The picked agent will work the same way and have full functionalities compared with original NetSolve agent. Only one NetSolve agent can be running on a given machine at a given time. The new agent is responsible for registrations of NetSolve servers wanting to participate in the NetSolve system. After servers are registered, client programs can contact this agent and have requests serviced by one or more of the registered servers to enable direct communications between NetSolve Servers. Currently this process is typically done manually by system administrator. Next step is to implement an algorithm to select a NetSolve agent automatically from one of remote NetSolve servers which have new feature enabled, and investigate how to pick the best agent to execute the remote task. It is one of future development work for this research topic.

4.2.2 Server Connector

On the server side, a proxy program called connector is responsible for interacting with clients and other servers to enable direct communication. The connector is consists of two parts. One is dedicated to pass handlers information between all clients and servers. This lets servers know how to get the data directly. The other part is

responsible for extracting handlers' information and using it to download needed data through direct communication. After all needed data are acquired, it calls the NetSolve functions to perform computation that the user exactly request for.

To send finished computation results to the client or other servers. There is no difference for either client or another computational server to download the result. The server firstly returns result information including IP and port to the client, then waits sockets which acquire its own IP and unique port number to connect in to download result. That connect-in socket could be from a client or any other computational server. This means that the server just passes the result when accepting a legal connection. The pseudo code for connector*()* can be summarized as follows:

```
int connector(ProcName, ArgList_info) {
// check the ArgList_info

// get all input source information by
// extracting ArgList_info
source_info = extract(ArgList_info);

// set up sockets to download all input
// data by using input source information
for (i=0;i<ArgNum;i++) {
   mysocket_get(source_info[Arg_num]);
   }

// call our computational function which
// user want to compute result
result = call(ProcName,input1,input2,…);

// fill result_info with server's ip and port
// number
local ip and port -> result_info;

// return result_info to client
mysocket_send(result_info);

// set up socket waiting for client or another computational
// server to download result
mysocket_wait(result);
}
```

4.2.3 Job Name Server (JNS)

Any procedure registers itself on a dedicated *Job Name Service* (JNS) upon its invocation. Other procedures may send requests to the JNS to search for this registered procedure. JNS is set up on the client side automatically. During the execution of the application, it contains all information about every handler. Only client has the permission to register or access a handler on the JNS. There is no communication and interaction between JNS and computational servers. Handler publication is made by calling *jobPublish(Handler, dataInfo)*, and *jobQuery(Handler, dataInfo)* is used for searching. In the prototype version of the system, we use the following format to label a specific job:

```
<jobAddr>
Handler= "hdlA"
dataInfo[0] = "csa004b3pc2.ucd.ie" // ip
dataInfo[1] = 2919 // port number
dataInfo[2] = 100 // matrix size
dataInfo[3] = 2 // requested times
dataInfo[4] = 0 // broadcast type

……
</jobAddr>
```

In this example, *Handler* contains the name of the handler used in the function prototype. The array *dataInfo* specifies the data's location, data's format details and transaction mode. This information allows the job to be uniquely identified in the network. Different jobs can use the JNS to publish themselves, search others, and exchange data. Also, the JNS is designed as a system-independent system on the client side, so that it can be applied to different RPC-based systems and not influenced by any fault or crash on the server side.

4.3 Installation and Deployment

To use NI-Connect, on the client side, programmer should install the wrapper API and compile the client program with the wrapper library. Our wrapper API allows the

programmers to explicitly specify the dataflow among remote tasks. So they only need to slightly modify their client code, but the principle is quite easy: just replace the input/output arguments with handlers and pass the handlers as the input/ output data. During the initial configuration of Client API & Argument Parsers, note that services/tasks does not need re-compilation when NetSolve system has been configured with enabled Server Connect software component.

The steps of installation of our software component are summarized as follows:

- Run configure program to create a directory named “dc” at the root of NetSolve root directory.
- Install library files including mynetsl.c, mynetsl.h to the dc directory.
- Build lib files for Wrapper API:

```
% gcc –Wall –g –c –o libmynetsl.o mynetsl.c –I$NETSOLVE_ROOT/include
% ar rcs libmynetsl.a libmynetsl.o
```

More details are given in Appendix A.

To enable direct communication in the grid environment on the server side, the procedure programmers should do nothing to enable direct communications. They develop their procedures as usual. The supplementary software component has no effect on both existing procedures and newly added procedures.

For server administrator, on each grid node they need to register the software component as a new problem file to NetSolve. No re-installation and re-compilation to NetSolve itself are needed. The only work is to set up Job Name Service to enable tasks to locate each other. And the software component does not have to be installed on all nodes simultaneously. It can be used to enable direct communications between remote tasks incrementally. It allows for remote calls both to tasks enabled and to tasks not enabled within the framework of the same application.

4.4 Case study: matrix multiplications

To demonstrate the approach, this section presents a case study which enables direct communication in NetSolve in a non-intrusive and incremental way. The case, namely matrix multiplication, uses two remote servers to perform three matrix multiplications, and the client, agent and servers all are in the same Ethernet segment. In this case, there are eight remote servers to perform eight matrix multiplications. The interconnecting network is based on 100 Mbit Ethernet with a switch enabling parallel communications between computers. The client code *WITH* bridge communications looks as follows:

```
/* Compute matrix multiplications */
mynetsl("matmul ()", matA, matB, matC, n);
mynetsl("matmul ()", matC, matD, matE, n);
mynetsl("matmul ()", matE, matF, matG, n);
mynetsl("matmul ()", matG, matH, matI, n);
mynetsl("matmul ()", matI, matJ, matK, n);
mynetsl("matmul ()", matK, matL, matM, n);
mynetsl("matmul ()", matM, matN, matO, n);
mynetsl("matmul ()", matO, matP, matQ, n);
```

The client code with direct communications is as follows:

```
/* Compute matrix multiplications */
mynetsl("matmul ()", matA, matB, hdlC, n);
mynetsl("matmul ()", hdlC, matD, hdlE, n);
mynetsl("matmul ()", hdlE, matF, hdlG, n);
mynetsl("matmul ()", hdlG, matH, hdlI, n);
mynetsl("matmul ()", hdlI, matJ, hdlK, n);
mynetsl("matmul ()", hdlK, matL, hdlM, n);
mynetsl("matmul ()", hdlM, matN, hdlO, n);
mynetsl("matmul ()", hdlO, matP, matQ, n);
```

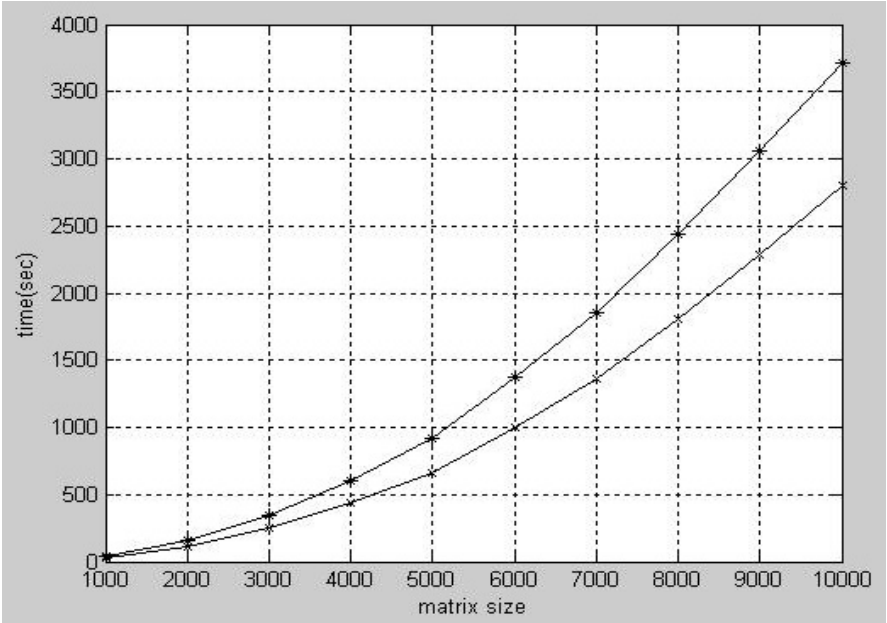
Parameter *n* is the dimension of matrices. *matA*, *matB*, *matC*, *matD*, *matE*, *matF*, *matG*, *matH*, *matI*, *matJ matK*, *matL*, *matM*, *matN*, *matO, matP* and *matQ* are matrix data. h*dlC, hdlE, hdlG, hdlI, hdK, hdlM* and *hdlO* are handlers, which are used to eliminate bridge communication. In the experiments, we only measure the communication time of trails.

In this case there are three trails for matrix of different sizes. Experimental results are presented in Table 4.1.

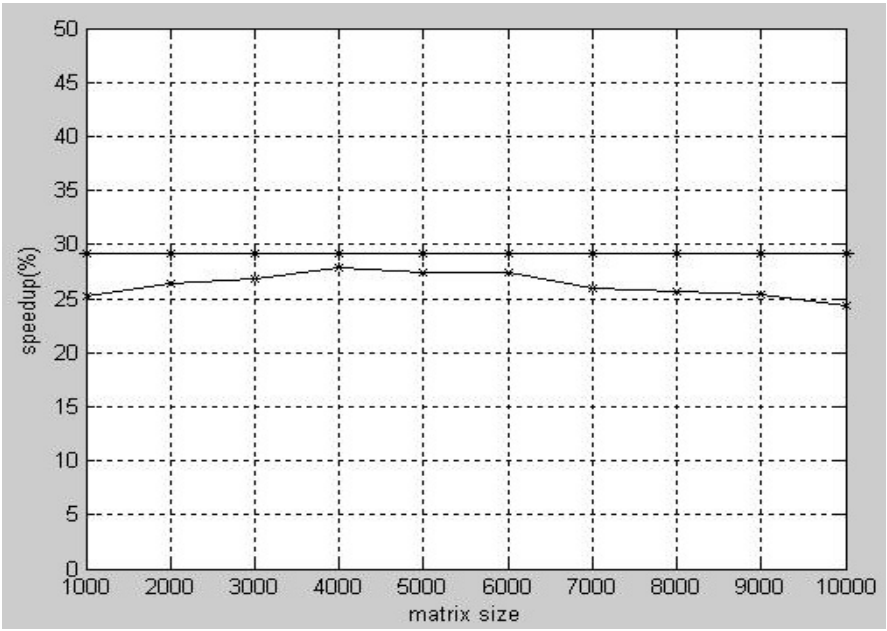
Size	**Trail 1**		**Trail 2**		**Trail 3**		**Average**		**Speedup**
	B	**D**	**B**	**D**	**B**	**D**	**B**	**D**	
1000	38.3	28.7	39.5	29.2	38.6	29.1	38.8	29	25.2%
2000	155.5	115.7	151.2	113	153.4	110	153.4	112.9	26.4%
3000	342.9	238	345	255	340.8	260	342.9	251	26.8%
4000	607	428	604	436	611	450	607	438	27.8%
5000	920	691	923	671	908	636	917	666	27.4%
6000	1354	901	1379	1005	1402	1094	1378	1000	27.4%
7000	1840	1391	1810	1392	1895	1321	1848	1368	26.0%
8000	2460	1773	2395	1810	2453	1853	2436	1812	25.6%
9000	3069	2349	3095	2298	3023	2205	3062	2284	25.4%
10000	3563	2670	3810	2894	3750	2845	3708	2803	24.4%

Table 4.1 Comparison of different communication Approaches

Figure 4.2(a) shows the communication time as a function of matrix size. Figure 4.2(b) shows the speedup of the application with direct communications.



(a)



(b)

* bridge communication x direct communication

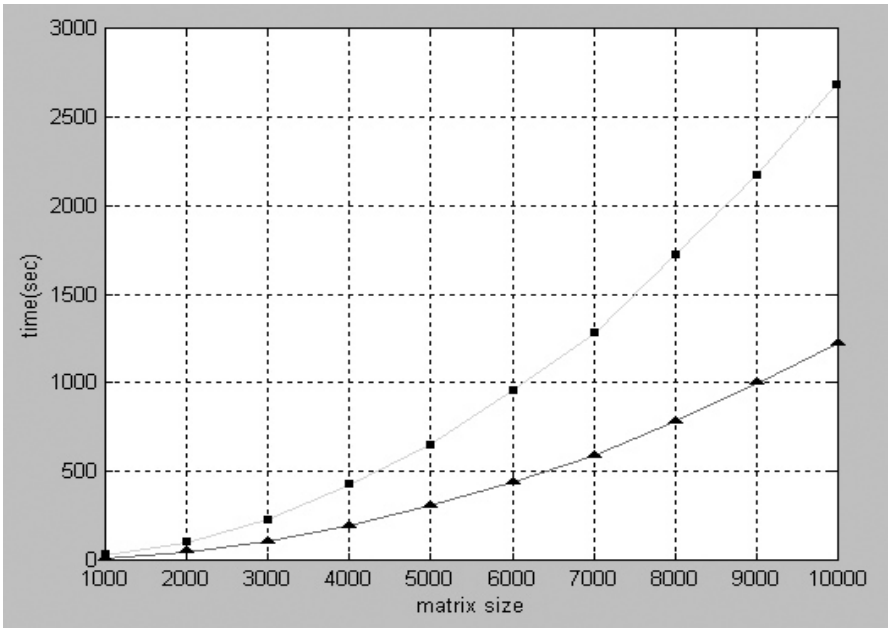
Figure 4.2 Experimental results of matrix multiplication

(a) Time elapsed for both communication types when all communication links have the same bandwidth, 100Mb per sec. (b) Speedup due to the use of direct communications for the homogeneous communication network.

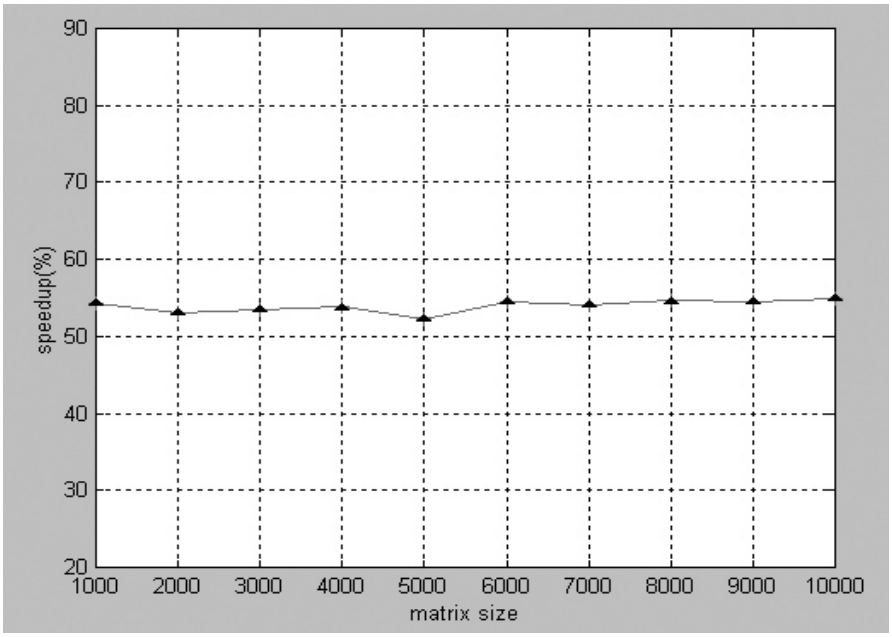
As expected, the communication cost is visibly reduced by using direct communications. In the experiments, seven communication bridges are eliminated among twenty four communications. So, the theoretical speedup is 7/24 = 29.2%. The obtained experimental speedup ranges from 24% to 27%, which is close to the theoretical value. We can also see that the experimental results are similar to the REDGRID ones, which range from 18% to 28%. In the Table 4.1 experiment data is given to show the speedup for different ratio of eliminated bridge communication, while we choose same matrix size 10000 for each case. The result of experiments shows that the speedup using new approach is crescent while the ratio of eliminated bridge communication is increasing. If the network interconnecting the computers is based on same bandwidth connection, limit value of speed up is as follows:

$$\lim_{n->\infty}(\frac{n-1}{n\times 3}) = 1/3 = 33.3\%$$

If communication links connecting remote computers are much faster than communication links connecting the remote computers and the client computer, the speedup due to elimination of bridge communications will be much higher. To corroborate it, we design another experiment. We manually make all bridge communications be performed at the rate of 10 Mbit per second. For the direct communications between remote servers, we still use 100 Mbit Ethernet interconnecting network. Figure 4.3(a) shows the communication time for this configuration of the communication network. Figure 4.3(b) presents the speedup of the application with direct communications over the one with bridge communications in this case. The experimental speedup is around 54% when the ratio of eliminated bridge communications is 2/9. Thus, much higher speedup can be achieved in heterogeneous communication networks, which are more typical for real-life Grid environments than in artificially designed homogeneous ones.



(a)



(b)

■ bridge communication ▲ direct communication

Figure 4.3 Experimental Results in Heterogeneous Network

(a) Time elapsed for both communication types when communication client and servers is at the rate of 10 Mb per sec, and communication between servers is at the rate of 100 Mbit per sec. (b) Speedup due to the use of direct communications for the heterogeneous communication network.

4.5 Contribution

The main advantages of the approach we have showed in this chapter are as follows:

- The approach is non-intrusive, requiring no changes in the enabled programming system.
- It does NOT need recompilation or reinstallation of the Grid programming system.
- The approach is incremental by nature allowing direct communication enabled for remote tasks, and to be freely mixed in a single application.
- Programmers are given the ability to explicitly specify the data flow in their code.
- Finally the experimental results of case study have shown that the performance of Grid applications can be significantly improved by using our supplementary software component.

It is proved in this chapter that a selected Grid Programming System NetSolve can be enabled with new feature in a non-intrusive and incremental way. Next, we will study how to apply the approach to different RPC-based Grid programming systems.

Chapter 5

Generic Implementation of Non-intrusive and Incremental Approach

5.1 Targets of Generic Approach

In Chapter 4, a supplementary software component NI-Connect has been implemented for case study which enables direct communication in NetSolve in a non-intrusive and incremental way. The Non-intrusive and Incremental approach for evolution of Grid Programming Systems is proved feasible. The experimental results show higher performance to run the applications on several computational servers by performing a series of matrix multiplication. As a result, the feasibility and performance of our approach is demonstrated with targeted Grid Programming System and particular feature. Next, we formulate principles and standards for generic implementation of non-intrusive and incremental approach for general Grid Programming Systems to enable new features. The research work in this chapter is to achieve following targets:

- Formulate the principles and standards to generate generic software component for all GridRPC-based grid programming systems.
- Design a generic non-intrusive and incremental structure to enable new features in GridRPC-based grid programming system.

- List the libraries and components that can be easily re-used for generic implementation of Non-intrusive and Incremental approach.

5.2 Principles and Standards

Same to the NI-Connect, the features of generic software component are Non-intrusiveness and Increment. Non-intrusiveness means that the original system does not change and the new feature is provided by a supplementary software component working on the top of the system. Increment means that the software component does not have to be installed on all computers to enable applications with the new feature. It can be done incrementally, and the new feature will be enabled in part. The implementation of generic software component is based on client-server model as well, which consists of three parts: local development library, JNS components, and add-on servers with control functions.

Local Development Library: It is used to develop client-side calling procedure in the grid programming system. It provides a uniform interface for the client user to make remote procedure calls. Client API & Argument Parser which is already developed in NI-Connect is major part of local development library. Despite the modification on the remote side, the API allows the calls to be made in the same manner. The only difference is in the arguments that can be not only variables storing real data but also handlers. Like in NetSolve, we parse the list of arguments to construct the handler array. For each argument, the relevant communication information is generated. For each input argument, which is a variable storing real data, the local IP address and the port number are used as such communication information. If this input argument is a handler, then a request is sent to the JNS to get the IP address and the port number of the remote resource and this information is used as communication information for this handler. For each output argument, which is a variable storing real data, the client wrapper function will set up a socket to download output data from computational servers. If this output argument is a handler, the returned result information from computational servers is sent to JNS and registered there.

Job Name Service (JNS) component: It is responsible for registration of a procedure upon its invocation during RPC call. As same to the JNS that we have developed for

NI-Connect, for the other generic software component to enable new features for GridRPC-based system the functions are absolutely same. Each task running in the remote servers called through NI-Connect Client API are registered on the local name server. Other procedures may send requests to the JNS to search for the registered procedure. JNS is set up on the client side automatically upon each task is scheduled. During the execution of the application, it contains all information about every handler. Only the client has the permission to register or access job registration information which includes application definition class, grid resources running the task, etc. There is no direct communication and interaction between JNS and computational servers. Because JNS is designed as a system-independent system on the client side, it can be applied to different RPC-based systems and not influenced by any fault or crash on the server side.

Add-on Servers and Control functions: Server Connector in NI-Connect is an implementation for NetSolve to enable direct communications. It is on the server side, which is a proxy program responsible for interacting with clients and other Server Connectors to enable direct communications. For generic non-intrusive and incremental software component, it is quite similar with extensions of control and management functions. Add-on Server consists of three main parts. The first part is to pass handler information between clients and servers. This allows servers to know how to get the data without bridge communication. The second part is the extraction of the handlers' information and using it to download needed data through direct communication. After all the needed data have been acquired, the Server Connector calls the procedure to re-submit to the local host to perform computations that the user exactly requested for. There is no difference in the way the client and computational servers download the result of the computations. The third part is management interface for administrator to enable the new features on the server side. It allows administrator to easily switch the status of server connector by start or kill the process running on the servers.

The use of generic software component to enable new features in GridRPC-based Programming System should follow the standards below:

- Programmers have to install the wrapper API and Job Name Service on the client side and then compile the client program with the wrapper library.
- The wrapper API allows the programmers to explicitly specify the dataflow between remote tasks. They only need to slightly modify their client code.
- The procedure developers should do nothing to enable new features. They develop their procedures as usual. The generic software component has no effect on both existing procedures and newly added procedures.
- To enable or disable the new features on the server side, the server administrator needs to install an add-on Servers. No reinstallation and re-compilation of either GridRPC system itself or registered application procedures are needed.

5.3 Generic Structure of Software Component

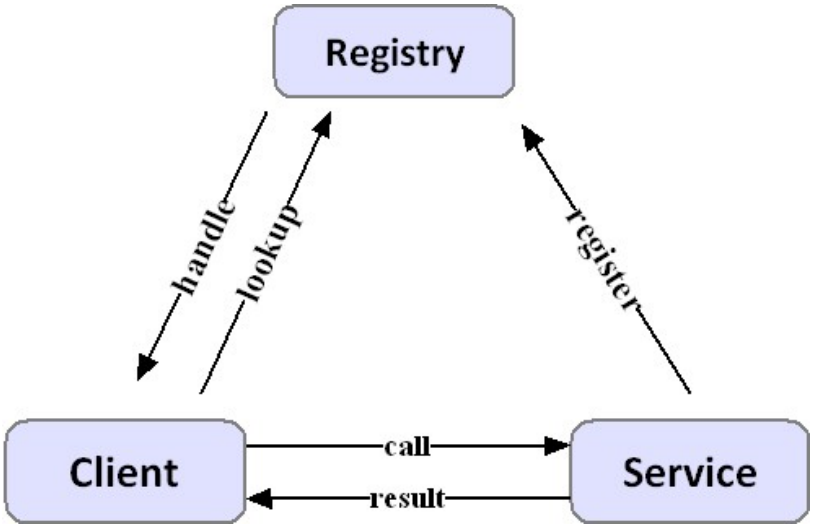


Figure 5.1 Original GridRPC Model

The original GridRPC model typically consists of three parts: Client, Registry and Service. It is shown in Figure 5.1. On the client side, in order for a task to be available on a server, a programmer has to define the specific information that describes various aspects of the remote task. Each server of the Grid environment registers its tasks available in a Registry. This involves the servers sending the task's information to the Registry. The Registry is an abstract term that could indicate a single entity or several

entities, which works as a resource discovery. On the server side, once a particular task-to-server mapping has been established by initialising a task handle, all GridRPC task calls using that function handle will be executed on the server specified in that binding. Each GridRPC task call gets processed individually, where each task is discovered (task look-up) and executed separately from all the other tasks in the application.

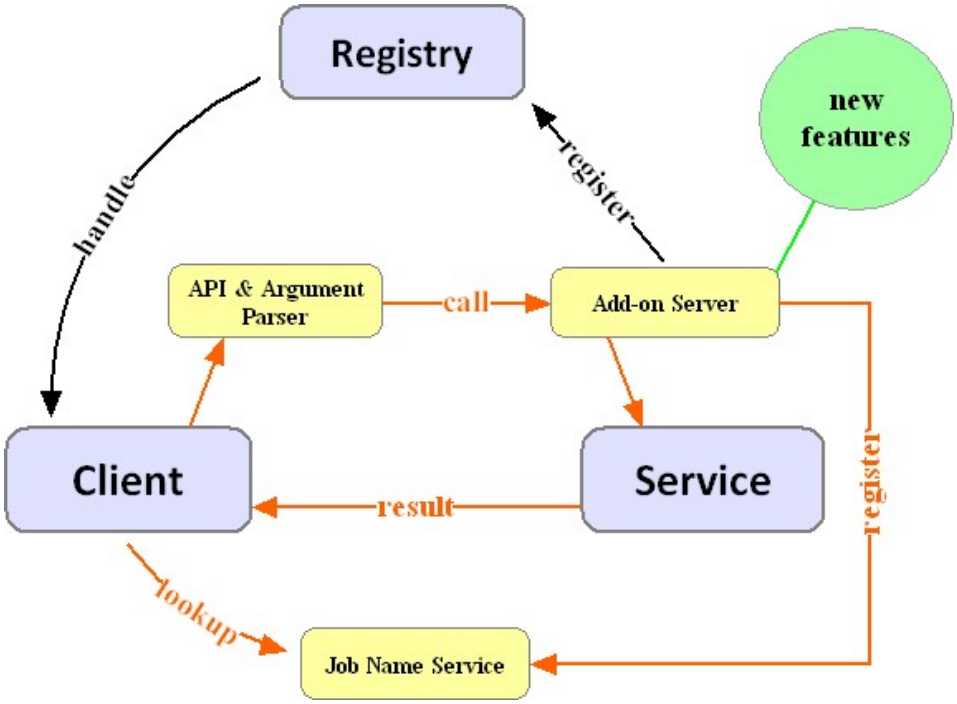


Figure 5.2 Generic Non-intrusive and Incremental Approach Model

Compared to original GridRPC model, generic software component is implemented in a non-intrusive and incremental way based on GridRPC model. It is shown in Figure 5.2. Generic Non-intrusive and incremental approach has following features:

- Client does not communicate with server directly. Services on the server side are called through add-on servers. If add-on servers are not enabled on the grid nodes, the original grid remote procedure call will be invoked.
- Client submits remote procedure calls to add-on servers through API & Argument Parser. The programs are developed using development library on the client side.

- Add-on servers are registered as a server-side application on the original Registry in the GridRPC system. By contrast, all services are registered on Job Name Server.
- Client looks up Job Name Server for the information of all computational tasks and receives the result from computational server directly.
- Add-on Servers look up Job Name Server for the information of all computational tasks and manage computing tasks.
- New features are added into the GridRPC system in a form that adding extended functions to the add-on servers.
- Add-on servers on different gird nodes can communicate with each other directly. Those GridRPC servers without add-on servers enabled can only communicate with client.

Generic software component is built on top of GridRPC system and there is no change at all in the original Grid Programming System.

5.4 Libraries and Components

In this section it lists major available APIs and component which are implemented during the development of NI-Connect. All these functions can be re-used for implementation of other generic software component to enables new features to GridRPC system in a non-intrusive and incremental way.

5.4.1 Client-side Functions

array **ni_client_parse** (string[] *$process_array*, int *$parse_mode*)

ni_client_parse() returns parsed calling parameter from handler array.
$process_array: application name and handler array
$parse_mode: 0 - disable all features, 1- enable new features

string **ni_client _lookup** (string *$handler*)

ni_client_lookup() *returns the details for the specific handler.*
$handler: client defined handler to describe tasks

int **ni_client _register** (string *$task*, string[] *$list_server_info*)

ni_client_register() *register specific task on Job Name Server.*

$task: computational task to run on the grid environment.

$list_server_info: list of servers' information which are performing tasks.

int **ni_client_connect** (string *$add_on_server*, int *port*)

ni_client_connect () *set up connection to add-on server using ip address and port number.*

$add_on_server: information of add-on server

$port: port number to connect to add-on server

data **ni_client_collect** (string *list_server_info*, int *$block*)

ni_client_collect() *returns output data from list of add-on servers.*

$list_server_info: list of servers' information which are performing tasks.

$block: 0 - non-block mode, 1 - block mode.

int **ni_client_close** (string *$task*)

ni_client_close() *stop/clear all computations for the specific task.*

$task: computational task to run on the grid environment.

5.4.2 Server-side Functions

string[] **ni_server_discover** (string *$task*)

ni_server_discover() *returns list of servers' info which can perform specific task.*

$task: computational task to run on the grid environment.

string **ni_server_lookup** (string *$handler*)

ni_server_lookup() *returns the details for the specific handler from JNS.*

$handler: client defined handler to describe tasks

int **ni_server_register** (string *$task*, string *$server_info*)

ni_server_register() *register add-on server and specific task on Job Name Server.*

$task: computational task to run on the grid environment.

$server_info: one of add-on's servers' information.

int **ni_server_listen** (conn *$connection*, string *$server_info*)

ni_server_listen() *set up listen thread on the add-on server for incoming connection*

$connection: listening thread for incoming connection.

$server_info: one of add-on's servers' information.

int **ni_server_comm** (string[] *list_server_info*, data[] *$data*, string *$task*)

ni_server_comm() *setup communications between add-on servers.*

$list_server_info: list of servers' information which are performing tasks.

$data: data needed for finishing computations of task.

$task: computational task to run on the grid environment.

data **ni_server_return** (string *$task*, int *$sync*)

ni_server_return() *returns output data from add-on server to client.*

$task: computational task to run on the grid environment.

$sync: 0 - synchronies mode, 1 – asynchronies mode.

int **ni_server_**op (time *$sec*)

ni_server_op() *turn on/off add-on server on the grid node.*

$sec: action delayed seconds.

5.4.3 Job Name Server (JNS)

string[] **ni_jns_retrieve** ()

ni_jns_retrieve() *list all tasks registered on Job Name Server.*

int **ni_jns_insert** (string *$task*)

ni_jns_insert() *insert a new task records into Job Name Server.*
$task: computational task to run on the grid environment.

int **ni_jns_update** (string *$task*)

ni_jns_update() *update an existing task records on Job Name Server.*
$task: computational task to run on the grid environment.

int **ni_jns_remove** (string *$task*)

ni_jns_remove() *delete an existing task records on Job Name Server.*
$task: computational task to run on the grid environment.

5.5 Practices and Challenges

To generically implement non-intrusive and incremental software component for Grid Programming Systems to enable new features, we have formulated principles and standards, described the structure of developing generic software component, and roughly listed the available development libraries and functions. For the programmers who get into detailed development of generic software component, there are a few practices tips and possible challenges listed below:

- Most new features can be easily implemented within generic software component in a non-intrusive and incremental way by borrowing the existing APIs either from the targeted Grid Programming Systems, or using libraries and functions developed in our research work.
- Most work for building generic software component should be implementing new features based on current model since the non-intrusive and incremental approach is to minimize the development work for the users.
- There is possibility that the targeted Grid Programming System release a new version which certain amount of related work for the software component need to be done.
- Different Grid Programming Systems are deployed on different grid nodes, where sometimes the ranges of blocked ports are different. To avoid this

firewall-like problem, developers will need to develop flexible programs to automatically skip the difficulties.

- Last, make sure to achieve better quality of generic software component, as a result testing is needed.

Next part of thesis will focus more on the experiments and applications in different cases. Performance analysis is also discussed by using Non-intrusive and Increment Approach to add new features for Grid Programming System.

Part III

Application and Experiments

Chapter 6

Real-world Applications with Different Commutation Structures

6.1 Overview

This thesis reports on experiments with three typical scientific NetSolve applications having different communication structures: (i) protein tertiary structure prediction, (ii) image processing using sequential algorithms, and (iii) the matrix chain product. The presented experimental results show that the performance of these Grid applications can be easily and significantly improved by using the proposed supplementary software component [ZL07].

.

6.2 Algorithms and Network Resources

Currently, we have a prototype implementation of the software component. It is interesting to point out that despite all of the above discussions assume NetSolve as the target, none in the implementation relies on this particular system. This makes sense to apply the same approach on other Grid RPC systems. Since we provide the inter-job communication as external function, a side-effect is that it is possible for the client to connect the calls of different Grid RPC. To prove that our approach can improve the performance of RPC-based Grid programming systems in the area of scientific research by using our supplementary software component, a series of real-world computing application need to be tested and analyzed. In this paper, we have selected three typical applications with different communication structures. Our software component is used for these applications to enable direct communications in a Non-Intrusive and Incremental way. Experimental environment is an interconnecting network based on 100 Mbit Ethernet with a switch enabling parallel communications between servers in School of Computer Science and Informatics in University College Dublin. The specifications of the servers are shown in Table 6.1.

Name	Architecture	Cpu (Mhz)	Main Memory (mb)	Cache (kb)	Relative speed (mxm)
Pg1cluster01	Linux 2.6.8 - 1.521 smp Intel(R) EON™	2048	1024	512	341
Pg1cluster02	Linux 2.6.8 - 1.521 smp Intel(R) EON™	2048	1024	512	341
Pg1cluster03	Linux 2.6.8 - 1.521 smp Intel(R) EON™	2048	1024	512	341
Csultra01	SunOS 5.8 sun4u sparc SUNW, Ultra-5_10	440	512	2048	175
Csultra02	SunOS 5.8 sun4u sparc SUNW, Ultra-5_10	440	512	2048	100
Csultra03	SunOS 5.8 sun4u sparc SUNW, Ultra-5_10	440	512	2048	100
Csultra04	SunOS 5.8 sun4u sparc SUNW, Ultra-5_10	440	512	2048	100
Csultra05	SunOS 5.8 sun4u sparc SUNW, Ultra-5_10	440	512	2048	100

Table 6.1 Installation and specifications of computational nodes

6.3 Genetic Crossover in Protein Tertiary Structure Prediction

Protein tertiary structure prediction systems are proposed for progress of the bioinformatics which is mainly performed by the protein energy minimization. However, large-scale computing environment would be valuable for this system. In the system, Parallel Simulated Annealing using Genetic Crossover (PSA/GAc) [HMO00] is a minimization engine. To use the grid resource, NetSolve is a basic tool and implementations are already prepared [TAH04] to improve the computing performance. Their approach has reduced critical overhead due to large communication delay over the Internet by using asynchronous Crossover model. But bridge communication still exists and these un-necessary communications can be eliminated by using our software component.

6.3.1 Introduction and Analysis

Figure 6.1 shows both synchronous and asynchronous Master-slave models for Genetic Crossover in protein tertiary structure prediction system. Figure 6.2 depicts how bridge communications occurs between NetSolve servers while performing Genetic Crossovers.

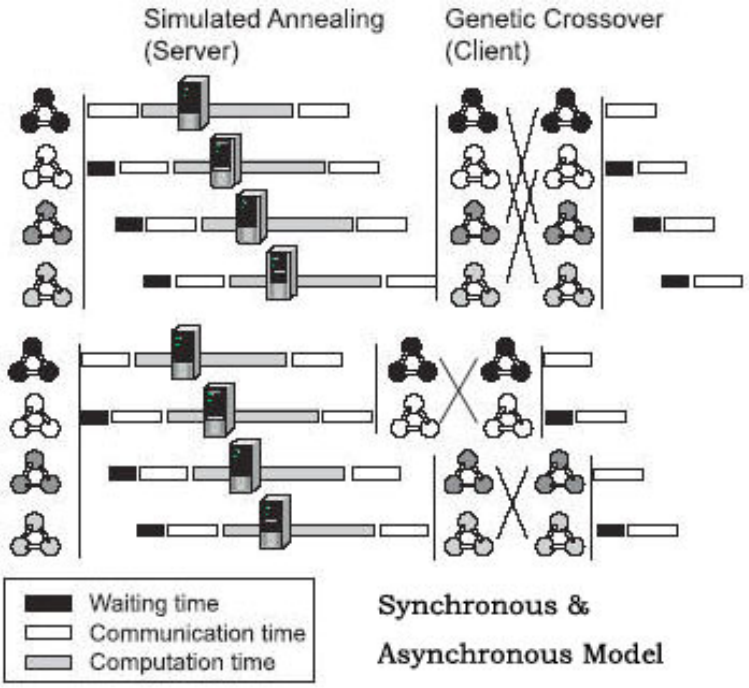


Figure 6.1 Synchronous and asynchronous models for PSA/Gac

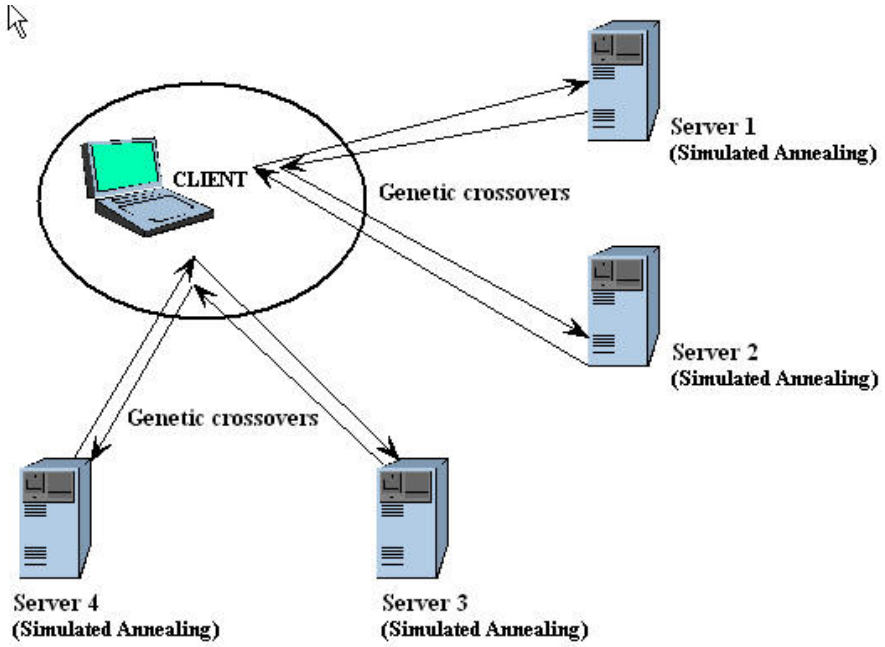


Figure 6.2 Bridge communications between NetSolve servers

6.3.2 Optimize communication structure by using NI-Connect

Figure 6.3 depicts how to enable direct communications between NetSolve servers while performing Genetic Crossovers.

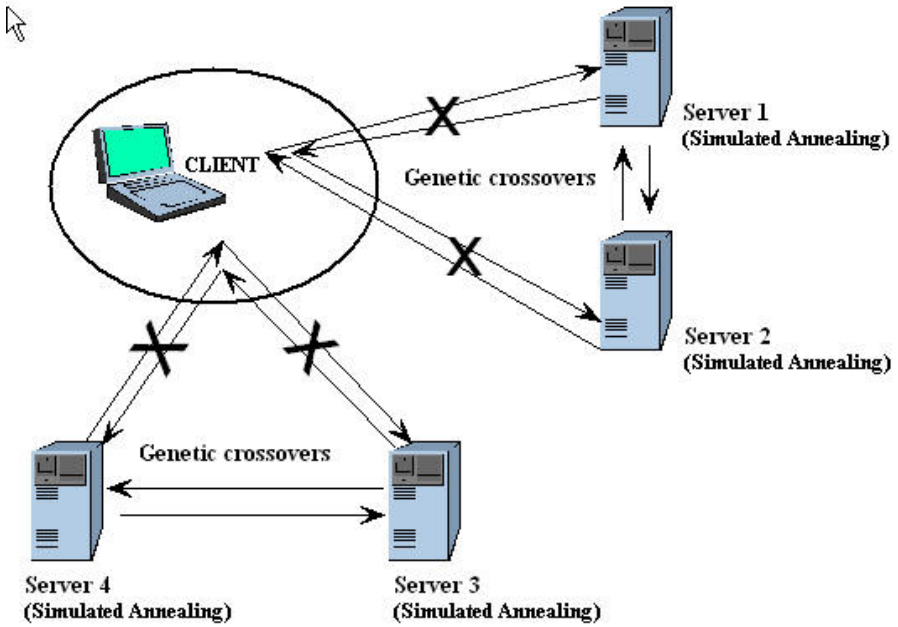


Figure 6.3 Enabling direct communications between NetSolve servers

6.3.3 Results and Conclusion

By enabling direct communication using our Non-Intrusive and Incremental Approach, Genetic Crossovers is executed between servers directly. The original approach must depend on client-side. Direct communication is enabled between Server 1 and Server 2, and between Server 3 and Server 4. Simulated Annealing is executed on each server separately. So the exchanging data is not returned to the client while direct communication is enabled. This reduces communication links between the client and servers. Figure 6.3 shows that communication links are reduced from 8 to 4 by using our software component in NetSolve.

Table 6.2 shows experimental results of three trails with different sizes of protein. It gives communication time of the original NetSolve application using bridge communications and the modified application employing direct communications. The average communication speedup due to elimination of bridge communications is around 43%.

Protein Size (kb)	Trail 1		Trail 2		Trail 3		Average		Speedup
	B	D	B	D	B	D	B	D	
1000	50	30	51	30	53	31	52	30	45%
2000	106	62	108	63	108	62	107	62	42%
3000	175	98	170	100	178	105	174	101	42%

Table 6.2 B – Bridge communication time (in seconds); D – Direct communication time (in seconds).

6.4 Image Processing Using Sequential Algorithms

So far, image and video processing software has been predominantly written for conventional (sequential) desktop computers and embedded digital signal processors (DSPs), which implement a wide range of operations [WRS98] such as smoothing, sharpening, noise reduction, etc. These applications usually have a tremendous potential for parallelism but unfortunately, existing techniques are not adequate for

compiling sequential multimedia programs to such parallel architectures. Therefore, some researchers focus on extracting the essential computations and data dependency to ensure that each computation has the data it requires [BW02] [CAS95]. Our research aims to optimize communications of data transaction for sequential multimedia operations. The method is to enable direct communications for sequential image processing by using our supplementary software component. For experiments, we chose an example, which is Simple Linear Combination Filtering [Slc]. Linear combination filtering functions are taken from Image Processing Library 98 [IPL98].

6.4.1 Introduction and Analysis

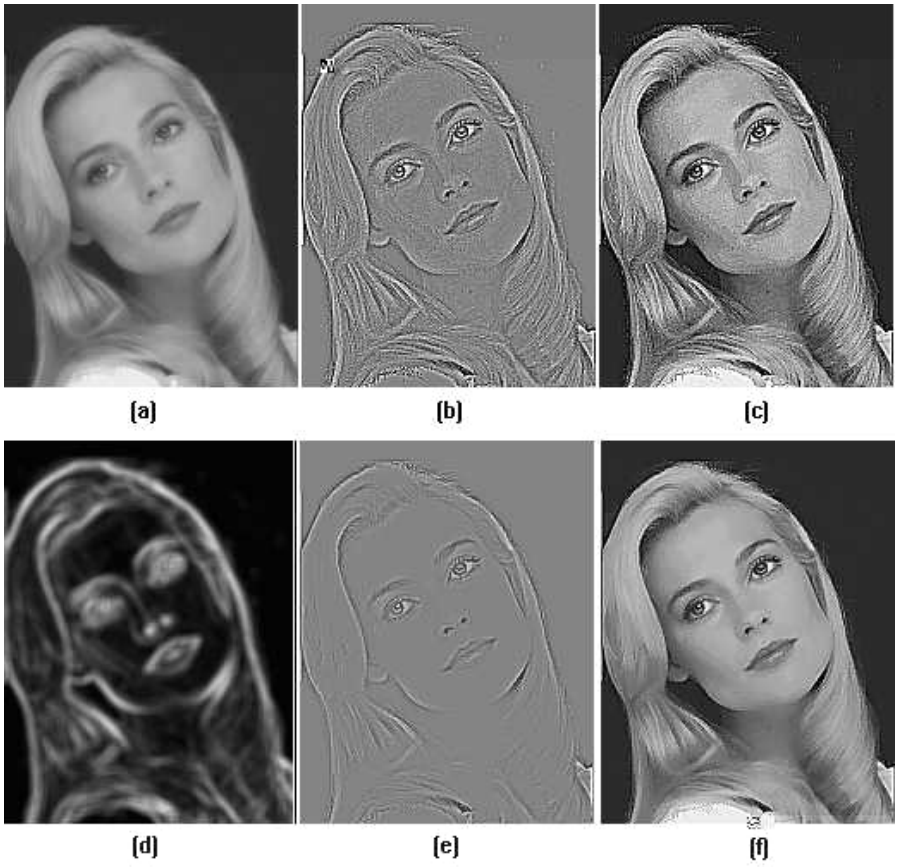


Figure 6.4 Simple Linear Combination Filtering

(a) Input image; (b) Laplacian of (a); (c) Spatially invariant high-pass filtering [sum of (a) and (b)]; (d) Mask image [Sobel gradient of (a) smoothed by a 5x5 box filter]; (e) Product of (b) and (d); (f) Space-variant enhancement [sum of (a) and (e)].

For image enhancement, linear combination filtering can blur smooth parts of an image while sharpening areas that contain detail. The reason for this combination is that blurring reduces noise, but degrades edges and image detail while sharpening enhances edges and detail but makes noise more visible. Figure 6.4 displays the example pictures of simple linear combination filtering. Figure 6.5 depicts how bridge communications between the client and NetSolve servers are replaced by direct communications between NetSolve servers while performing linear combination filtering functions in this case.

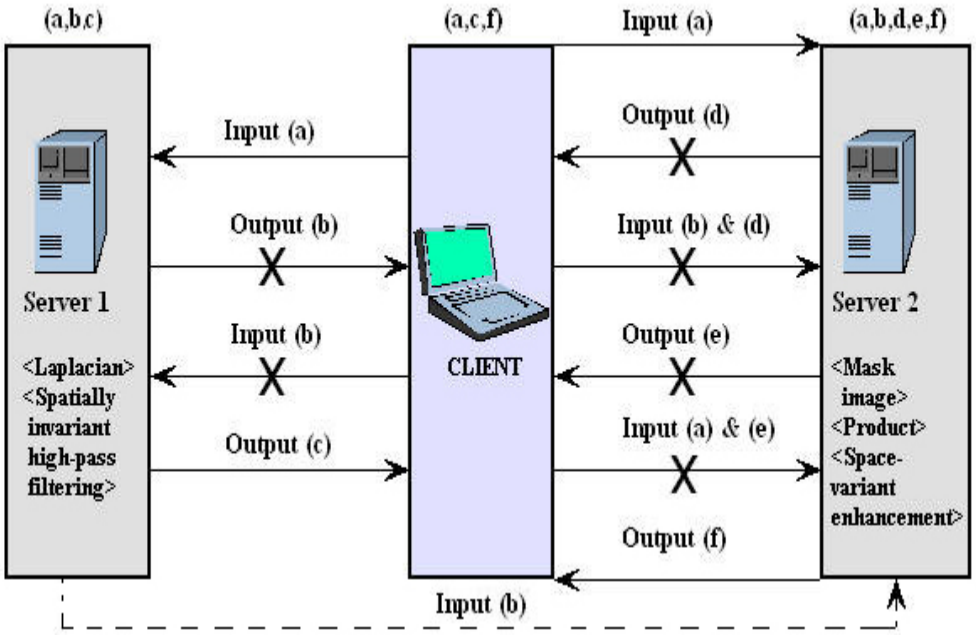


Figure 6.5 Communication structure of linear combination filtering

6.4.2 Optimize communication structure by using NI-Connect

To eliminate un-necessary communications between the client and the servers while performing linear combination filtering, we select two servers to perform linear combination filtering functions in parallel:

Server 1:

- Laplacian of image (a);
- Spatially invariant high-pass filtering, sum of image (a) and image (b);

Server 2:

- Mask image, Sobel gradient of image (a) smoothed by a 5x5 box filter;
- Product of image (b) and image (d);
- Space-variant enhancement, sum of image (a) and image (e);

Direct communications are enabled between the servers by transferring image (b) from Server 1 to Server 2 directly. Those images, which will be used as input for other image processing functions, will NOT be returned to the client. This reduces bridge communications between the client and servers. We can see that in Figure 3 communication links are reduced from ten to five by using our software component. Only necessary images (a), (c) and (f) will be on the client side.

6.4.3 Results and Conclusion

We experimented with the linear combination filtering application for pictures of different sizes. Table 6.3 shows experimental results of three trails with matrix of different sizes. The results show that the average communication speedup is around 50%. This is due to the fact that six communication bridges were eliminated and one direct communication was established between two servers.

Picture Size (kb)	Trail 1		Trail 2		Trail 3		Average		Speedup
	B	D	B	D	B	D	B	D	
1000	60	29	60	29	61	29	60	29	51%
2000	125	61	122	62	125	63	124	62	50%
3000	195	97	209	98	203	98	200	98	51%

Table 6.3 B – Bridge communication time (in seconds); D – Direct communication time (in seconds)

6.5 Matrix chain product problem in general scientific computations

Given N matrices A*1*, A*2*, …, A*n* of size N × N, the matrix chain product problem is to compute A*1* × A*2* × … × A*n*. The matrix chain product is an important computational kernel that is used in computing the characteristic polynomial, determinant, rank, and inverse of a matrix, in solving graph theory problems, and in general scientific computations [Lk01][Lk02]. In the thesis, we have manipulated a sequential matrix multiplication by enabling direct communication in a *non-Intrusive* and *incremental* way. The approach proved that the performance can be significantly improved by using our supplementary software component in NetSolve. In this paper, our goal is to demonstrate our approach can even improve the performance of matrix chain product computation problems in general scientific computations.

6.5.1 Introduction and Analysis

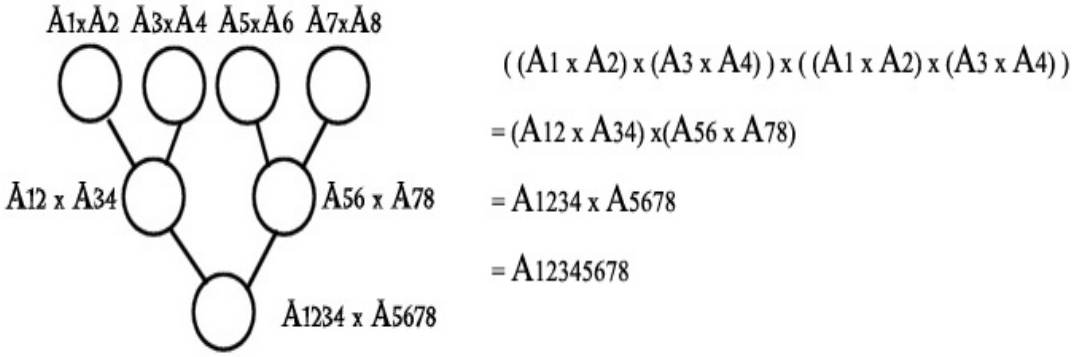


Figure 6.6 Standard binary tree method used for matrix chain product problem.

In Figure 6.6, it shows how the product of A*1*, A*2*, …, A*8* can be obtained by using the standard binary tree method. The leaves are input matrices A*1*, A*2*, …, A*8*, and the root task of the tree computes the final result A*12345678*. Figure 5 depicts how bridge communications between the client and NetSolve servers are replaced by direct communications between NetSolve servers for matrix chain product computation, where six communication bridges were eliminated among the total fourteen. In figure 6.7, it describes bridge communication during computation of matrix chain product.

In this experiment, we select four servers to perform matrix chain product computation in parallel:

Server 1: - perform A1 × A2;

Server 2: - perform A3 × A4, A12 × A34;

Server 3: - perform A5 × A6;

Server 4: - perform A7 × A8, A56 × A78, A1234 × A5678;

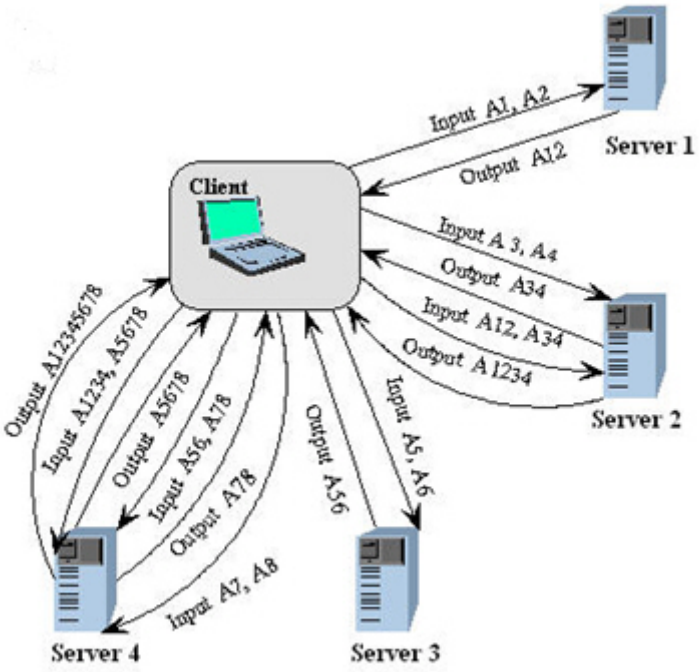


Figure 6.7 Bridge communications between NetSolve servers for matrix chain product computation.

6.5.2 Optimize communication structure by using NI-Connect

Direct communication is enabled between these four servers by directly transferring output A12 from Server 1 to Server 2, output A56 from Server 3 to Server 4, output A1234 from Server 2 to Server 4. These output matrices will NOT be returned to the client. This reduces bridge communications between the client and servers. Communication links are reduced from 14 to 8 by using our software component in

NetSolve. Only required result matrix A12345678 is returned to the client side. Figure 6.8 describes how to enable the direct communication in the structure of matrix chain product.

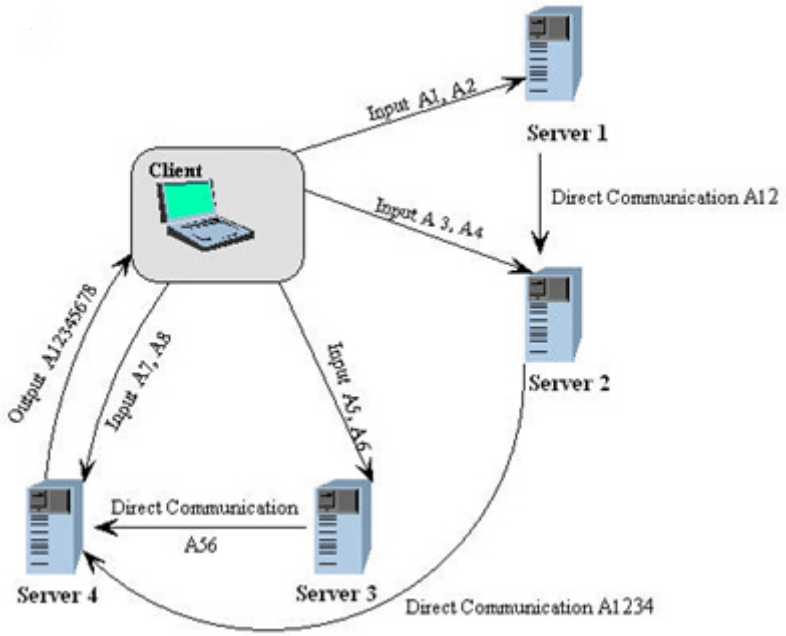


Figure 6.8 Enabling direct communications between NetSolve servers for matrix chain product computation.

6.5.3 Results and Conclusion

Matrix Size	Trail 1		Trail 2		Trail 3		Average		Speedup
	B	D	B	D	B	D	B	D	
1000	102	66	101	67	103	67	102	67	38%
2000	210	132	220	136	212	138	214	135	36%
3000	335	220	315	226	310	216	320	221	31%

Table 6.4 B – Bridge communication time (in seconds); D – Direct communication time (in seconds)

In our experiments we select different sizes of matrix for computation. Table 6.4 shows experimental results of three trails with different sizes of picture. The experimental results show that communication time is reduced by around 35%, where 6 communication bridges are eliminated among 14 communications. This significantly improves the performance of matrix chain product computation by using our Non-Intrusive and Incremental Approach to enable direct communications in NetSolve.

In this chapter, by using the supplementary software component to enable direct communications in NetSolve the average speed up for selected real-world scientific applications are within the range of 30% to 50%. This experimental result does not indicate that general real-world scientific applications can achieve similar benefits. The experimental results we get in this thesis are based on selected networks which are described in table 6.1. The servers to run experiments have relative high performance of computing power rather than inter-connected communication speed. Also, the computation time and communication time for selected scientific applications have same order of magnitude. As a result, such experiment environment can have significant impact showing the benefits of enabling direct communications between remote NetSolve servers.

Chapter 7

Large-scale Experiments on Heterogeneous Grid Environment

7.1 Objective

One feature of our approach is increment. It means that the supplementary software component does not have to be installed on all computers to enable applications with direct communications. In this case, direct communications can only happen between those computing nodes, where our supplementary software component is installed. Non-enabled computing nodes can only communicate with the client. The speedup of a NetSolve application due to the use of our software component depends on how large the fraction of computing nodes with enabled direct communications in the overall set of computing nodes used by the application is.

7.2 Using NI-Connect in Heterogeneous Network

This section presents experiments to investigate the use of NI-Connect in heterogeneous networks.

7.2.1 Homogeneous and Heterogeneous Computing

There are three main issues determining the classification which are the hardware, the communication layer, and the software (operating system, compiler, compiler options).

Any differences in these areas can potentially affect the behaviour of the application. As a result, the definition of homogeneous computing environment is:

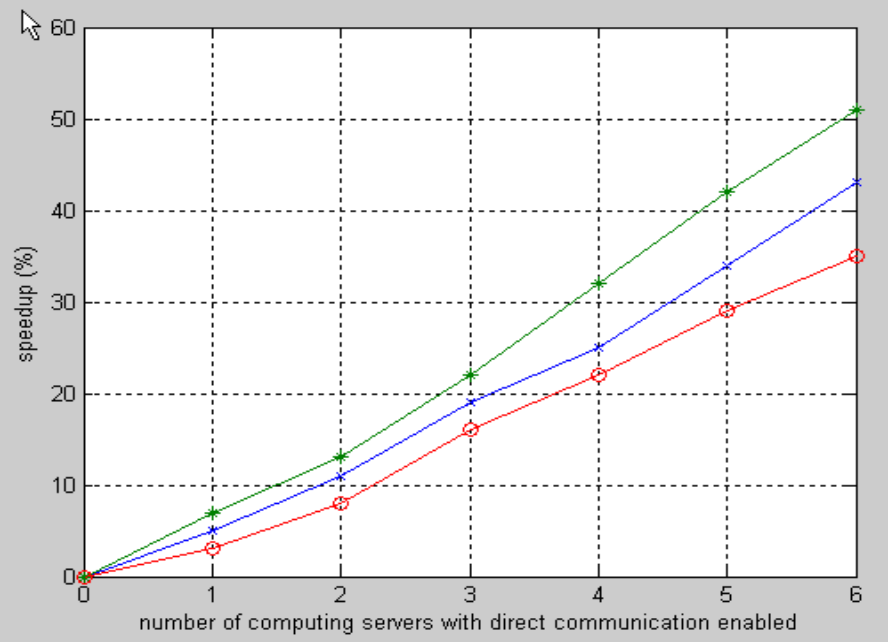
- The hardware of each processor guarantees the same storage representation and the same results for operations on floating point numbers.
- Communication layer guarantees the exact transmittal of the floating point value between processors.
- The software on each processor also guarantees the same storage representation and the same results for operations on floating point numbers.

By contrast, we can then make the obvious definition that a heterogeneous computing environment is one that is not homogeneous. The requirements for a homogeneous computing environment are quite stringent and are frequently not met in networks of workstations, or PCs, even when each computer in the network is the same model. The recent availability of advanced-architecture computers has had a significant impact on all spheres of scientific computation. In the last 50 years, a rapid change of vendors, architectures, technologies and the usage of systems has been seen in the field of scientific computing. Despite all these changes the evolution of performance on a large scale however seems to be a very steady and continuous process [DL06]. So far, two things remain consistent in the realm of computational science: i) there is always a need for more computational power than we have at any given point, and ii) we always want the simplest, yet most complete and easy to use interface to our resources. In recent years, much attention has been given to the area of Grid Computing. The analogy is to that of the electrical power grid. The ultimate goal is that one day we are able to plug any and all of our resources into this Computational Grid to access other resources without worry, as we do our appliances into electrical sockets today.

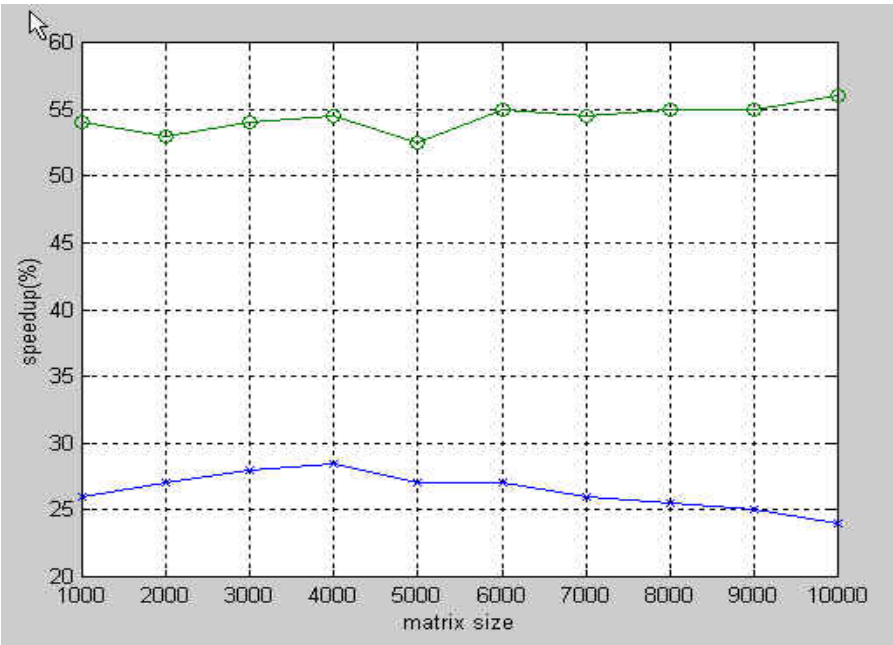
7.2.2 Comparison of Experimental Results

In this experiment, there are six computing servers for computation. We manually change the number of computing nodes enabled with direct communications. Figure 7.1 (a) shows that the average communication speedup for our three applications

grows linearly while the number of computing servers with direct communication enabled increases from 0 to 6.



(a)



(b)

Figure 7.1 Experimental results of three applications in both homogeneous and heterogeneous network.

(a) Speedup for the three applications increases linearly with the increase of the number of computing servers with direct communication enabled from 0 to six ('o' – the matrix chain product; '.' – Genetic crossover; '*' – Image processing using sequential algorithms). (b) Speedups for the matrix chain product. ('*' – homogeneous network; 'o' – heterogeneous network).

If communication links connecting remote computers are much faster than communication links connecting the remote computers and the client computer, the speedup due to elimination of bridge communications will be much higher. In our next experiment, we manually set all bridge communications to perform at the speed of 10 Mbit per second. For direct communications between remote servers, we still use 100 Mbit Ethernet interconnecting network. Figure 7.1 (b) presents the average communication speedup of performing matrix chain computations with direct communications on both homogeneous and heterogeneous networks. The experimental speedup for the heterogeneous network is around 54% when the ratio of eliminated bridge communications is 2/9. Thus, much higher speedup can be achieved in heterogeneous communication networks, which is typical for real-life Grid environments, than in artificially designed homogeneous ones.

7.2.3 Case Study

In this section, it presents the case study of using NI-Connect to enable direct communication in NetSolve in heterogeneous Network. In our next experiment, we use four computing servers for computation. We manually change the number of computing nodes enabled with direct communications. The servers' details are shown in Table 7.1.

The servers which are used to perform computation are considered as a heterogeneous network for following reasons:

- Geographic location. There are two servers (Server 1 & Server 2) in the same local network. Server 3 is located in the same place which is University College Dublin but belongs to a different network. Server 4 is located in University of Tennessee in the United States.

- Operating System. There are two types of known operating system among there servers. Server 1 and Server 3 are running on the Debian linux 3.0. Server 2 is running on the Fedora Core 4.
- Processors and Hardware. These servers include single processor and multi-processors. The performance of the servers varies on either CPU power or memory size.

NetSolve Servers	Software Component
Server 1 (http://csserver.ucd.ie)	Installed
Server 2 (http://cssa.ucd.ie)	Installed
Server 3 (http://pg1cluser01.ucd.ie)	Installed
Server 4 (http://netsolve.cs.utk.edu)	**NOT** Installed

Table 7.1 Installation and specifications of heterogeneous servers

The application which is used to run on the grid programming system is Matrix Chain Product. In Figure 7.2 it shows the bridge communication structure of performing matrix chain product with the size of eight. As we can see, there are 14 communication links totally. By using NI-Connect to enable direct communications in NetSolve, communication links can be reduced to 9. There are totally 5 links are removed by enabling direct communications. In Figure 7.3, it shows the communication structure of enabling direct communication in NetSolve by performing matrix chain product with the size of eight. Table 7.2 shows experimental results of three trails with different sizes of matrix. The results show that the average communication speedup is around 48%. This significantly improves the performance of matrix chain product computation by using our Non-Intrusive and Incremental Approach to enable direct communications in a heterogeneous network.

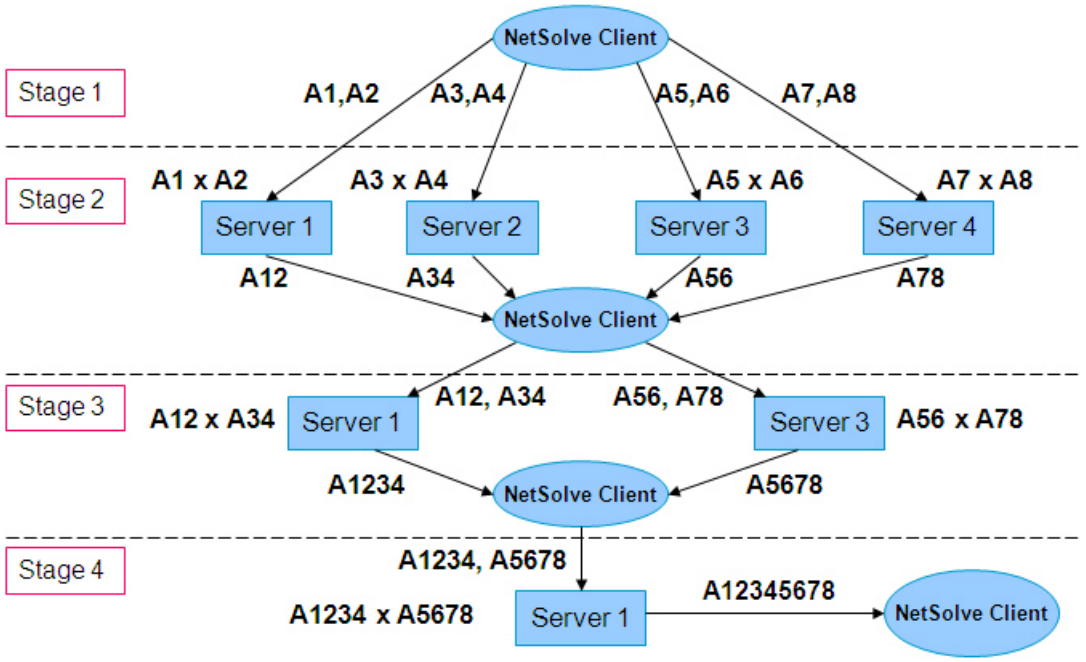


Figure 7.2 Communication structure of performing matrix chain product without NI-Connect enabled.

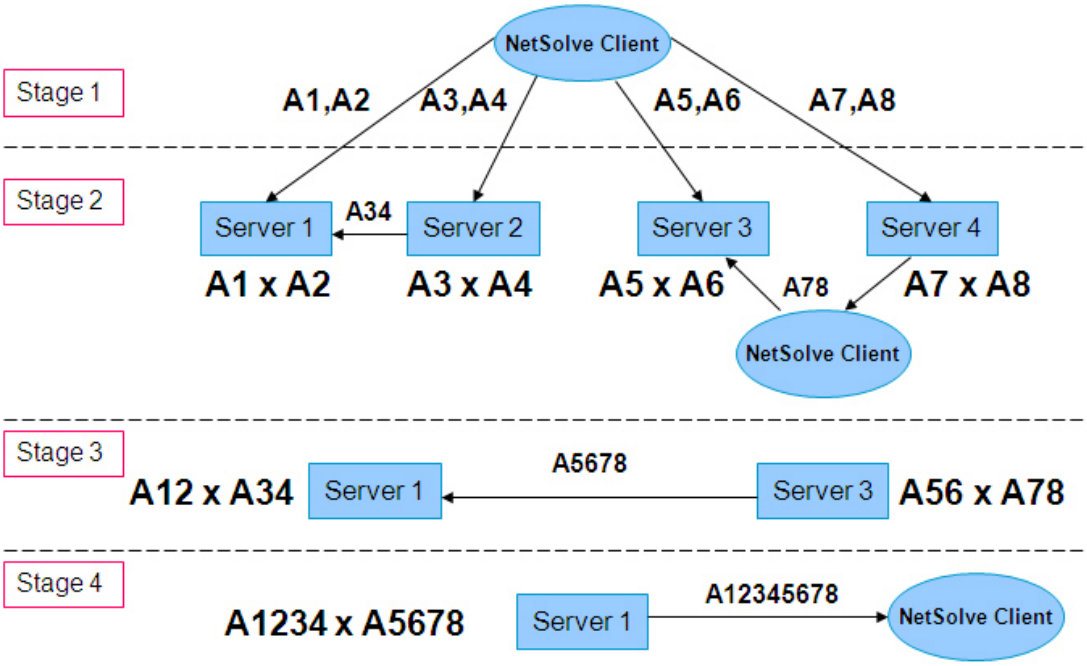


Figure 7.3 Communication structure of performing matrix chain product with NI-Connect enabled.

Picture Size (kb)	Trail 1		Trail 2		Trail 3		Average		Speedup
	B	D	B	D	B	D	B	D	
1000	65	34	64	34	65	35	65	34	48%
2000	130	74	132	70	135	73	133	72	46%
3000	225	107	213	108	233	103	222	105	47%

Table 7.2 B – Bridge communication time (in seconds); D – Direct communication time (in seconds)

7.3 Large-scale Experiments in Grid 5000

This section presents the experiment of using NI-Connect in a larger Grid environment which is Grid 5000 in France.

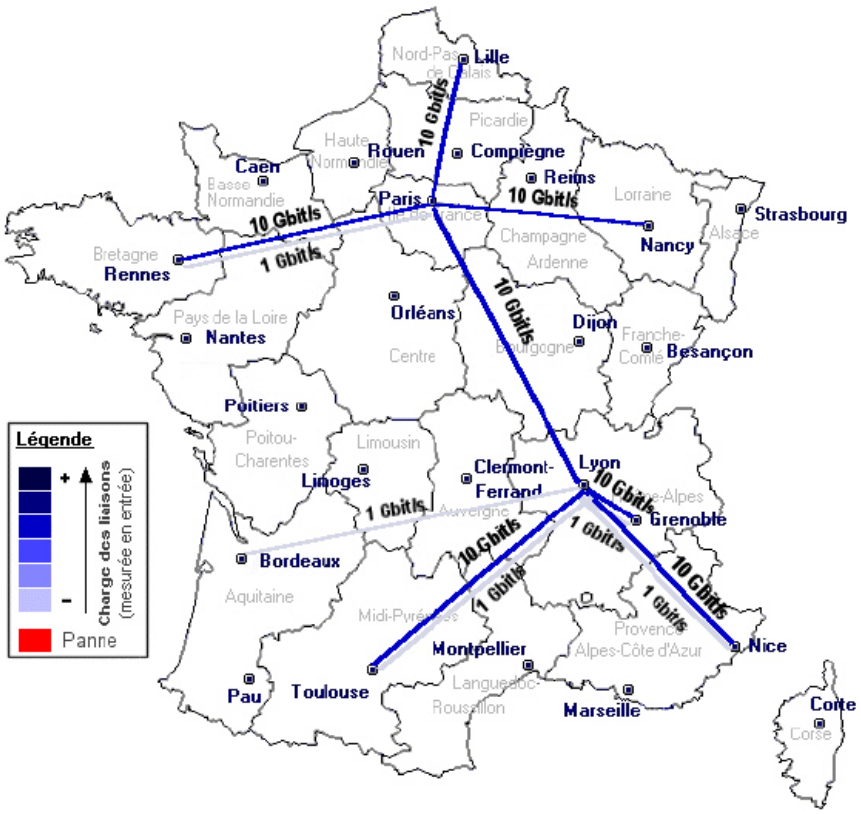


Figure 7.4 Network Overview of Grid 5000.

7.3.1 Grid 5000

The Grid'5000 project of the French ACI GRID incentive is launched in 2003 and the first phase of the project Grid 5000 platform is opened to users in 2005. INRIA is currently focusing on further develop till the year of 2011 [Gri][BCC06].

The basic design concepts of Grid 5000 are as follows:

- Large-Scale and distributed
- 1/3 heterogeneous and 2/3 homogenous hardware resources
- Dedicated network links between sites
- Isolate Grid5000 from the rest of the Internet
- Let packets fly inside Grid5000 without limitation
- Deep reconfiguration mechanism for experiments on all layers of the software
- User has full control of the reserved experimental resources

Grid 500 is a nation-wide grid including 9 sites in France and 1 site in Brazil. They are Bordeaux, Grenoble, Lille, Lyon, Nancy, Orsay, Rennes, Sophia, Toulouse and Porto Alegre (Brazil). In figure 7.4, it shows the network overview of Grid 5000. There are 4792 cores within 9 sites. The families of CPU include AMD Opteron (78%), Inter Xeon EMT 64 (22%), MonoCore (41%), DualCore (46%), QuadCore (13%). All servers in Grid 5000 are bi-processors. Its high performance networks include Myrinet 2000 (222 cards), Myrinet 10G (423 cards) and InfiniBand 10G (161 cards).

Grid 5000's software stack is flexible to use for different type of users:

- Standard tools (e.g. ssh, openldap, ganglia, squid, mediawiki, bugzilla, ...)
- Tools dedicated to Grid'5000, developed and supported by teams loosely related to Grid'5000 technical staff (OAR, taktuk, GRUDU) and now under the maintenance of the technical staff (kadeploy)
- User contributed tools, sometimes hosted on the grid5000-code project on gforge.inria.fr (e.g. oargrid, katapult, kanon)

In Grid 5000, OAR [OAR] is the resource manager used to allocate resources to users for their experiments. The resource manager creates jobs for users, which are basically an execution time on a set of resources. Grid'5000 features 1 OAR resource manager per site. OAR features include:

- Interactive jobs: I want resources now for a bunch of time
- Advanced reservations: I want resources at that date/time for that duration
- Batch jobs: I want my job to run by itself with this script
- Best effort jobs: I use many resources but accept to release them at any time
- Deploy jobs: I want to be granted to deploy a customized OS environment and have full access to the resources
- Powerful resource filtering/matching: I want only quad core machines with more than 8GB of RAM located on the same network equipment

7.3.2 Experimental Results

To perform the experiments on Grid 5000, again we choose NetSolve as targeted Grid Programming System and the particular feature is enabling direct communication between remote tasks. The first goal of this experimental series is to check the feasibility of NI-Connect and to measure several aspects related to the usage of the software component. Measurements include:

- Non-intrusiveness. The original NetSolve system in Grid 5000 does not change and the new features are provided by NI-Connect working on the top of the system. Correspondingly, all applications not requiring new feature which is enabling direct communications will only use the basic original software and will be developed and executed in the same way in the original and modified systems.
- Increment: It means that NI-Connect does not have to be installed on all computers to enable applications with the new features. It can be done incrementally, step by step, and the new feature that enabling direct communications will be enabled in part, with the completeness dependent on

how many nodes in Grid 5000 participating in the execution of the application have been upgraded with the supplementary software component.

The second goal of this experimental series is to evaluate the behaviour of NI-Connect. For this goal we will configure our applications so that the amount of time spent in computation is small in regards to the time spent on communication.

In this experiment we choose to perform numerical algorithm to solve large-scale linear algebra applications. Particularly we study the eigen-problem with large sparse matrices. In order to compare the result of enabling direct communications, we deploy NI-Connect on server computing resources and provide new mechanisms to run the applications.

In table 7.3, it gives the general information of the computing resources in Grid 5000 to run the experiments:

Nodes involved	200
Sites involved	3
Minimum wall time	8h
Batch Mode	No
CPU bound	Yes
Memory bound	Yes
Storage bound	Yes
Interlink bound	Yes
Tools used	NetSolve, NI-Connect

Table 7.3 Computing Resources to perform experiments using NI-Connect

In table 7.4, it gives several trails of experimental results of running eigen-problem on grid 5000 platform. The matrix is the pde1000000 from Matrix market [MM].

	Trail 1	Trail 2	Trail 3
Matrix Size	500000	400000	300000
Computed eigen values	10 20 30 40 50	12 18 24 30 36 42	15 25 35 45 50
Executed tasks	148	120	115
Number of total tasks	4006	5068	3124
Number of worker nodes	45	35	40
Execution time (B)	8550	9006	6542
Execution time (D)	4200	5003	3502

Table 7.4 Experiment results of performing experiments using NI-Connect to enable direct communications on Grid 5000 platform

The experimental results show that total execution time is reduced by around 45%, This significantly improves the performance of solving eigen-problem with large sparse matrices. Thus, by using NI-Connect to enable direct communications in NetSolve on Grid 5000 platform it demonstrate the advantages of the Non-Intrusive and Incremental Approach for the Evolution of Grid Programming System.

Conclusion and Perspectives

Chapter 8

Conclusion

8.1 Context

This thesis advocates a non-intrusive and incremental approach to enable existing Grid programming systems with new features. There are two keywords describing the approach which are Non-intrusiveness and Increment. Non-intrusiveness means that the original system does not change and the new features are provided by a supplementary software component working on the top of the system. Correspondingly, all applications not requiring those new features will only use the basic original software and be developed and executed in the same way both in the original and modified systems. Increment means that the supplementary software component does not have to be installed on all computers to enable applications with the new features. It can be done incrementally, step by step, and the new features will be enabled in part, with the completeness dependent on how many nodes participating in the execution of the application have been upgraded with the supplementary software component.

To demonstrate the approach, in particular we work on a case that presents software component enabling NetSolve applications with direct communications between remote tasks. The target grid programming system is NetSolve, which is positioned as a programming system for high performance distributed computing on global networks based on GridRPC. And the feature which is added into the grid programming system is direct communication between remote tasks. The new feature

deals with the situation when we use the output data of remote tasks. The function is typically used to sent back the completed data to the client upon completion of each remote task even if the data are only needed as input for some other remote tasks, resulting in so-called bride communications when data between remote tasks are sent through the client machine. In our research work, we have developed software component NI-Connect and carried out the experiment to estimate the feasibility and the performance of the method based on the selected case. The software component NI-Connect consists of three parts: Client API & Argument Parser, Server Connector and Job Name Service (JNS). Client API & Argument Parser provides a uniform interface for the client to make remote procedure calls. Despite the modification on the remote side, the wrapper API allows the calls to be made in the same manner. The only difference is that the arguments can be not only variables storing real data but also handlers. Server Connector is responsible for interacting with clients and other Server Connectors to enable direct communications. Job Name Service (JNS) is used for registration of procedure upon its invocation during RPC call. Other procedures may send requests to the JNS to search for registered procedure. JNS is set up on the client side automatically.

After a study on targeted grid programming system and selected feature, we formulate principles and standards for generic implementation of non-intrusive and incremental approach to enable new grid features to general Grid Programming Systems. In this part, generic non-intrusive and incremental structure has been presented and libraries and components are proposed, which can be easily re-used for generic implementation of Non-intrusive and Incremental approach by other programmers.

Lastly, based on the implementation of enabling direction communications between remote servers in NetSolve, we present another three typical scientific applications with different communication structures and demonstrate the performance improvement achieved due to the use the software component for elimination of bridge communications. Those typical scientific NetSolve applications have different communication structures, which are: (i) protein tertiary structure prediction, (ii) image processing using sequential algorithms, and (iii) the matrix chain product. The experimental results show that the performance of NetSolve applications could be significantly and easily improved by using our software component. In this thesis it

also presents experiments that the uses of NI-Connect in heterogeneous networks. It proves that there is much higher speedup can be achieved in heterogeneous communication networks, which are typical for real-life Grid environments, than in artificially designed homogeneous ones. Finally, we prove the feasibility of the approach by running experiment of using NI-Connect in a large-scale Grid environment which is Grid 5000 in France.

8.2 Results and Discussion

We have demonstrated that the evolution of grid programming system can be implemented in a non-intrusive and incremental way. Below, we present more precisely the contributions of this work and other choice of methodology.

8.2.1 Contributions of the Thesis

There are a few contributions we expect the scientific researchers and programmers can benefits from, which are listed as following paragraph.

The Design of Non-Intrusive and Increment Model: In the view of software engineering, the update of the software involves many issues the same as evolution of grid programming system. By using non-intrusive and increment model, it not only improves the process of updating software, but also benefits the software application running on the grid environment. For example, in order to update a web based software on the latest grid-like platform "Cloud" [Ama][AFG09][Wa07]. Software developers can implement a cooperating application within the same cloud that handles the issues of current web services without changing the codes of original deployed class, database tables, etc. And this also can be applied to the web services those belong to different cloud, which the updating can be done incrementally. We expect the uses of the non-intrusive and increment approach in the area of IT industry in the future.

Software component NI-Connect and API: In the area of grid computing system, by using NI-Connect and developed API, scientific researchers and programmers can easily implement the application to execute on the remote servers. This gives the

flexibility in the coding levels for network discovery, task discovery, application mapping and data distribution. The API is already fully developed for the programmers to use or extend. And the software component NI-Connect is currently compatible with NetSolve grid programming system.

Enabling direct communications between remote tasks: For the tasks running in the grid environment, data dependency is an important aspect regarding the performance of the application. The feature we have added to the grid programming system in this thesis significantly improves the performance of grid application by reducing the un-necessary communication links. Unfortunately, the dependency of most grid applications is quite strict, whereas it proves the feature that enabling direct communications is useful. In this thesis there are three typical real-world applications demonstrating that the better performance can be achieved by using NI-Connect. This feature can be applied to other application which have similar communication model to improve the performance of the application in the grid programming system.

8.2.2 Possible Improvements

There are a few improvements can be done for the development of the non-intrusive and incremental approach.

- Currently, the implement of NI-Connect is for NetSolve Grid programming system. In the future, it is can be easily extended to other grid programming system as well, such as GridSolve, NINF-G, DIET grid programming system. By doing so, the features can be added to these grid programming system in a non-intrusive and incremental way.

- In this thesis, we have implemented NI-Connect to add the feature which is enabling direct communications to the existing grid programming system. It is durable to add other feature to the grid programming system in the same way that is non-intrusive and incremental. The other possible features could be added into current grid programming system are like broadcasting, fault tolerance and task monitoring, etc.

- In the case study which is enabling direct communications between remote servers, the NetSolve agent is configured manually by system administrator. One of future development for this project is to implement an algorithm to select a NetSolve agent automatically from one of remote NetSolve servers which have new feature enabled, and investigate how to pick the best agent to execute the remote task.

8.2.3 Towards a Complete Development Frame for the approach

Based on this thesis, we would like to propose a possible work that create framework for the development of Non-intrusive and Incremental approach in the future. In Figure 8.1, it shows the structure of proposed project NI-Manager.

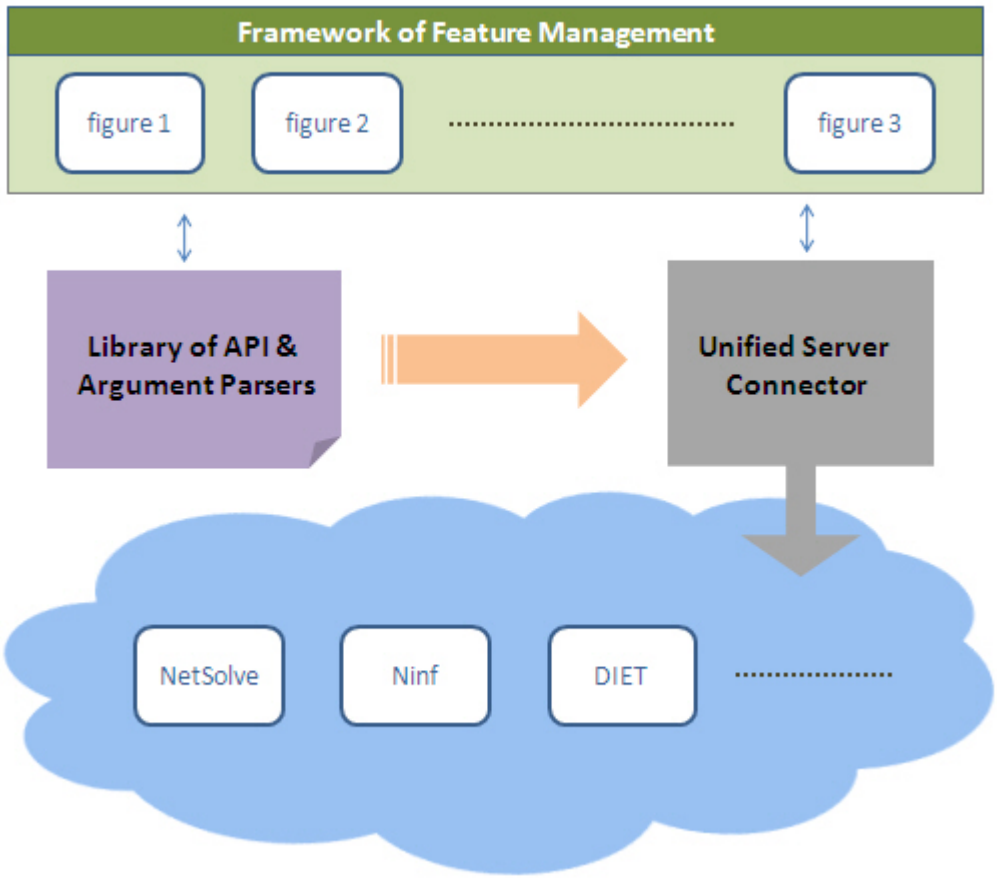


Figure 8.1 Structure of NI-Manager

The project should involve following parts:

- **Library of API & Argument Parsers**: The library is based on the API & Arguments Parsers already developed for use in NetSolve. It can be added incrementally with slightly modification for use of other Grid Programming System such as GridSolve, NINF-G and DIET. The library is sited on the client side and can be easily included by programmers using grid programming system to develop tasks. For the grid application developers, they don't need to change anything to make their applications available to new API.

- **Unified Server Connector**: In NetSolve, Server Connector is responsible for interacting with clients and other Server Connectors to enable direct communications. This is not applied to other grid programming system. Thus a unified Server Connector need to be developed for use of different grid programming system. The unified Server Connector can interact with the clients' request that is the function called within the library of NI-Connect API. It can also communicate with other Unified Server Connector those are set up on different grid programming system.

- **Framework of Feature Management**: In this thesis, one feature we added to the existing grid programming system is enabling direct communications between remote tasks. In the future in order to add more features to the grid programming system in a non-intrusive and incremental way, a framework of feature management would be necessary to implement as a module inside NI-Connect. This allows programmers to easily add, remove and manage the features added to grid programming system through NI-Connect in non-intrusive and incremental way.

Appendixes

Appendix A: User Manual of NI-Connect

Non-intrusive and incremental evolution of Grid programming systems Project, Heterogeneous Computing Laboratory (HCL), School of computer Science and Informatics, University College Dublin

PROGRAMMING ENVIRONMENT

The software component for NetSolve to enable direct communications is installed and run on following linux workstations:
Red Hat Linux 3.3.3-7, gcc version 3.3.3 20040412;
Fedora Core 2.6.11-1.1369_FC4, gcc version 4.0.0 20050519;

REQUIREMENTS

The version of NetSolve which our software component is added and tested on is NetSolve 2.0.

Important:

1. To use our software component, GPG option must be disabled.
2. Direct communications uses port 6234.

DOWNLOAD

The Software Component can be downloaded from http://hcl.ucd.ie. A recent version can also be downloaded from following links:

ORGANIZATION OF SOFTWARE COMPONENT

Source codes:

Client wrapper API: mynetsl.c ; mynetsl.h ;
Server Connector: serverConnector.c; serverConnector.h; serverSetup.c;

INSTALLATION

[On the server side]

To install our software component to a netsolve server, the steps of installation are as follows:

1) Create a directory named “Dc” at the root of NetSolve root directory.
2) Copy files serverConnector.c, serverConnector.h, serverSetup.c to the Dc directory.
3) Build Obj file for Server Connector:

$gcc –Wall –g –c serverSetup.c –o serverSetup.o –I$NETSOLVE_ROOT/include –I$NETSOLVE_ROOT/Dc

4) Link it to the NetSolve Library:

$gcc –g –o serverSetup serverSetup.o –L. –lserverConnector –L$NETSOLVE_ROOT/Dc –lnetsolve –L$NETSOLVE_ROOT/lib/i686_pc_linux_gnu

5) Run the server Connector:

$./serverSetup

[On the client side]

To enable direct communications, the steps of installation of our software component are as follows:

1) Create a directory named “Dc” at the root of NetSolve root directory.

2) Copy files mynetsl.c ; mynetsl.h to the Dc directory.

3) Build lib files for Wrapper API:

$gcc –Wall –g –c –o libmynetsl.o mynetsl.c –I$NETSOLVE_ROOT/include
$ar rcs libmynetsl.a libmynetsl.o

APPLICATION TUTORIAL

This tutorial shows how to enable direct communications by using the software component. The example is performing matrix operations. Files can be downloaded from: http://hcl.ucd.ie/

This includes:

o libmatmul.c - contains the function which should perform the calculation

o matmul.idl - a file which describes how the problem should be included into the NetSolve repository

o myprog.c - an example program which invokes the NetSolve function Instructions:

1. Building the Library

NetSolve provides all functions through statically linked libraries. Execute the following steps on the server to build libmatmul.a

$ gcc -c libmatmul.c
$ ar rc libmatmul.a libmatmul.o

2. Installing the Problem to the NetSolve Repository

To invoke a new function from the NetSolve client you have to add the function to the problems list in a NetSolve server and recompile the server. Perform the following steps to include 'matmul' o create a problem description file with

$ $NETSOLVE_ROOT/bin/$NETSOLVE_ARCH/idltopdf matmul.idl

which generates the file matmul.pdf

- copy the file matmul.pdf to the problems subdirectory in your NetSolve directory

- edit the file "server_config" in your NetSolve directory and add the following line in the "@PROBLEMS:" section:

./problems/matmul.pdf

- to rebuild the server, you have to set an environment variable which points to the directory of the previously created library (required by the problems definition file); depending on you shell execute

$ export NSMATMUL_LIB=/path/to/libmatmul

or

$ setenv NSMATMUL_LIB /path/to/libmatmul

- Afterwards rebuild the server with the command in your NetSolve directory

$ make server

3. Invoking the Function via NetSolve

The file myprog.c demonstrates how to invoke the function calc via NetSolve in C. the original NetSolve calling is like:

Info = netsl("matmul()", metA, metB, metC);
Info = netsl("matmul()", metC, metD, metE);

To enable direct communication, our wrapper API mynetsl() and handlers are used:

Info = mynetsl("matmul()", matA, matB, hdlC);
Info = mynetsl("matmul()", hdlC, matD, matE);

To build an application program, the command of using our library is as follows:

$gcc -Wall –g –c myprog.c myprog.o –I$NETSOLVE_ROOT/include –I$NETSOLVE_ROOT/Dc

$gcc –g –o myprog myprog.o –L. –lmynetsl –L$NETSOLVE_ROOT/Dc –lnetsolve –L$NETSOLVE_ROOT/lib/i686_pc_linux_gnu

Invoking myprog uses 2 arguments:

- *matrix size: integer with the dimension of the used matrix*
- *mode: 1 - blocking call , 2 - non-blocking call*

Appendix B: Core function of NI-Connect

mynetsl() : C Interface to the NetSolve

```
1   int mynetsl(char *nickname,...)
2   {
3   va_list argptr;
4   char * buf;
5   /* Getting the problem's nickname */
6   va_start(argptr,nickname);
7   buf = strdup(nickname);
8   return mynetslX(buf, argptr, NS_CALL_FROM_C, NS_BLOCK,
    NS_NOASSIGNMENT);
9   }

10  int mynetslX (char *buf, va_list arglist,int language, int blocking,
    int assignment)
11  {
12  return mynetslX_common (buf, 1, arglist, NULL, language, blocking,
    assignment);
13  }

14  int mynetslX_common (char *pname, int mode_valist, va_list arglist,
    void **arglist_arr, int language, int blocking,
    int assignment)
15  {
16  int i, len;
17  int request_id;
18  int status;
19  void **tmp_output_ptr;
20  int elapsed;
21  NS_Node *act_node = 0;
22  NS_ProblemDesc *pd;
23  char *nickname, *serverhostname;
24  char *buf;

25  if(!pname) {
26  ns_errno = NetSolveBadProblemName;
27  return ns_errno;
28  }

29  buf = strdup(pname);

30  #if defined(DEBUG)

31  ns_printinfo();

32  fprintf(stderr, "Processing a Client Request: language = %d,
    ""blocking = %d, assignment = %d\n", language, blocking, assignment);

33  #endif
34  printf("%s", "finished.00\n");
35  /* find a slot for the request */
36  request_id = 0;
37  while(requests[request_id] != NULL)
38  {
```

```
39  request_id++;
40  if (request_id == NB_MAX_REQUESTS)
41  {
    ns_errno = NetSolveTooManyPendingRequests;
    free(buf);
    return ns_errno;
42  }
43  }

44  if(!proxy_pid){
45  if(netslinit(NULL) < 0){
    free(buf);
    return ns_errno;
46  }
47  }

48  #if defined(DEBUG)
49  ns_printinfo();
50  fprintf(stderr, "Setting Default Client Major ...\n");
51  #endif
52  /*== Initializing the major ==*/
53  if(language == NS_CALL_FROM_C)
54  setMajorDefault("Row");
55  else
56  setMajorDefault("Col");

57  #if defined(DEBUG)
58  ns_printinfo();
59  fprintf(stderr, "Stripping `(` and `)` from problemname %s ...\n",
     buf);
60  #endif
61  /*== Stripping off the parenthesis ==*/
62  len = strlen(buf);
63  if ((buf[len-2] != '(' ) ||
    (buf[len-1] != ')' ) )
64  {
65  #if defined(DEBUG)
66  ns_printinfo();
67  fprintf(stderr, "There is a bad problem name...\n"); fflush(stderr);
68  #endif
69  free(buf);
70  ns_errno = NetSolveBadProblemName;
71  return ns_errno;
72  }
73  buf[len-2] = '\0';

74  #if defined(DEBUG)
75  ns_printinfo();
76  fprintf(stderr, "Extracting the problem name [and
    serverhostname] ...\n");
77  fflush(stderr);
78  #endif

79  if(assignment)
80  {
81  /* Extracting the nickname, serverhostname */
82  serverhostname = strtok(buf,":");
83  nickname = strtok(NULL,"\0");

84  if ((serverhostname == NULL)||
    (nickname == NULL))
```

```
85  {
    free(buf);
    return NetSolveBadProblemName;
86  }
87  }
88  else{
89  nickname = buf;
90  serverhostname = NULL;
91  }


92  /*== Getting the problem descriptor ==*/
93  status = netsolveInfo(nickname, &pd);
94  if (status == -1) {
95  free(buf);
96  return ns_errno;
97  }


98  /* general variables */
99  int n;

100 /* variables for blocking calls */
101 //int info;
102 double alpha, beta;
103 double *mat;
104 double *vec;

105 n = 100;
106 srand( time( NULL ) );

107 /* --blocking calls example--------- */
108 /* creating matrix & vector */
109 mat = calloc(n*n, sizeof(double));
110 for (i=0; i<n*n; i++)
      mat[i] = (double) (10.0*rand()/RAND_MAX);
111 vec = calloc(n, sizeof(double));
112 for (i=0; i<n; i++)
      vec[i] = (double) (10.0*rand()/RAND_MAX);
113 alpha = 1.0;
114 beta = 0.0;

115 /* blocking call to NetSolve test netsl in mynetsl: done */

116 /* write message to stdout */
117 printf("%s", "finished\n");


118 /* gets the server name of an appropriate server */
119 /* make it assignment */
120 NS_RequestDesc *rd_getServer;
121 rd_getServer = CP_sendJobRequest(pd,0,0,-666);
122 serverhostname = strtok(rd_getServer->hostname, "\0");
123 //fprintf(stdout, rd_getServer->IPaddr);
124 //free(rd_getServer);
125 assignment = 1;


126 #if defined(DEBUG)
127 ns_printinfo();
128 fprintf(stderr, "Problem name: %s\n", nickname);
```

```
129 if(assignment){
130 ns_printinfo();
131 fprintf(stderr, "Assigned Server: %s\n", serverhostname);
132 }
133 #endif

134 #if defined(WIN32)
135 /*== Initialize WinSock ==*/
136 if ((!WinSockInit()))
137 {
138 ns_errno = NetSolveNetworkError;
139 free(buf);
140 return ns_errno;
141 }
142 #endif /* WIN32 */

143 #if defined(DEBUG)
144 ns_printinfo();
145 fprintf (stderr, "Getting Problem Descriptor via netsolveInfo()\n");
146 #endif
```

---

serverConnector() : C Interface to the NetSolve

```
1   typedef struct {
2   NS_ServerDesc *my_self;         /* The server's own descriptor */
3   NS_LinkedList *problems;    /* The problem linked list    */
4   NS_LinkedList *agents;          /* The agent linked list      */
5   NS_AgentDesc   *master;         /* The master descriptor      */
6   char *scratch_path;         /* The scratch area          */
7   char *Condor_path;          /* Possible Condor path       */
8   int Condor_nb;          /* Possible # of Condor procs  */
9   char *MPI_path;         /* Possible mpirun path       */
10  int nb_MPInodes;        /* Possible Number of MPI nodes*/
11  char* MPInodefile;          /* Possible MPI node file     */
12  int workload_manager_pid;       /* PID of the workload manager */
13  #ifdef HBM
14  int hbmlm_pid;          /* PID of the HBM hbmlm       */
15  #endif
16  char *netsolve_root_path;   /* $NETSOLVE_ROOT         */
17  NS_Statistics *statistics;          /* statistics    */
18  NS_ScaLAPACKGlobal Scalapack_global;  /* globals for ScaLAPACK    */
19  int nb_problems;            /* number of problems    */
20  int sock;               /* Listening socket descriptor */
21  char *log_file;     /* file for message logging */
22  char *config_file;     /* file for message logging */
23  char *ip_address;       /* IP-Address of the server   */
24  #ifdef KERBEROS5
25  krb5_principal server;
26  krb5_keytab keytab;
27  krb5_context context;
28  char *users_file;
29  int require_auth;     /* true if authentication required */
30  #endif
31  } NS_myGlobal;
```

```
32  NS_myGlobal myglobal;
33  int myinitMySelf();
34  int myserver_init(int argc,char **argv);


35  /*
36  Main routine. Does nothing but calling the initialization routine
37  and waiting for a network connection
38  */


39  int main(int argc,char **argv)
40  {

41  // test
42  /* general variables */
43  int n,i;

44  /* variables for blocking calls */
45  //int info;
46  double alpha, beta;
47  double *mat;
48  double *vec;

49  n = 100;
50  int info;
51  srand( time( NULL ) );

52  /* --blocking calls example--------- */
53  /* creating matrix & vector */
54  mat = calloc(n*n, sizeof(double));
55  for (i=0; i<n*n; i++)
56  mat[i] = (double) (10.0*rand()/RAND_MAX);
57  vec = calloc(n, sizeof(double));
58  for (i=0; i<n; i++)
59  vec[i] = (double) (10.0*rand()/RAND_MAX);
60  alpha = 1.0;
61  beta = 0.0;

62  /* blocking call to NetSolve test netsl in mynetsl: done */
63  info = netsl("matmul()",mat,vec,n);

64  /* error handling for blocking call */
65  if (info <0) {
        netslerr(info);
        exit(0);
66  }


67  int sockfd, newsockfd, portno, clilen;
68  char buffer[256];
69  struct sockaddr_in serv_addr, cli_addr;
70  //int n;

71  sockfd = socket(AF_INET, SOCK_STREAM, 0);
72  if (sockfd < 0)
        error("ERROR opening socket");
73  bzero((char *) &serv_addr, sizeof(serv_addr));
74  portno = atoi(argv[1]);
75  serv_addr.sin_family = AF_INET;
76  serv_addr.sin_addr.s_addr = INADDR_ANY;
```

```
77  serv_addr.sin_port = htons(portno);
78  printf("%s", "finished.03\n");
79  if (bind(sockfd, (struct sockaddr *) &serv_addr,
                sizeof(serv_addr)) < 0)
                error("ERROR on binding");
80  printf("%s", "finished.04\n");
81  listen(sockfd,5);
82  printf("%s", "finished.05\n");
83  clilen = sizeof(cli_addr);
84  newsockfd = accept(sockfd,
                (struct sockaddr *) &cli_addr,
                &clilen);
85  printf("%s", "finished.06\n");
86  if (newsockfd < 0)
         error("ERROR on accept");
87  bzero(buffer,256);
88  n = read(newsockfd,buffer,255);
89  if (n < 0) error("ERROR reading from socket");
90  printf("Here is the message: %s\n",buffer);
91  n = write(newsockfd,"I got your message",18);
92  if (n < 0) error("ERROR writing to socket");

93  NS_Communicator *comm;
94  int tag;

95  comm = acceptTransaction(newsockfd);

96  if (recvInt(comm,&tag) == -1)
97  {
         netsolvePerror("");
         endTransaction(comm);
         return -1;
98  }

99  if (sendInt(comm, &tag) == -1)
100 {
101 endTransaction(comm);
         printf("%s", "finished3333\n");
102 return -1;
103 }

                printf("%s", "test1\n");

104 NS_ProblemDesc *pd, *pd_new;
105 pd = recvProblemDesc(comm);
106 //pd_new->input_objects = (void
      **)calloc(pd->nb_input_objects,sizeof(void*));
107 pd_new->output_objects = (void
      **)calloc(pd->nb_output_objects,sizeof(void*));
108 pd->output_objects = pd_new->output_objects;
109 //pd->input_objects = pd_new->input_objects;

110 if (pd == NULL) {
111 netsolvePerror("");
112 return;
113 }
114 // comm send object
115 /* receiveing the objects */
116 NS_Object **list = pd->input_objects;
117 int nb = pd->nb_input_objects;
118 char buffer2[256];
```

```
119 //int i;

120 for (i=0;i<nb;i++)
121 {
122 /* do initializations */
123 switch(list[i]->object_type)
124 {
125 case NETSOLVE_MATRIX:
        list[i]->attributes.matrix_attributes.m = -1;
        list[i]->attributes.matrix_attributes.n = -1;
        list[i]->attributes.matrix_attributes.l = -1;
        list[i]->attributes.matrix_attributes.major = -1;
        list[i]->attributes.matrix_attributes.ptr = NULL;
        list[i]->attributes.matrix_attributes.d = NULL;
        break;
126 case NETSOLVE_SPARSEMATRIX:
        list[i]->attributes.sparsematrix_attributes.m = -1;
        list[i]->attributes.sparsematrix_attributes.n = -1;
        list[i]->attributes.sparsematrix_attributes.major = -1;
        list[i]->attributes.sparsematrix_attributes.f = -1;
        list[i]->attributes.sparsematrix_attributes.rc_ptr = NULL;
        list[i]->attributes.sparsematrix_attributes.rc_index = NULL;
        list[i]->attributes.sparsematrix_attributes.ptr = NULL;
        list[i]->attributes.sparsematrix_attributes.d = NULL;
        break;
127 case NETSOLVE_VECTOR:
        list[i]->attributes.vector_attributes.m = -1;
        list[i]->attributes.vector_attributes.ptr = NULL;
        break;
128 case NETSOLVE_SCALAR:
        list[i]->attributes.scalar_attributes.ptr = NULL;
        break;
129 case NETSOLVE_STRING:
        list[i]->attributes.string_attributes.ptr = NULL;
        break;
130 case NETSOLVE_STRINGLIST:
        list[i]->attributes.stringlist_attributes.strings = NULL;
        list[i]->attributes.stringlist_attributes.m = -1;
        break;
131 case NETSOLVE_FILE: /* Create a file name */
        sprintf(buffer2,"fileinput%d",i);
        list[i]->attributes.file_attributes.filename =
        strdup(buffer2);
        break;
132 case NETSOLVE_PACKEDFILES: /* Create a file name */
        sprintf(buffer2,"packedfileinput%d",i);
        list[i]->attributes.packedfiles_attributes.defaultprefix =
        strdup(buffer2);
        list[i]->attributes.packedfiles_attributes.filenames = NULL;
        list[i]->attributes.packedfiles_attributes.m = -1;
        break;
133 case NETSOLVE_UPF: /* Create a file name */
134 #if (defined(VIEW) || defined(DEBUG))
        logIt("There should not be any UPF here !!!\n");
135 #endif
        return -1;
        break;
136 default: /* Nothing to so */
        break;
137 }
```

```
138 /* receive the object from the net */
139 if (recvObject(comm,list[i]) == -1)
140 {
141 netsolvePerror("recvObject()");
142 return -1;
143 }
144 }


145 int blocking = 1;
146 int assignment = 0;
147 char *serverhostname = NULL;
148 int request_id = 0;
149 int elapsed, status;
                printf("%s", pd->name);
                printf("%d", pd->input_objects[0][3]);
                printf("%s", "test4\n");

150 status = submit_problem(blocking, assignment, serverhostname, pd,
                        a.  pd->input_objects, pd->output_objects,
                              &elapsed, request_id);


151 printf("%s", "finished.0\n");
152 return 1;

153 NS_Socket_type sock;

154 /* Initialize the server */

155 listen(myglobal.sock,MAX_CONNECTIONS);

156 printf("%s", "finished.01\n");

157 while(1)
158 {
159 //printf("%s", "finished.02\n");
160 if ((sock = myacceptConnection(myglobal.sock)) == -1)
161 {
162 //fprintf("%d",sock);
163 //printf("%s", "finished.11\n");
164 continue;
165 }
166 printf("%s", "finished.03\n");
167 //processMessage(sock);
168 fflush(stderr);
169 fflush(stdout);
170 }
171 }

172 /*
173 acceptConnection()
174 */
175 NS_Socket_type myacceptConnection(NS_Socket_type listening_socket)
176 {
177 struct sockaddr_in addr;     /* INET socket address */
178 int addrlen;    /* address length     */
179 NS_Socket_type sock;

180 addrlen = sizeof(addr);
    printf("%s", "accpeptconnection.1\n");
```

```
181 sock = accept(listening_socket,
                 (struct sockaddr *)&addr,&addrlen);
182 printf("%s", "acceptconnection.02\n");
183 while(sock < 0 && errno == EINTR){

184 sock = accept(listening_socket,
                 (struct sockaddr *)&addr,&addrlen);
185 }
186 if (sock >= 0)  printf("%s", "finished.05\n");
187 return sock;
188 }
```

Bibliography

A

[AAB02] D. Arnold, S. Agrawal, S. Blackford, J. Dongarra, M. Miller, K. Seymour, K. Sagi, Z. Shi, S.Vadhiyar: "Users' Guide to NetSolve V1.4.1". Innovative Computing Dept. Technical Report ICL-UT-02-05, University of Tennessee, Knoxville, TN (2002)

[Acc86] M. Accetta, R. Baron, D. Golub, R. Rashid, A. Tevanian, and M. Young. ``Mach: A New Kernel Foundation for UNIX Development," roceedings of the Summer 1986 USENIX Conference, June 1986.

[ACD02] D. Arnold, H. Casanova, J. Dongarra. "Innovation of the NetSolve Grid Computing System". Concurrency: Practice and Experience, 14(13-15):1457-1479, 2002.

[AFG09] M. Armbrust, A. Fox, R. Griffith, Anthony D. Joseph, Randy H. Katz. "Above the Clouds: A Berkeley View of Cloud Computing". UCB/EECS-2009-28, Publisher: EECS Department, University of California, Berkeley, Pages: 07-013

[Ama] Amazon Web Services - Simple Storage Service (S3). http://aws.amazon.com/

[Apg] Asia Pacific Grid: http://www.apgrid.org/

B

[BAB02] M. Beck, D. Arnold, A. Bassi, F. Berman, H. Casanova, J. Dongarra, T. Moore, G. Obertelli, J. Plank, M. Swany, S. Vadhiyar, R. Wolski,

"Middleware for the use of storage in communication. Parallel Computing", Volume 28, Issue 12, December 2002

[BB05] L. Baduel and F. Baude. "Effective and Efficient Communication in Grid Computing with an Extension of ProActive Groups", International Parallel and Distributed Processing Symposium (April 2005).

[BCC06] R. Bolze, F. Cappello, E. Caron, M. Daydé, F. Desprez, E. Jeannot, Y. Jégou, S. Lanteri, J. Leduc, N. Melab, G. Mornet, R. Namyst, P. Primet, B. Quetier, O. Richard, E.-G. Talbi, I. Touche. "Grid'5000: A Large Scale And Highly Reconfigurable Experimental Grid Testbed", in: International Journal of High Performance Computing Applications, November 2006, vol. 20, no 4, p. 481–494.

[BDG09] T. Brady, J. Dongarra, M. Guidolin, A. Lastovetsky, and K. Seymour, "SmartGridRPC: The new RPC model for high performance Grid computing", University College Dublin, pp. 55, 10/2009

[BG08] T. Brady, M. Guidolin, and A. Lastovetsky, "Experiments with SmartGridSolve: Achieving Higher Performance by Improving the GridRPC Model", The 9th IEEE/ACM International Conference on Grid Computing, Tsukuba, Japan, Sep 29 - Oct 1, 2008

[BK92] V. Bala and S. Kipnis. Process group: "a mechanism for the coordination of and communication among processes in the Venus collective communication library". Technical report, IBM T. J. Watson Research Center, October 1992.

[BKL06] T. Brady, E. Konstantinov, A. Lastovetsky, "SmartNetSolve: High Level Programming System for High Performance Grid Computing", Proceedings of the 20th International Parallel and Distributed Processing Symposium (IPDPS 2006), Rhodes Island, Greece, IEEE Computer Society, 25-29 April 2006

[BKR92] V. Bala, S. Kipnis, L. Rudolph and Marc Snir. “Designing efficient, scalable, and portable collective communication libraries”. Technical report, IBM T. J. Watson Research Center, October 1992

[BN84] A. Birrell and B. Nelson, “Implementing remote procedure calls,” ACM Transactions on Computer Systems (TOCS), vol. 2, no. 1, pp. 39–59, 1984

[BSD] BSD License. http://www.linfo.org/bsdlicense.html

[Buy02] Rajkumar Buyya, "Economic-based Distributed Resource Management and Scheduling for Grid Computing", PhD Thesis, Monash University, Melbourne, Australia, April 12, 2002

[BW02] L. Baumstark, L. Wills, “Exposing data-level parallelism in sequential image processing algorithms”, Proc. Of the 9th Working Conference on Reverse Engineering, pp. 245- 54, 2002.

C

[CAS95] L. Cordella, A. d'Acierno, C. De Stefano. “Mapping schemes for sequential image processing algorithms”. Proc. of CAMP'95, pp. 184 - 189, 1995.

[CD97] H.Casanova, J.Dongarra. “NetSolve: A Network Server for Solving Computational Science Problems”. The International Journal of Supercomputer Applications and High Performance Computing, 11(3):212-223, 1997.

[CD06] E. Caron and F. Desprez. “DIET: A Scalable Toolbox to Build Network Enabled Servers on the Grid”, International Journal of High Performance Computing Applications, 20(3):335-352, 2006

[CDK01] R. Chandra, L. Dagum, D. Kohr, D. Maydan, J. McDonald, R. Menon, Parallel programming in OpenMP, Morgan Kaufmann Publishers Inc., San Francisco, CA, 2001

[Com03] Computer Science and Telecommunications Board (CSTB). The Book: "The future of supercomputing an interim report". 2003.

D

[DJ04] F. Desprez, E. Jeannot. "Improving the gridrpc model with data persistence and redistribution", In: International Symposium on Parallel and Distributed Computing in association with HeteroPar (2004)

[DL06] J. Dongarra and A. Lastovetsky, "An Overview of Heterogeneous High Performance and Grid Computing", Engineering the Grid: Status and Perspective: American Scientific Publishers, February 2006

[DLP03] J. DONGARRA, P. LUSZCZEK, A. PETITET, The LINPACK benchmark: Past, present, and future. Concurrency and Computation: Practice and Experience 15, 1-18. 2003

[DM98] L. Dagum, R. Menon, and S. Inc, "OpenMP: an industry standard API for shared-memory programming," IEEE Computational Science & Engineering, vol. 5, no. 1, pp. 46–55, 1998

E

[EDG] The European DataGrid Project. http://eu-datagrid.web.cern.ch/eu-datagrid/

[EM10] EuroMPI 2010, The 17th EuroMPI conference. http://www.eurompi2010.org

[ESG] Earth System Grid (ESG). http://www.earthsystemgrid.org

F

[FK04] I. Foster, C. Kesselman. The Book: “The Grid 2: Blueprint for a New Computing Infrastructure”. 2004.

[FK97] I. Foster and C. Kesselman, “Globus: A metacomputing infrastructure toolkit,” International Journal of High Performance Computing Applications, vol. 11, no. 2, p. 115, 1997

[For] FORTRAN: a general-purpose, procedural, imperative programming language. http://www.fortran.com/

[FTF01] J. Frey, T. Tannenbaum, I. Foster, M. Livny, and S. Tuecke, "Condor-G: A Computation Management Agent for Multi-Institutional Grids", Proceedings of the Tenth IEEE Symposium on High Performance Distributed Computing (HPDC10) San Francisco, California, August 7-9, 2001

G

[GCC] GCC, the GNU Compiler Collection. http://gcc.gnu.org/

[GL] W. Gropp and E. Lusk, Message Passing Interface (MPI) http://www.mcs.anl.gov/mpi/

[Gli] gLite: middleware for grid computing. http://glite.web.cern.ch/glite/

[GNO] GNOME: The Free Software Desktop Project. http://www.gnome.org/

[GNU] GNU General Public License. http://www.gnu.org

[Gra05] L. Grandinetti, “Grid Computing: The New Frontier of High Performance Computing”, 2005.

[Gri] Grid5000: a large scale nation wide infrastructure for Grid research. https://www.grid5000.fr

H

[HCL] Heterogeneous Computing Laboratory, http://hcl.ucd.ie

[HMO00] T. Hiroyasu, M. Miki, M. Ogura, "Parallel Simulated Annealing using Genetic Crossover", Proc. of the IASTED Int'l Conference on Parallel and Distributed Computing Systems, pp.145-150, 2000.

I

[Ian95] F, Ian. Designing and Building Parallel Programs. Addison-Wesley ISBN 0201575949, chapter 8 Message Passing Interface, 1995

[ICL] Innovative Computing Laboratory, University of Tennessee. NetSolve. http://icl.cs.utk.edu/netsolve/

[ICLG] Innovative Computing Laboratory, University of Tennessee. GridSolve. http://icl.cs.utk.edu/gridsolve/

[IDL] Interface Definition Language: http://www.eecs.wsu.edu/~mckinnon/./idl-adm.pdf

[INR] INRIA : The french national institute for research in computer science and control. DIET. http://graal.ens-lyon.fr/~diet/

[IPG] NASA Information Power Grid (IPG) Infrastructure. http://www.ipg.nasa.gov

[IPL98] Image Processing Library 98: http://www.mip.sdu.dk/ipl98/

K

[KR] K. Kennedy of Rice University. High Performance FORTRAN. http://hpff.rice.edu/

[KTF03] N. T. Karonis, B. Toonen, I. Foster. "MPICH-G2: A Grid-enabled implementation of the Message Passing Interface". Department of Computer Science, Northern Illinois University, DeKalb, IL 60115, USA, 2003

L

[LHP04] E. Laure, F. Hemmer, F. Prelz, S. Beco, S. Fisher, M. Livny, L. Guy, M. Barroso, P. Buncic, Z. Kunszt, A. Di Meglio, A. Aimar, A. Edlund, D. Groep, F. Pacini, M. Sgaravatto, O. Mulmo. "Middleware for the next generation Grid infrastructure" Computing in High Energy Physics and Nuclear Physics 2004, Interlaken, Switzerland, 27 Sep - 1 Oct 2004, pp.826

[Lk01] K. Li, "Fast and Scalable Parallel Algorithms for Matrix Chain Product and Matrix Powers on Distributed Memory Systems," Procs. of the 15th International Parallel and Distributed Processing Symposium (IPDPS'01), 2001

[Lk02] K. Li, "Fast and Scalable Parallel Algorithms for Matrix Chain Product and Matrix Powers on Reconfigurable Pipelined Optical Buses", Journal of Information Science and Engineering 18, 713-727, 2002

[Lov93] D. B. Loveman, "High Performance Fortran," IEEE Parallel & Distributed Technology: Systems & Applications, vol. 1, no. 1, pp. 25-42, February, 1993.

[LZZ06] A. Lastovetsky, X. Zuo, P. Zhao. "A Non-intrusive and Incremental Approach to Enabling Direct Communications in RPC-Based Grid

Programming Systems". Proc, of the 2006 Int'l Conference on Computational Science. pp 1008-1011.

M

[Mat] Matlab. http://www.mathworks.com/products/matlab/

[Mic] Microsoft Corporation. http://www.microsoft.com

[ML09] Guidolin, M., and A. Lastovetsky, "Grid-Enabled Hydropad: a Scientific Application for Benchmarking GridRPC-Based Programming Systems", The 23rd IEEE International Parallel and Distributed Processing Symposium, Rome, Italy, May 25 - 29, 2009

[MM] Matrix Market. http://math.nist.gov/MatrixMarket/

[MPI3] MPI 3.0 Standardization Effort. http://meetings.mpi-forum.org/MPI_3.0_main_page.php

[MPICH] MPICH-A Portable Implementation of MPI http://www.mcs.anl.gov/research/projects/mpich2/

[Mys] MySQL Database: http://www.mysql.com/

N

[Ncu92] nCUBE Corporation. nCUBE 2 Programmers Guide. December, 1990.

[News] NetSolve/GridSolve news. http://icl.cs.utk.edu/netsolve/news/index.html

O

[OAR] Tutorial for Grid 5000 and OAR http://mescal.imag.fr/membres/yiannis.georgiou/grid5000_tutorial.html

[Oct] Octave: a high-level language, primarily intended for numerical computations. http://www.gnu.org/software/octave/

[OMP] OpenMP: API specification for parallel programming. http://openmp.org

[Osi] Open Source Initiative. http://www.opensource.org/

P

[PCP92] Parasoft Corporation, Pasadena. CA. Express User's Guide. 1992.

[Pie88] Paul Pierce. The NX/2 operating system. In Proceedings of the Third Conference on Hypercube Concurrent Computers and Applications. pages 384-390. ACM Press, 1988.

R

[RW04] J. Des Rivières, J. Wiegand, IBM Software Group, Ottawa, Ontario. “Eclipse: a platform for integrating development tools”.IBM Systems Journal,Volume 43, Issue 2, Pages: 371-383,2004, ISSN:0018-8670

S

[SL03] J. Squyres and A. Lumsdaine, “A Component Architecture for LAM/MPI”, Springer Berlin / Heidelberg, Volume 2840/2003

[Slc] Simple Linear Combination Filtering: http://www.adires.com/05/Project/LinCom.shtml

[Sm77] M. Schwartz, “Computer Communications Network Design and Analysis”, Prentice Hall PTR, Upper Saddle River, NJ, 1977

[SNM02] K. Seymour, H. Nakada, S. Matsuoka, J. Dongarra, C. Lee, and H. Casanova, “Overview of GridRPC: A remote procedure call api for grid computing,” Lecture notes in computer science, pp. 274–278, 2002.

[SOW95] M. Snir, S. Otto, D. Walker, J. Dongarra, and S. Huss-Lederman, “MPI: The complete reference”. MIT Press Cambridge, MA, USA, 1995.

[Sub] Subversion: An open-source revision control system. http://subversion.apache.org

T

[TAH04] Y. Tanimura, K. Aoi, T. Hiroyasu, M. Miki, Y. Okamamoto and J. Dongarra. “Implementation of Protein Tertiary Structure Prediction System with NetSolve,” Proc, of the 7th Int’l Conference on High Performance Computing and Grid in Asia Pacific Region, pp. 320–327, 2004

[TNS03] Y. Tanaka, H. Nakada, S. Sekiguchi, T. Suzumura, S. Matsuoka, “Ninf-G: A reference implementation of RPC-based programming middleware for Grid computing”. Journal of Grid Computing, 1(1): 41-51, 2003.

[Top500] Top 500 Super Computers. http://www.top500.org/

U

[Uni] UNICORE: Uniform Interface to Computing Resources. http://www.unicore.eu/

W

[Wa07] A. Weiss, “Computing in the clouds”, Net Worker, v.11 n.4, p.16-25, December 2007

[Wol] Wolfram Research of Champaign, Illinois. Mathematica. http://www.wolfram.com/products/mathematica

[WRS98] J. Worley, T. Robey, K. Shuldberg. “Image Processing Operations”, Khoral Research, Inc., April 28, 1998.

Z

[ZL07] X. Zuo, A. Lastovetsky, "Experiments with a Software Component Enabling NetSolve with Direct Communications in a Non-Intrusive and Incremental Way", Proceedings of the 21st International Parallel and Distributed Processing Symposium (IPDPS 2007), Long Beach, California, USA, IEEE Computer Society, 26-30 March 2007