Année 2010 N° d'ordre : 40387

Université Lille1 Sciences et Technologies
École Doctorale Sciences Pour l'Ingénieur
Laboratoire d'Informatique Fondamentale de Lille (UMR CNRS 8022)

Expériences et paradigmes de programmation pour calcul scientifique à grande échelle sur les grilles, les grilles de PC et les nuages informatique privés

Thèse

pour obtenir le grade de

Docteur de l'Université Lille1 Sciences et Technologies

Informatique

Présentée et soutenue publiquement par

Ling SHANG

Le 06 Décembre 2010

Membres du jury :

M. CODOGNET Philippe, Pr - Université Paris 6	Président
M. SONG Fangmin, Pr – Nanjing University	Rapporteur
Mme. EMAD Nahid, Pr –Université Versailles	Rapporteur
M. QIN Xiaolin, Pr–Nanjing University of Aeronautics and Astronautics	Examinateur
M. WANG Zhijian, Pr – Hohai University	Examinateur
M. PETITON Serge, Pr – Université Lille 1	Directeur de thèse

Expériences et paradigmes de programmation pour calcul scientifique à grande échelle sur les grilles, les grilles de PC et les nuages informatique privés

par Ling Shang

These presentee a

L'Universite des Sciences et Technologies de Lille

Pour obtenir le titre de

Docteur en Informatique

These soutenue le 6 december 2010 devant la commission d'examen

Experiments and Programming Paradigms for Large Scale Scientific Computing on Grids, Desktop Grids and Private Clouds

by Ling Shang

A Dissertation Submitted in

Partial Fulfilment of the Requirements for the Degree of

Doctor of Philosophy in Computer Science

at

University Lille 1 Sciences and Technologies

Dissertation presented on December 6th 2010 to the Committee:

President :	***Philippe CODOGNET***	Université Paris 6
Reviewers:	***Nahid EMAD***	Université Versailles St. Quentin-en-Yvelines
	Fangmin SONG	Nanjing University
Defence Member:	***Xiaolin QIN***	Nanjing University of Aeronautics and Astronautics
	Zhijian WANG	Hohai University
Supervisor :	***Serge G. PETITON***	Université Lille 1 Sciences et Technologies

Abstract

Grid computing and Desktop Grid computing provide interesting alternatives for large scale scientific computing which needs very large scale computing resources. However gridification is hard to develop because of series of factors such as complex programming interface. The aim of this dissertation is to find a solution to make large scientific computing in an easy way. To do that, research on Gauss Jordan algorithm is made and a new parallel programming adapted version is presented. The adapted parallel version can achieve maximum degree parallelism between operations. Also the Gauss Jordan algorithm as an excellent example is used to evaluate different experimental environments and tools. Experiments with YML, OmniRPC and XtremWeb on Grid and Desktop Grid environments testify YML can be a good solution for end users to make large scale scientific computing for its series of good features such as higher level interface, component reuse and acceptable overhead. To get better performance of platform, related issues such as task granularity, data persistence and schedule mechanism are also discussed in this dissertation. According to analysis made above and the common features of Clouds possessed, YML-PC a reference architecture based on workflow for building scientific Private Clouds is proposed. YML-PC inherits those good features presented above and some other key technologies such as “data persistence”, “available time prediction” and “evaluation on heterogeneous computing nodes” for YML-PC are also discussed in this dissertation. Evaluations are made based on Gauss Jordan algorithm on Grids, Desktop Grids and Private Clouds which build on Grid5000, Polytech Lille platform, France and Hohai platform, China.

Key words: Large scale scientific computing, Gauss Jordan algorithm, Grids, Desktop Grids, Private Clouds, YML, OmniRPC, XtremWeb

Résumé

Les grilles de calcul et les grille de PC sur Internet offrent des alternatives intéressantes pour le calcul scientifique à grande échelle, qui demande des ressources de calcul importantes. Toutefois, l'adaptation des applications pour ces systèmes est difficile à cause des facteurs nombreux tels que l'interface complexe de programmation. L'objectif de cette thèse est de trouver une solution pour faciliter le calcul scientifique à grande échelle. Pour ce faire, j'ai travaillé sur l'algorithme de Gauss Jordan et une nouvelle version d'un schéma de parallélisme. Ce schéma peut exploiter le maximum de parallélisme entre des opérations. Comme un exemple excellent, l'algorithme de Gauss Jordan est également utilisé pour évaluer des environnements expérimentaux et des outils différents. Les expérimentations avec YML, OmniRPC et XtremWeb sur les grilles et les grilles de PC montrent que YML peut être une bonne solution pour que les utilisateurs fassent du calcul scientifique à grande échelle, à cause des bonnes caractéristiques comme « l'interface d'abstraction de haut niveau», « les composants réutilisables » et «le surcoût acceptable». Pour obtenir les meilleures performances de cette plate-forme, les questions concernées, telles que la granularité des tâches, la persistance des données et le mécanisme d'ordonnancement, sont également abordés dans cette thèse. Selon les analyses faites ci-dessus et les caractéristiques communes des nuages informatiques ciblés, YML-PC, une architecture de référence basée sur les workflows pour les constructions de nuages informatiques privés scientifique est proposée. YML-PC hérite les bonnes caractéristiques présentées ci-dessus et des autres technologies clefs telles que « la persistance des données », « La prévision du temps disponible » et « l'évaluation sur des nœuds de calcul hétérogènes » pour YML-PC, qui sont également abordées dans cette thèse. Les évaluations sur l'algorithme de Gauss Jordan sont réalisées sur les grilles, les grilles de PC et les nuages informatiques privés qui sont implantés sur la plate-forme Grid5000, la plateforme de calcul de Polytech Lille en France et la plateforme de calcul de Hohai, en Chine.

Mots clés: le calcul scientifique à grande échelle, l'algorithme de Gauss Jordan, grilles, grilles de PC, Nuages informatiques privé, YML, OmniRPC, XtremWeb

To all members of my family

Memory to my grandfather and grandmother

Acknowledgments

I would like to thank my PhD supervisor Prof. Serge G. PETITON, for advice and support. Thanks him again for givinig me the chance to work with him in France and I am very appreciated of his trust and help. I am grateful for working with him since his leadership was a perfect balance between guidance and liberty.

I am grateful to my friends who support and help me a lot during my study in France, special thanks to Haiwu He, Maxime Hugues, Oliver Delannoy, Ye Zhang, Laurant Choy, Eric Wartelle and other members in MAP team.

Thanks to the people in CROUS Lille, LIFL and USTL. They help me a lot during my study in France. Thanks to France Embassy in China for their financial supports. I am very appreciated of their warm hearted help and it is their help that make me study in France more smoothly. Specail thanks to Daniele Fabis, Fatima Hammadi and Juliette Jin who help me a lot during my stay in France.

I would like to express my gratitude to Philippe CODOGNET, Fangmin SONG, Nahid EMAD, Xiaolin QIN, Zhijian WANG who accepted to preside my defense. I am greatly honoured by their presence.

Contents

List of Figure

List of Table

Chapter 1

Introduction

1.1 Context

Scientific computing is the field of study concerned with constructing mathematical models and quantitative analysis techniques and using computers to analyze and solve scientific problems. In practical use, it is typically the application of computer simulation and other forms of computation to problems in various scientific disciplines. There are many examples for scientific computing such as LHC computing. The trend of scientific computing in real life is the scale of computation becomes larger and larger. Two reasons can explain that, one is that the problems to be solved become more complex and the other is the requirement of scientists to get more precise final results. To achieve satisfying results, the evolution of platform for scientific computing is made in two ways: the first is to improve the performance of computer itself. It can be achieved through integrating more CPUs or cores in a computer. But this method has the limition from the fact that communication speed can't surpass the light speed. The second way is based on distributed computing. Its improvement on computing power through harnessing more and more compouting nodes (cluster can be a very good example to explain this case). The emergency of high speed network and improvement on personal computers' processing power make Grids and Desktop Grids reality.

Generally speaking, Grids is an infrastructure to harness distributed and heterogeneous dedicated computing resources from all over the world. It can deal with almost all the problems solved in traditional supercomputer. Many Grid projects have been launched and achieved fruitful results. For example, TeraGrid integrates high-performance computers, data resources and tools, and high-end experimental facilities around USA. Currently, TeraGrid resources include more than a petaflop of computing capability and more than 30 petabytes of online and archival data storage, with rapid access and retrieval over high performance networks. Researchers can also access more than 100 discipline-specific databases. With this combination of resources, the TeraGrid is the world's largest, most comprehensive distributed cyberinfrastructure for open scientific research. At the same time, lot of Grid middlewares which can harness distributed computing resources into a virtual platform, are developed by different research groups in different countries. Such as, Globus Toolkit, Gridsolve, OmniRPC…

Normally, Desktop Grids is an infrastructure aiming at collecting idle CPU cycle of volunteer computers under the Internet based environment. Lately, some researchs on harnessing idle CPU cycle in local network based environment (for example, in the environment of inner enterprises and research institutes) also belong to Desktop Grid research areas. As well known to us all, thoses collected CPU cycle can deal with very large scale problems which are difficult to finish in the fastest supercomputer. SETI@Home is a good example. SETI@Home is the program and the project that involves millions of at home users from around the world in the Search for Extraterrestrial Intelligence (SETI). Since 1999, volunteers have been able to recycle their unused computing cycles in helping to analyze data collected from radio telescopes in search of signals from other worlds. This unprecedented network of over 5 million independent SETI@Home volunteers constitutes the world's largest supercomputer. The success of SETI@Home makes scientists be aware of the importance of volunteer computing resrouces in scientific computing. Many Desktop Grid middlewares also have been developed such as Boinc, XtremWeb and Entropia.

Cloud computing is a buzzword and umbrella term applied to several nascent trends in the turbulent landscape of information technology. Computing in the Clouds alludes to ubiquitous and inexhaustible on-demand IT resources accessible through the Internet. Many famous IT enterprises (Google, Amazon, Microsoft) have launched their Cloud products (Chrome, EC2, Azure) and a lot of research insititutes (Berkeley university, for example) show their understandings on Clouds. To summary, the advantages of large scale scientific computing in Clouds are that the Clouds can provide end user a higher level interface; users can get those services in Clouds with lower costs and those services are on demand; resources in Clouds can be utilized in maximum degree, i.e., Clouds can help to fully dig the poentencial of computing resources. Some middlewares are proposed such as Eucalyptus, but more architectures and middlewares for Clouds are under research.

The INRIA Futurs Grand Large project is a pioneer in large scale heterogeneous computing and the globalization of computer resources and data Grid initiative. The project is especially interested in large scale parallel and distributed systems supposed to work over the Internet, intranets, LANs or broadband networks. The project's approach concerns middleware and low level programming environments, between low level system mechanisms and high level programming environments. MAP team in LIFL aims to make research on parallel method and algorithm in scientific domain which belongs to application level of scientific computing. The scientific experiments often use the tools developed by Grand Large project. The MAP team plays a major role in the necessary testing step of computing software and it provides a precious user feed back.

1.2 Motivation and scope of study

The goal of researches in this dissertation comes from the collaboration between MAP team, LIFL, university Lille1 science and technology, France and college of computer and information, Hohai University, China.

Scientific computing in Hohai university is mainly on numerical computing in water resources domain. Its situation in this field can be summarized as follows:

- Most scientific compouting is based on Cluster which belongs to dedicated computing resources.
- It is not easy to scale up this kind of computing platform with low costs.
- The computing resources are not enough for scientific computing.
- Programming on the existing platform is not easy, especially to those end users who are non professional computer.

To settle the problems presented above in real application environments, collaboration between MAP team, France and Hohai, China is made. The general idea can be described as follows:

- Make volunteer computing resources as an extension of the existing platforms which consist of dedicated computing resources. The reasons can be summarized as follows: 1) there are a lot of volunteer computing resources can be collected; 2) their costs are very low and almost nothing; 3) volunteer computing resources based platform has the ability of scalability by nature.

- Try to provide end user with a high level programming interface. The interface is better to be platform/software independent, i.e., users can program with the interface without knowing about the hardware infrastructure of the platform and softwares deployed.

To achieve the goal presented above, the scope of study can be desribed as follow:

- Researches on Grid and Desktop Grid middleware have been made. Related issues such as programming model, influence on Grids and Desktop Grids from task granularity, data persistence and schedule mechanism are also discussed in this dissertation.

- Researches on parallelism of block based Gauss Jordan (BbGJ) algorithm have been made. The performance of new parallel adapted algorithm and traditional parallel algorithm will be compared. BbGJ as an example for scientific computing is also used to evaluate the performance of different middlewares and platforms in this dissertation.

- Researches on Cloud computing have been made. Some famous architecures for Cloud computing by famous enterprises will be dissected. What are the features Clouds should possess for scientific computing will also be discussed. Finally, according to the analysis, we will try to propose a solution to build Private Clouds system.

1.3 Contributions

The research goal and scope of study in this dissertation are presented in the previous section. In this section, we will introduce our works in detail to achieve the goal.

The first contribution is on presenting a new parallel programming adapted version for block based Gauss Jordan algorithm. Block based Gauss Jordan algorithm is a classical numerical algorithm to invert large scale matrix. Previous recent works for Grid, Desktop Grid and Cluster computing just focus on intra-iterative step based parallelism without considering inter-steps based parallelism, as past experiments on Gauss Jordan just for smaller computers. This dissertation analyzes all the potential parallelism in the algorithm and exploits both the inter-iteratives step and intra-iterative step based parallelism, which depend on data dependence between basic operations. Then analysis on those data dependences is made through an intuitive tool which is a series of tables and according to the analysis on those tables, regularity of those data dependences can be summarized. Then, formal descriptions on the regularity of data dependences between operations are made. According to those formal descriptions, executable rules on each operation in Gauss Jordan algorithm are made, i.e., whether the operation can be executed or not will totally depend on the executable rules. Based on those executable rules and the method of designing parallel algorithm, a new parallel programming adapted version for Guass Jordan algorithm is presented. The new parallel adapted vesion can adapt to any programming environment and it is platform independent, i.e., you can program with any programming tool (MPI, Fortuan, RPC based programming interface and so on) you would like to use. The advantage of this new parallel adapted version is its achieving maximum degree parallelism between operations and it helps to generate more tasks and make them executed in parallel, thus decreasing the time span of overall program. So it can achieve better performance than that of old ones under the situation of enough computing resources. (Chapter 4)

The second contribution is on survey of evolution of large scale scientific computing systems. Traditional large scale scientific computing platform mainly adopts three kinds of high performance systems. First, it is the commodity systems consisting of commodity commercial off-the-shelf microprocessors or customized commercial off-the-shelf microprocessors; Second, it is the hybrid systems composed of commercial off-the-shelf microprocessors, customized commercial off-the-shelf and customized interconnections (for example, TSUBAME at TITech); Third, it is custom systems which are entirely made of custom microprocessors and custom interconnections (e.g., Earth simulator). To summary, they all belong to supercomputers. But with the development of commercial off-the-shelf microprocessors and emergence of high speed network, three kinds of new paradigms become alternatives of making large scientific computing, which are Cluster, Grids and Desktop Grids. Generally speaking, Cluster and Grids are used to harness dedicated computing resources and Destop Grids aims at collecting idle CPU cycle of non-dedicated compouing resources connected to Internet. This dissertation makes detail description on Grids and Desktop Grids, because our experimental environments are based on those two kinds of parallel paradigms. The description focuses on their programming model, data transfer model, schedule mechasnism, characters and challenges. Primary comparison between Grids and Desktop Grids is also made in Chapter 2. In this dissertation, OmniRPC for harnessing dedicated computing resources and XtremWeb for collecting non dedicated computing recources and YML a workflow based framework, are chosen as our experimental tools which will be introduced detailly in Chapter 3. Further description about relation between YML and OmniRPC /XtremWeb is also made. (Chapter 2 and Chapter 3)

The third contribution is on evaluating large scale scientific computing on Grids and Desktop Grids, with Gauss Jordan algorithm as an example. As mentioned above, we use OmniRPC to harness dedicated computing resouces and XtremWeb to collect non dedicated computing resouces. But it is not easy for end users especial to those non professional computer users, to program directly using the interfaces provided by OmniRPC and XtremWeb. Through comparing the programming model of YML with that of OmniRPC and XtremWeb, we can conclude that YML can provide end users with a high level interface which are pseudo code based and software/platform independent. What we want to emphasize here is its character of platform independent, i.e., the code developed using YML can be run both on Desktop Grid and Grid platform without any change. As far as we know, little middleware can do it. Then analysis on overhead of YML is made and experiments tesify its overhead is acceptable. The reusable character of YML’s components can help to decrease the time/cost to solution when making large scale scientific computing. To summary, YML can be a good solution for end users to make large scale scientific computing for its series of good features (higher level programming interface, component reuse, support invocating third party libiary, acceptable overhead). We also discuss other factors influencing the performance of platform in Grid and Desktop Grid environments. Fine-grain based tasks parallelism can’t always achieve better performance in Desktop Grid environment for its low speed network. Worker-to-worker data transfer model can help computing platform to achieve better performance both in

Grid environment and Desktop Grid environment. Schedule mechanism in Desktop Grid environment should take the heterogeneous (from CPU, Memeory, Network…) into consideration. Only that, the stable performance of the platform can be achieved. (Chapter 5)

The fourth contribution is on presenting a reference architecture for building Private Clouds. Cloud computing has aroused great interests from indursties and research institutes since its birth. Many famous IT enterprises such as Google, Amazon, IBM and Microsoft have launched their products. Many prestigious scientists such as Ian Foster, Buyya and Geoffrey Fox also show their understandings on Cloud computing. But unfortunately, there is no agreement on what Cloud computing is. This dissertation dissects the existing Cloud system and summarizes their common characters behind different appearances of those Clouds. A viewpoint from Grids to observe Clouds based on lessons learned from Grid researches and our experiences on Grid5000 is proposed. Then according to those common characters and lessons from utilizing Grid systems, we present our viewpoint on Cloud computing in scientific computing area: *Cloud computing is a specific problem solving environment based on large scale resources pool (consist of clusters, grid, desktop grid, supercomputer or hybrid platform). It encapsulates all technological details through virtual technology and can provide end users with on demand provision, non-trivial quality of service and pay by use, easy of use, high level program interface; End users can utilize all these services provided by Cloud platform in a very simple way without knowing where these services come from and on what kinds of platform/ system/ infrastructure these services run.* Based on the idea of our definition, we extend YML and present YML-PC which is a reference architecture based on workflow for building scientific Private Clouds. Some important features of YML-PC can be summarized as follows:

- YML-PC can provide end users a pseudo code based programming interface which can help to decrease the burden of users' programming. The developed code can be run both on Grid and Desktop Grid platforms without any change.

- The components of YML-PC can be reused, i.e., users can reuse the developed components in different parallel algorithm without any change. Those components are "independent operations" of parallel algorithm. For example, the operation "matrix product" can be invoked by both "Matrix Prodcut algorithm" and "Gauss Jordan" algorithm without any change. This help to reduce time/cost to solution of their scientific computing.

- YML-PC also can harness dedicated computing resources and volunteer computing resources at the same time. As well known to us all, volunteer computing resources have huge processing power beyond imagination and they possess the character of scalabitliy by nature. So the computing resources pool of YML-PC can be scaled up dynamically through collecting more volunteer computing resouces into platform and thus, the platform can provide processing power on deamnd for users with low costs.

Some research works on extending YML to YML-PC in this dissertation can be summarized as follows:

- Evaluate the hybrid computing environment (dedicated computing resources and volunteer PCs) with large scale scientific computing (Gauss Jordan algorithm as an example). As far as we know, no other evaluation is made using two different kinds of computing resources at the same time.

- Data flows. Data flows can be added in the "application file" of YML. Through adding data flow, data persistence and data anticipated migration can be realized in YML-PC. And it can help to improve the efficiency of platform greatly.

- Monitor and Trust model. They are introduced to monitor available status of non dedicated computing resources (volunteer PCs). The aim is to predicate future available status of non dedicated computing resources. Also a method to evaluate expected execution time based on standard virtual machine is adopted. Through this method, heterogeneous computing resources can be changed into homogeneous computing resources and then can be evaluated. According to this evaluation and predication, tasks can be allocated to appropriate computing resources.

(Chapter 6)

1.4 Organisations

The following chapters will be organized as follows:

Chapter 2 will introduce the evolution of large scale scientific computing platform and more details will be described on Grids and Desktop Grids. The experiment environments and tools in this dissertation are presented in Chapter 3. Chapter 4 will propose a new parallel version adapted to Grid and Desktop Grid environments for Guass Jordan algorithm based on analysis on data dependence between different operations. As an example for large scale scientific computing in Chapter 5, Gauss Jordan algorithm will be used to evaluate different middlewares and environments on related issues such as programming model, overhead of middleware, data perisitence and schedule mechanism. Cloud compouting in scientific domain is discussed in Chapter 6 and a reference architecture for building Private Clouds is also presented. Finally, conclusion and future works are summarized in Chapter 7.

Chapter 2

The State of the Art

Scientific Computing is the collection of tools, techniques, and theories required to solve mathematical models of problems in science and engineering on computer system [1]. Generally speaking, it is the fields of study concerned with constructing mathematical models and numerical solution techniques, and with using computers to analyze and solve scientific and engineering problems. Scientific Computing draws on modeling in science and engineering, numerical mathematics, and computer science to develop the best ways to use computer systems to solve problems from science and engineering. The Scientific Computing approach is to gain understanding, mainly through the analysis of mathematical models implemented on computers. As Richard Hamming has observed many year ago, the purpose of Scientific Computing is insight, not numbers [2]. The process of making scientific computing can be described using Figure 2.1.

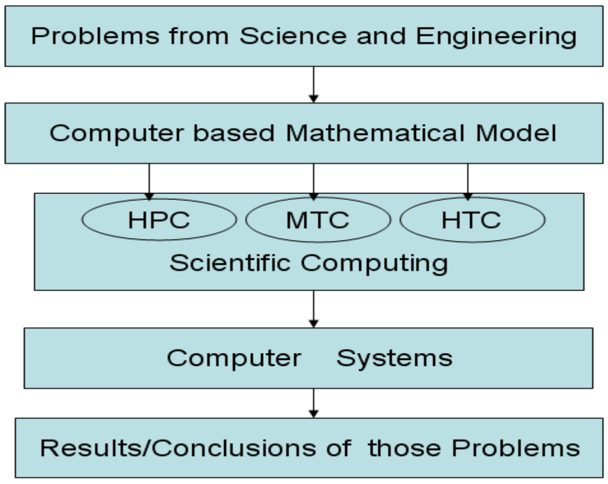

Problems from Science and Engineering
Computer based Mathematical Model
HPC
MTC
HTC
Scientific Computing
Computer Systems
Results/Conclusions of those Problems

Figure.2.1 Process of scientific computing on computer systems

Scientific Computing involves a broad range of applications, from high-performance computing (**HPC**) which is heavily focused on compute-intensive applications, high-throughput computing (**HTC**) which focuses on using many computing resources over long periods of time to accomplish its computational tasks, many-task computing (**MTC**) which aims to bridge the gap between HPC and HTC by focusing on using many resources over short periods of time, to data-intensive computing which is heavily focused on data distribution and harnessing data locality by scheduling of computations close to the data.

To solve these problems from science and engineering, the notion of parallel computer is proposed. *Parallel Computer is a collection of processing elements that communicate and cooperate to solve large problems fast* [3]. The need for processing elements increases with more and more scientific problems are proposed. At the same time, more complex scientific models and more accurate results also call for more processing elements. Experiences from over past few decades show the amount of computation required increases faster than the computing capacities provided by parallel computer.

To support the demand of computation capacity, large scale scientific computing platforms (LSSCP) evolve at a fearsome speed and they will soon reach the capacity of executing a Peta floating point operations per second. In a little more than ten years, large scale scientific computing platforms have multiplied their capacities a thousand times. The success of LSSCP comes from the evolution of its architecture (supercomputer, distributed computing system), hardware (processors, networks) and software (operating systems and programming environments). Next, the evolution of LSSCP will be described in detail.

2.1 Evolution of large scale scientific computing platform

2.1.1 Classification of architecture

According to the taxonomy of Flynn [5], execution models of LSSCP can be classified four types.

SISD single instruction single data	SIMD single instruction multiple data
MISD multiple instructions single data	MIMD multiple instructions multiple data

Figure.2.2 Classifications according to Flynn

SISD machines: only one instruction stream is being acted on by the CPU during any one clock cycle (Single instruction). Only one data stream is being used as input during any one clock cycle (Single data). Module of execution can be deterministic execution. These are the conventional systems that contain one CPU and hence can accommodate one instruction stream that is executed serially.
Remark: older generation mainframes, minicomputers and workstations; most PCs today adopt this structure.

SIMD machines: all processing units execute the same instruction at any given clock cycle (Single instruction). Each processing unit can operate on a different data element (Multiple data). Module of execution can be synchronous (lockstep) and deterministic execution. Single Instruction Multiple Data systems often have a large number of processing units, ranging from 1,024 to 16,384 that all may execute the same instruction on different data in lock-step. So, a single instruction manipulates many data items in parallel. Two varieties of SIMD machines are Processor Arrays and Vector Pipelines. Some examples of those machines are: Connection Machine CM-2, MasPar MP-1 & MP-2, ILLIAC IV (belong to Processor Arrays) and IBM 9000, Cray X-MP, Y-MP & C90, Fujitsu VP, NEC SX-2, Hitachi S820, ETA10 (belong to Vector Pipelines).
Remark: most modern computers, particularly those with graphics processor units (GPUs) employ SIMD instructions and execution units.

MISD machines: a single data stream is fed into multiple processing units. Each processing unit operates on the data independently via independent instruction streams. But no practical machine in this class has been constructed nor are such systems easy to conceive. The only one experiment platform is the experimental Carnegie-Mellon C.mmp computer (1971).

MIMD ma chines: every processor may be executing a different instruction stream and every processor may be working with a different data stream. Module of execution can be synchronous or asynchronous, deterministic or non-deterministic. MIMD systems may run many subtasks in parallel in order to shorten the time-to-solution for the main task to be executed. Most current supercomputers, networked parallel computer clusters and "grids", multi-processor SMP computers, multi-core PCs belong to MIMD systems.
Remark: now, MIMD systems are the main stream architectures of LCSSP. Grid, Desktop Grid and future Clouds should belong to this category. Also, many MIMD architectures also include SIMD execution sub-components

The Flynn taxonomy is not enough to classify architecture of LCSSP. Nowadays almost all systems fall in the MIMD class of machines. However, the classification of Flynn can be refined according to its memory model used.

2.1.1.1 Shared memory systems

Shared memory systems have multiple CPUs and those CPUs have the ability to access all memory as global address space. In shared memory systems, the communication between processors occurred through memory accesses. The memory controller is often more complex than in other systems. It is responsible for retrieving memory area accessed for reading and writing and to maintain the consistency of local caches. Some characters can be summarized as follows:

- Multiple processors can operate independently but share the same memory resources.
- Changes in a memory location effected by one processor are visible to all other processors.

Generally speaking, Shared memory machines can be divided into two main classes based upon memory access times: ***UMA*** and ***NUMA***.

UMA: All the processors in the UMA model share the physical memory uniformly. In UMA architecture, access time to a memory location is independent of which processor makes the request or which memory chip contains the transferred data. In the UMA architecture, each processor may use a private cache. The UMA model is suitable for general purpose and time sharing applications by multiple users. It can be used to speed up the execution of a single large program in time critical applications [6].

NUMA: NUMA is a computer memory design used in multiprocessors, where the memory access time depends on the memory location relative to a processor. Under NUMA, Each group of processors has its own memory and possibly its own I/O channels. However, each CPU can access memory associated with the other groups in a coherent way. Each group is called a NUMA node. The number of CPUs within a NUMA node depends on the hardware vendor. It is faster to access local memory than the memory associated with other NUMA nodes. NUMA systems distribute the memory to each processor. In such systems, the network of inter-connexion is still used for all memory operations (read/write). However, the round trip time to retrieve a memory area is not fixed and varies depending on the distance between the two processors involved.

Pros: Global address space provides a user-friendly programming perspective to memory; Data sharing between tasks is both fast and uniform due to the proximity of memory to CPUs
Cons: Primary disadvantage is the lack of scalability between memory and CPUs. Adding more CPUs can geometrically increases traffic on the shared memory-CPU path, and for cache coherent systems, geometrically increase traffic associated with cache/memory management; Programmer responsibility for synchronization constructs that insure "correct" access of global memory; Expense: it becomes increasingly difficult and expensive to design and produce shared memory machines with ever increasing numbers of processors.

2.1.1.2 Distributed memory systems

In computer science, distributed memory refers to a multiple-processor computer system in which each processor has its own private memory. Computational tasks can only operate on local data, and if remote data is required, the computational task must communicate with one or more remote processors. In a distributed memory system there is typically a processor, a memory, and some form of interconnection that allows programs on each processor to interact with each other. The inter-connection can be organised with point to point links or a special switching network. The network topology is a key factor in determining how the multi-processor machine scales. The links between nodes can be implemented using some standard network protocol (for example Ethernet), using bespoke network links (used in for example the Transputer), or using dual ported memories[1]. The characters can be summarized as follows:

- Distributed memory systems require a communication network to connect inter-processor memory. The network "fabric" used for data transfer varies widely, though it can be as simple as Ethernet.
- Processors have their own local memory. Memory addresses in one processor do not map to another processor, so there is no concept of global address space across all processors.
- Each processor has its own local memory, it operates independently. Changes it makes to its local memory have no effect on the memory of other processors. Hence, the concept of cache coherency does not apply.
- When a processor needs access to data in another processor, it is usually the task of the programmer to explicitly define how and when data is communicated. Synchronization between tasks is likewise the programmer's responsibility.

Difference between distributed shared memory system and distributed memory system: distributed shared memory system offers a single memory space used by all processors. Processors do not have to be aware where data resides, except that there may be performance penalties, and that race conditions are to be avoided. While distributed memory system need users to control when and how to communicate between nodes. The advantage of distributed memory system is that it can be scaled very easily.

Pros: memory is scalable with number of processors. Increase the number of processors and the size of memory increases proportionately; each processor can rapidly access its own memory without interference and without the overhead incurred with trying to maintain cache coherency; Cost effectiveness: can use commodity, off-the-shelf processors and networking.
Cons: The programmer is responsible for many of the details associated with data communication between processors; It may be difficult to map existing data structures, based on global memory, to this memory organization;

2.1.1.3 Convergence and future systems

The largest and fastest computers in the world today employ both shared and distributed memory architectures. It belongs to Hybrid Distributed-Shared

[1] http://en.wikipedia.org/wiki/Distributed_memory

Memory system. Just as the analysis made in previous sections, the shared memory component is usually a cache coherent SMP machine and processors on a given SMP can address that machine's memory as global. While the distributed memory component is the networking of multiple SMPs and those SMPs only know about their own memory - not the memory on another SMP. Therefore, network communications are required to move data from one SMP to another. Now, current trends seem to indicate that hybrid distributed shared memory component will continue to prevail and increase at the high end of computing for the foreseeable future.

Nowadays, the performance of regular processors has been improved greatly and scientists are likely to use those kinds of processors to build high performance systems for its lower costs. High performance systems also converge on the aspect of interconnection by high speed network [7] [8] [9]. The concept of Grid computing is proposed in 1999 by Ian Foster [11][12] [13]. He describes the Grid as follows: A computational grid is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities. Through the description, you can imagine several million computers (include desktops, laptops, supercomputers, data vaults, and instruments like mobile phones, meteorological sensors and telescopes) from all over the world, and owned by thousands of different people. And all of these computers can be connected to form a single, huge and super-powerful computer! This huge, sprawling, global computer is what many people dream “The Grid” will be. Although "the Grid" is still just a dream, grid computing is already reality. Many famous grid projects have been built up successfully, such as WLCG[2], EGEE[3] and TeraGrid[4].

In the next parts, we will make summary on the roadmap of high performance systems.

2.1.2 Evolution roadmap of high performance systems

“...With the advent of everyday use of elaborate calculations, speed has become paramount to such a high degree that there is no machine on the market today capable of satisfying the full demand of modern computational methods.” - from the ENIAC patent (U.S.#3,120,606) filed on June 26, 1947.

From the birth of first electric computer in human history, scientists have been working on the improvement of computer performance. To achieve the goal of time to solution, many kinds of architectures of high performance systems are proposed. LSSCP have evolved from single-user environments, to Massively Parallel Processors (MPPs), to clusters of workstations, to distributed systems, and recently to grid computing systems and even to cloud computing system.

[2] http://lcg.web.cern.ch/LCG/
[3] http://public.eu-egee.org/
[4] https://www.teragrid.org/

Table 2.1 shows us its evolution roadmap.

From the Table2.1, there are two ways to improve the performance of a "computer" in the past 60 years. The first method is to improve the process power of a computer itself. This method is achieved through the development of computer chip technology and it went through these following stages: tubes and relays, transistor, central processor, vector processor, processor array, integrated circuit, multi processor, large scale integrated circuit (LSI) and very large scale integrated circuit (VLSI). The second manner is based on the emergency of high speed network and through which, lots of workstations/personal computers are harnessed to form Clusters, Grids and Desktop Grids.

Figure 2.3 shows us the trend of how to make large scale scientific computing. Nowadays, Clusters of computers are becoming increasingly popular solutions for high performance computing systems (HPC). For instance, architecture share for clusters in the top 500 supercomputer sites reaches 83.4% in November 2009, compared to a share of merely 16.2% seven years ago[5]. As the performance/price ratio of PC components and LAN connections keep increasing, more and more organizations and companies build computer clusters for matching the needs of their applications. A cluster within one administrative domain, or one site, typically consists of a number of processing units/nodes connected via networks (commonly Ethernets). With the increase of complexity of computational model and the requirement of calculation accuracy, clusters can't meet the requirement from scientific computing. Grids, proposed in 1999 are used to harness clusters, supercomputer and a kind of devices to realize the expanding of computational power. What Grids mainly are used to integrate is dedicated computing resources. At the same time, the success of Seti@home makes people know about that huge process power can be collected through stealing CPU cycles of personal computers from all over of world. Paper [14] also analyzes and testifies the huge potential capability to utilize volunteer computing resources. Then Desktop Grids, used to harness volunteer computing are proposed and arouse great interests of scientific researchers. After 10 years' efforts, many middlewares for Grids [15][16][17] and Desktop Grids [19][20] are developed. As described above, Grids and Desktop Grids have harnessed huge process power and to make full use of them, Clouds are proposed to provide end users with an easy-of-use and lower cost way. Three kinds of research on Clouds are made which are private clouds, public clouds and hybrid clouds. Generally speaking, public clouds refer to those clouds can provide services to the third parties. Such as Google can provide email services to people all over the world; Amazon can provide computing resources to who need them at different levels services (CPU, Core or platform). Private clouds refer to those clouds which are built for themselve and those Clouds will not provide services to the third parties. While hybrid clouds represent those clouds can provide services and those services come from public clouds and private clouds. This dissertation will make numerical computing on Grids platform, Desktop Grids and private Clouds. Here, what we want to emphasize is

[5] Statistics of architecture share for top 500 supercomputer sites are obtained from http://www.top500.org

that the private Clouds in this dissertation is designed for non-large enterprises and academic research institutes according to real requirements from China.

In the following sections, more details will be described on Grids, Desktop Grids and Clouds.

Table 2.1 Evolution Roadmap of High Performance System

Year	Name	Inventors	Key technology
1946	ENIAC	Moore School of Electrical Engineering, University of Pennsylvania	Can't store program in the ENIAC
1951	UNIVAC I	J.Presper Eckert and John William Mauchly	Tube Computer
1956	TX-0	MIT Lincoln Laboratory	Transistor Digital Computer
1960	LARC	UNIVAC company	2 CPU + I/O processor
1963	CDC6600	Cray	1 central processor + multi peripheral processor; instruction Pipeline
1964	ILLAC IV	University of Illinois	4 CU + 256 PE (13MHz); originator of MPP
1974	CDC STAR-100	Jim Thornton	Vector Processor
1975	Cray-1	Cray Research	Vector pipeline computer with integrated circuit
1985	VPP series	Fujitsu	Scalable Vector Parallel Computer Architecture
1986	CM-1	Thinking Machines Corporation	SIMD with Hypercube Architecture
1991	CM-5	Thinking Machines Corporation	Multiple Instruction stream Multiple Data stream
1991	Touchstone Delta	Inter	Micro Processor with 2D inter-connection
1993	T3D	Cray Research	3D torus inter-connection with 2048 DEC alpha MC21064 RISC micro processors at most.
1994	Beowulf	NASA	With 16 486DX4 micro computers inter-connected by 10Mb/s Earthnet
1997	NOW	Berkeley	100 UltraSPARC SUN workstations inter-connected by Myrinet
1999	Grids	Ian Foster	Harness dedicated computing resources
2000	Desktop Grids	Berkeley	Harness non-dedicated computing resources
2007	Clouds	IBM	Provide computing resources with lower cost, easy-of-use

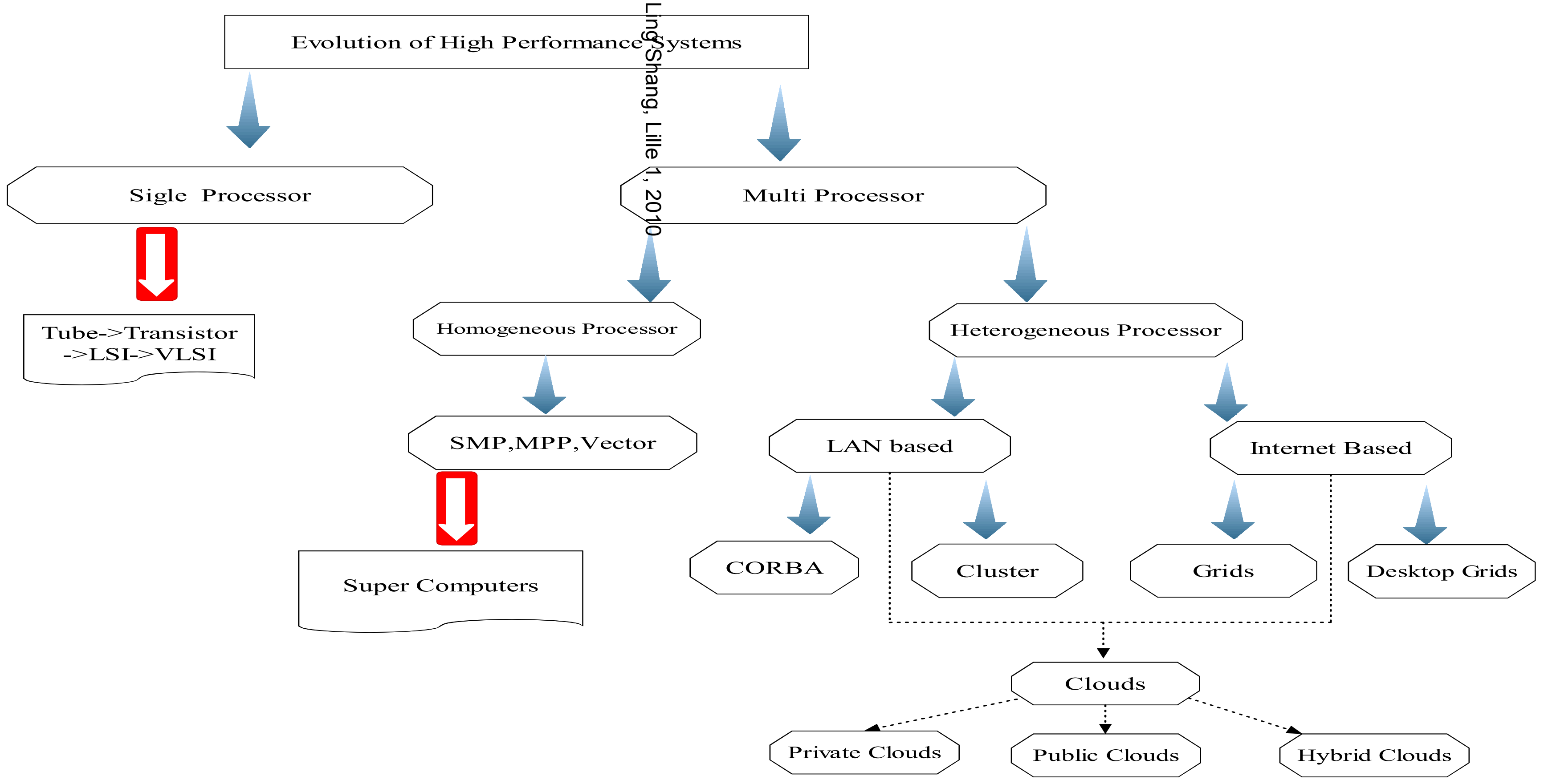


Figure.2.3 Evolution of high performance systems

2.2 Grid Computing

2.2.1 Introduction

The last decade has seen a substantial increase in commodity computer (PCs) and network performance, mainly as a result of faster hardware and more sophisticated software. These commodity technologies have been used to develop low-cost high-performance computing systems, popularly called clusters, to solve resource-intensive problems in a number of application domains. However, there are number of problems, in the fields of science, engineering, and business, which are not tractable using the current generation of high-performance computers. In fact, due to their size and complexity, these problems are often resource (computational and data) intensive and they also need to work collaboratively with distributed interdisciplinary application models and components. Consequently, such applications require a variety of resources that are not available in a single organisation.

The popularity of the Internet and the availability of powerful computers and high-speed networks as low-cost commodity components are changing the way of using computers today. This technology opportunity has led to the possibility of using networks of computers as a single, unified computing resource. It is possible to cluster or couple a wide variety of resources including supercomputers, storage systems, data sources, and special classes of devices distributed geographically and use them as a single unified resource, thus forming what is popularly known as Grid Computing [21]. Grid computing can be defined as the coordinated resource sharing and problem solving in dynamic, multi-institutional collaborations [23]. More simply, Grid computing typically involves using many resources (computer, data, I/O, instruments, etc.) to solve a single, large problem that could not be executed on any one resource. As a matter of fact, various Grid application scenarios have been explored within both science and industry. These applications include compute-intensive, data-intensive, sensor-intensive, knowledge-intensive and semantics-intensive scenarios. In this dissertation, research issues will be focused on computational grid.

There are three main issues that characterize Computational Grids: heterogeneity, scalability and dynamic adaptability. Grid involves a multiplicity of resources that are heterogeneous in nature and might span numerous administrative domains across wide geographical distances. Moreover, a Grid might grow from few resources to millions. This raises the problem of potential performance degradation as the Grid size increases. Consequently, applications that require a large number of geographically located resources must be designed to be extremely latency tolerant. Finally, in a Grid, resource failures are unavoidable. In fact, with so many resources in a Grid, the probability of some resource failing is naturally high. The resource managers or applications must tailor their behaviours dynamically so as to extract the maximum performance from the available resources and services.

To achieve the greatest performance of Grid systems, five stages can be divided into for the evolution of Grid computing systems. Those five stages are: early stage of Grids; early experimental stage of Grids; development stage of Grids; application stage of Grids and improvement stage of Grids.

- Embryonic stages of Grids: the idea of Grids comes from I-Way project [22]. It aims at demonstrating the feasibility of connecting 17 institutions providing heterogeneous computing resources using 1Gbit/s dedicated network for high performance applications.
- Early experimental stage of Grids: some middleware for grid computing have been developed based on the ideas from first stage. During this stage, representative projects are Globus[6] and Legion[7].
- Development stage of Grids: Since the release of the Globus Toolkit 3.0, the Globus Project offers an open source collection of Grid services that follow Open Grid Services Architecture (OGSA[8]) architectural principles. The Globus Toolkit also offers a development environment for producing new Grid services that follow OGSA principles. WSRF is a set of Web service specifications being developed by the OASIS organization. Taken together and with the WS-Notification (WSN) specification, these specifications describe how to implement OGSA capabilities using Web services. The Globus Toolkit 4.0 and later versions provide an open source WSRF development kit and a set of WSRF services. Based on OGSA and WSRF, many grid middlewares are developed. Such as Ninf, Gridsolve, OmniRPC, Ninf-G. Those grid systems reflect the goal of the Nowadays grid projects are based on this work.
- Application stage of Grids: with almost 10 years' effort, many Grids system are established and begin to apply in many areas of scientific computing. There are many famous and influential projects during this stage. Such as: TeraGrid, EGEE, LCG and Earth System Grid [18].
- Improvement stage of Grids: to achieve higher performance, researches on quality of service and service level agreement of Grids system are made during this stage. Many compute and data-intensive functionalities in scientific and grid workflows (such as linear algebra, image processing, database searching, etc.) are characterized by coarse-grained parallelism that allows for increasing performance by exploiting a pool of distributed resources using parallel computing patterns such as simple parallelism, data parallelism and pipeline patterns A full exploitation of multiple resources to execute Grid workflows need take the following main issues into account: (1) the adoption of matching strategies able to find a pool of resources satisfying global constraints on applications; (2) definition of formal languages for QoS [26][27][28] description of Grid services in order to avoid ambiguity during matching; (3) mechanisms for dynamic and transparent composition and coordination of services. Based on the ideas of appropriate matching between

[6] http://www.globus.org/

[7] http://legion.virginia.edu/index.html

[8] http://en.wikipedia.org/wiki/Open_Grid_Services_Architecture

services provided by Grids and jobs generated from end-users' applications, the requirement of easy-of-use for end users is necessary to consider. So the cloud computing emergences. This dissertation will take cloud computing as a layer based on Grids system and this layer can make grids system be more efficient and easy-of-use.

The evolution of Grids system can be described through Figure2.4.

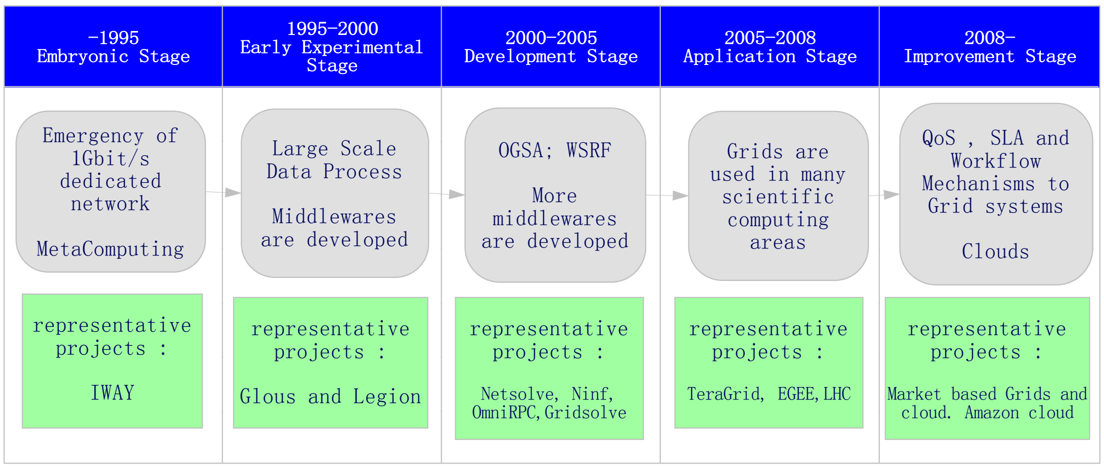

-1995
Embryonic Stage
1995-2000
Early Experimental Stage
2000-2005
Development Stage
2005-2008
Application Stage
2008-
Improvement Stage
Emergency of 1Gbit/s dedicated network
MetaComputing
Large Scale Data Process
Middlewares are developed
OGSA; WSRF
More middlewares are developed
Grids are used in many scientific computing areas
QoS , SLA and Workflow Mechanisms to Grid systems
Clouds
representative projects :
IWAY
representative projects :
Glous and Legion
representative projects :
Netsolve, Ninf, OmniRPC, Gridsolve
representative projects :
TeraGrid, EGEE, LHC
representative projects :
Market based Grids and cloud. Amazon cloud

Figure.2.4 Evolution of Grid systems

Just as described above, Grids enable the sharing, exchange, discovery, selection, and aggregation of geographically distributed heterogeneous resources, such as computers, databases, visualization devices, and scientific instruments. Accordingly, they have been proposed as the next-generation computing platform and global cyber-infrastructure for solving large-scale problems in science, engineering, and business. Unlike traditional parallel and distributed systems, Grids address issues such as security, uniform access, dynamic discovery, dynamic aggregation, and quality-of-services. A number of prototype applications have been developed and scheduling experiments have been carried out within Grids [24] [25]. The results of these efforts demonstrate that the Grid computing paradigm holds much promise. Furthermore, Grids have the potential to allow the sharing of scientific instruments such as particle accelerator, telescope and synchrotron that have been commissioned as national/international infrastructure due to the high cost of ownership and to support on demand and real-time processing and analysis of data generated by them. Such a capability will radically enhance the possibilities for scientific and technological research and innovation, industrial and business management, application software service delivery and commercial activities, and so on.

In order to provide users with a seamless computing environment, the Grid middleware systems need to solve several challenges originating from the inherent features of the Grid [29]. One of the main challenges is the heterogeneity in grid environments, which results from the multiplicity of heterogeneous resources and the vast range of technologies encompassed by the Grid. Another challenge involves the multiple administrative domains and autonomy issues because of geographically distributed grid resources across multiple administrative domains and owned by different organizations. Other challenges include scalability (problem of performance degradation as the size of Grids increase) and dynamicity/ adaptability (problem of resource failing is high).

Middleware systems must tailor their behavior dynamically and use the available resources and services efficiently and effectively. Middleware is made up of many software programs, containing hundreds of thousands of lines of computer code. Together, this code automates all the "machine to machine" (M2M) interactions that create a single, seamless computational grid. In Grid systems, Middleware can automatically negotiate deals in which resources are exchanged, passing from a Grid resource provider to a Grid user. There are many layers within the middleware to achieve the goal mentioned above. Generally speaking, middleware includes a layer of "resource and connectivity protocols" (core middleware in Figure2.5), and a higher layer of "collective services" (user level middleware in Figure2.5). Resource and connectivity protocols handle all grid-specific network transactions between different computers and grid resources. For example, computers contributing to a particular grid must recognize grid-relevant messages and ignore the rest. This is done with communication protocols, which allow the resources to communicate with each other, enabling exchange of data, and authentication protocols, which provide secure mechanisms for verifying the identity of both users and resources. The collective services are also based on protocols: information protocols, which obtain information about the structure and state of the resources on a Grid, and management protocols, which negotiate uniform access to the resources. Collective services include: 1) updating directories of available resources; 2) brokering resources (which like stock broking, is about negotiating between those who want to "buy" resources and those who want to "sell").; 3) monitoring and diagnosing problems; 4) replicating data so that multiple copies are available at different locations for ease of use; 5) providing membership/policy services for tracking who is allowed to do what and when. The components of Grid middleware that are necessary to form a Grid can be described using Figure2.5.

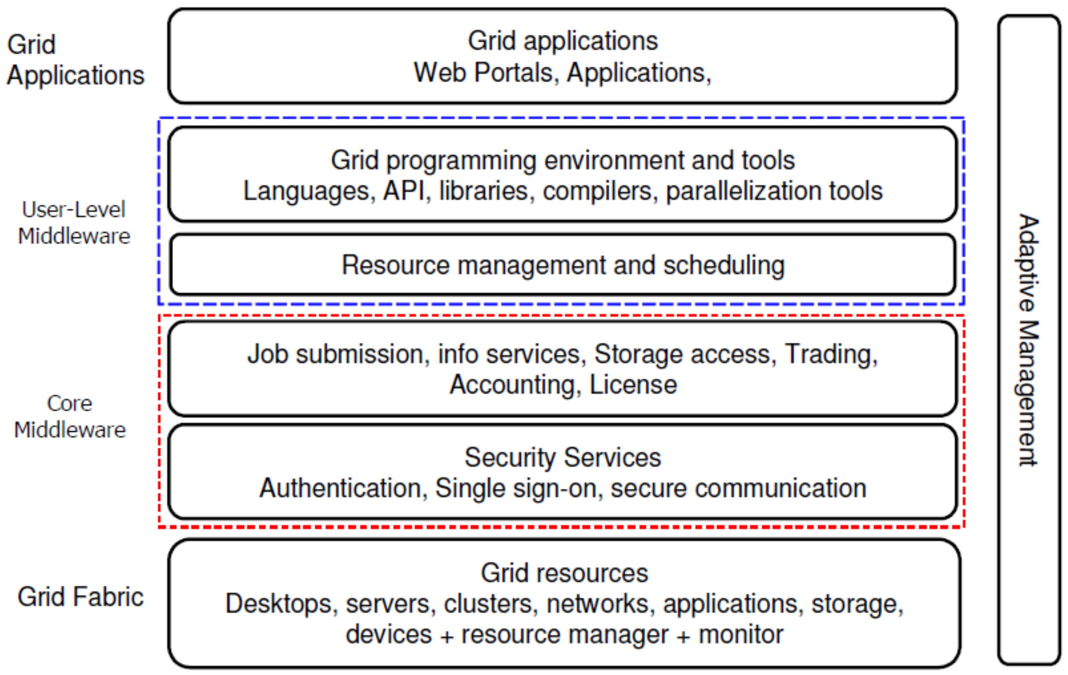

Grid Applications
Grid applications
Web Portals, Applications,
User-Level Middleware
Grid programming environment and tools
Languages, API, libraries, compilers, parallelization tools
Resource management and scheduling
Core Middleware
Job submission, info services, Storage access, Trading, Accounting, License
Security Services
Authentication, Single sign-on, secure communication
Grid Fabric
Grid resources
Desktops, servers, clusters, networks, applications, storage, devices + resource manager + monitor
Adaptive Management

Figure.2.5 Layer of Grid systems [30]

The goal of this dissertation is not to create/develop software but using existing ones to propose some solutions to solve large scale scientific computing. Just as mentioned above, core middleware and user level middleware are two main parts to form Grid system. Core middleware is low level one which is used to charge communicating safely between computing nodes. User level middleware belongs to high level one used to manage end users' interface, mechanism of mapping jobs to computing resources and so on. User level middleware is based on core middleware. In real Grid systems, those two layers are necessary to form Grid middleware. In the following sections, we first introducing communication models used in Grid middleware and then some representative Grid middlewares are surveyed.

2.2.1 Programming models on the Grids

The basis of parallel and distributed computing is the communication layer between the computing units which compose a complex system. Here five kinds of communication models are summarized in grid system.

2.2.1.1 Message passing model

PVM[9] (Parallel Virtual Machine) [31] [32] [33], [34] is a software system that

[9] http://www.netlib.org/pvm3/

enables a collection of heterogeneous computers to be used as a coherent and flexible concurrent computational resource. The individual computers may be shared- or local-memory multiprocessors, vector supercomputers, specialized graphics engines or workstations, which may be interconnected by a variety of networks, such as ethernet, FDDI, etc. It exposes to the application a unified, general, and powerful computational environment for concurrent applications. Users can program using C, C++ or Fortran which incorporate calls to the PVM library routines for facilities such as process initiation, message transmission and reception, and synchronization via barriers or rendezvous. The PVM system transparently handles message routing, data conversion for incompatible architectures, and other tasks that are necessary for operation in a heterogeneous, network environment. Briefly, the principles upon which PVM is based include:

- Explicit message passing model: Concurrent and parallel PVM applications fall under the category of "message passing parallelism". Collections of computational entities, each performing a part of an applications workload using data- functional- or hybrid decomposition, cooperate by exchanging messages to accomplish the overall computational task.
- User configured host pool: The application computational entities execute on a set of machines that are selected by the user for a given run of the PVM program. Both single-CPU machines and hardware multiprocessors (including shared-memory and distributed-memory computers) may be part of the host pool, which may also be altered by adding and deleting machines during operation.
- Translucent access to hardware: Application programs may view the hardware environment either as an attribute-less collection of virtual processing elements or may choose to exploit the capabilities of specific machines in the host pool by positioning certain computational entities on the most appropriate computers.
- Process based computation: The unit of parallelism in PVM is a process, an independent sequential thread of control that alternates between communication and computation. No process to processor mapping is implied or enforced by PVM; in particular, multiple processes may execute on a single processor.
- Heterogeneity support: The PVM system supports heterogeneity in terms of machines, networks, and applications. With regard to message passing, PVM permits strongly typed heterogeneous messages (containing more than one type of data)

The IceT [35] [36] project is to investigate issues surrounding dynamic collaborative resource alliances. A collaborative resource alliance is defined as an agreement between two or more entities to form a computational environment by combining together or removing subsets of resources available to each party. Within collaborative resource environments, resources are joined together in a more usable way. In particular, the following properties are present in any collaborative resource alliance:

- Individuals' applications are made available to all participants, with appropriate levels of access control.
- Designated local resources are made available to all participants. This

utilization of resources would be transparent to the applications and the users. Directed resource/application orchestration is also supported.

- Any application in the combined pool is capable of executing on any of the available resources, without concern for the heterogeneous nature of the resource collection (to the extent possible), and without concern for the program's actual physical location.
- Multiple users would be able to interact concurrently with applications.
- The environment is nomadic or migratory. That is, the environment is able to adopt (merge with) and relinquish (split off) resources as needed, or as directed by the users or the applications.

Consequently, a main focus of the ICET project has been to facilitate the passing of messages, processes, and data amongst various processes, data, users, and resources. In pragmatic terms, ICET aims to provide an infrastructure on which heterogeneous resources may be aggregated, upon these resources data and ICET processes may be passed freely, receiving interaction and being visualized by multiple entities.

MPI (Message Passing Interface) is a specification for a standard library for message passing that was defined by the MPI Forum, a broadly based group of parallel computer vendors, library writers, and applications specialists. Those Goals of MPI are high performance, scalability, and portability. Multiple implementations of MPI have been developed. Targets for those implementations were to include all systems capable of supporting the message-passing model. Two kinds of MPI are MPI-1[38] and MPI-2 [39] [40]. MPI-1 defined an interface for a specific message-passing model of parallel computation, in which a fixed number of processes with disjoint address spaces communicate through a cooperative mechanism (when two processes communicate, one sends and the other receives). MPI provides many types of point-to-point communication, to incorporate requirements for robustness, expressivity, and performance. Messages are strictly typed and scoped, allowing for communication in a heterogeneous environment. MPI also contains an extensive set of collective operations, process topology functions, and a profiling interface. While in MPI-2, processes may create other processes, so that the number of processes in an MPI computation can change dynamically. Processes can interact directly with the memory of other processes. Extensions, semantic modifications, and subset definitions in support of real-time and embedded systems also represent changes to the computational model.

The initial implementation of the MPI 1.x standard was MPICH [41], from Argonne National Laboratory and Mississippi State University. IBM also was an early implementor of the MPI standard, and most supercomputer companies of the early 1990s either commercialized MPICH, or built their own implementation of the MPI 1.x standard. LAM/MPI from Ohio Supercomputing Center was another early open implementation. Argonne National Laboratory has continued developing MPICH for over a decade, and now offers MPICH 2, which is an implementation of the MPI-2.1 standard. Several famous projects based on MPI are described as follows:

- MPICH is a freely available, complete implementation of the MPI specification, designed to be both portable and efficient. The "CH" in MPICH stands for "Chameleon," symbol of adaptability to one's environment and thus of portability. Chameleons are fast, and from the beginning a secondary goal was to give up as little efficiency as possible for the portability.

- LAM (Local Area Multicomputer) [42] is an MPI programming environment and development system for heterogeneous computers on a network. With LAM/MPI, a dedicated computer cluster or an existing network computing infrastructure can act as a single parallel computing resource. LAM/MPI is considered to be "cluster friendly", in that it offers daemon-based process startup/control as well as fast client-to-client message passing protocols. LAM/MPI can use TCP/IP, shared memory, Myrinet (GM), or Infiniband (mVAPI) for message passing.
- The HeteroMPI [43] project provides extensions to the MPI standard for heterogeneous computing. Library assistance is provided to assist in decomposing the problem to best fit the available resources. HeteroMPI still relies on an underlying MPI implementation for process startup and high-performance communication. Our current research in Open MPI seeks to provide the greatest flexibility and performance in precisely the areas where HeteroMPI depends on an underlying MPI implementation for functionality.
- MPICH-G [37] is a complete implementation of the MPI-1 standard and passes the MPICH test suite. Early experiences suggest that it achieves our goal of reducing barriers to the use of distributed computing by allowing the use of MPI as a portable, high-performance programming model for heterogeneous clusters and for wide-area computing systems.
- MPICH-G2 [45] is a Grid-enabled implementation of the popular MPI (Message Passing Interface) framework. MPI is especially useful for problems that can be broken up into several processes running in parallel, with information exchanged between the processes as needed. The MPI programming environment handles the details of starting and shutting down processes, coordinating execution (supporting barriers and other IPC metaphors), and passing data between the processes. Other MPI implementations are dealing with aspects of Grid computing, focusing on the given hierarchies, security aspects as well as user-friendliness. Examples are PACXMPI, Stampi, MetaMPICH or GridMP.

2.2.1.2 RPC and RMI model

RPC and RMI both rely on the same programming paradigm for distributed applications as for centralized applications. In RPC-based or RMI-based system, a remote method (procedure) is transparently invoked (called) across the network, as if it was local. The main difference between them is that client side RPC invokes FUNCTIONS through the proxy function and RMI invokes METHODS through the proxy function. And RMI is an object-oriented version of RPC.

RPC presumes the existence of low-level networking mechanisms (such as TCP/IP and UDP/IP), and upon them it implements a logical client to server communications system designed specifically for the support of network applications. With RPC, the client makes a procedure call to send a data packet to the server. When the packet arrives, the server calls a dispatch routine, performs whatever service is requested, sends back the reply, and the procedure call returns to the client.In order to allow servers to be accessed by differing clients, a number of standardized RPC systems have been created. Most of these use an interface description language (IDL) to allow various platforms to call the RPC.

RPC have been widely used in distributed systems. GridRPC [51] is a standard promoted by the Open Grid Forum, which extends the traditional RPC. GridRPC differs from the traditional RPC in that the programmer does not need to specify the server to execute the task. When the programmer does not specify the server, the middleware system, which implements the GridRPC API, is responsible for finding the remote executing server. When the program runs, each GridRPC call results in the middleware mapping the call to a remote server and then the middleware is responsible for the execution of that task on the mapped server. Another difference is that GridRPC is a stubless model, meaning that client programs do not need to be recompiled when services are changed or added. This facilitates the creation of interfaces from interactive environments like Matlab, Mathematica, and IDL. A number of Grid middleware systems have recently become GridRPC compliant including GridSolve, Ninf-G, DIET, DCE-RPC [46], DFN RPC [47], Peregrine [48], MRPC [49] and RPC-V [50].

RMI (Remote Method Invocation) is a way that a programmer, using the Java programming language and development environment, can write object-oriented programming in which objects on different computers can interact in a distributed network. RMI is the Java version of what is generally known as a remote procedure call (RPC), but with the ability to pass one or more objects along with the request. The object can include information that will change the service that is performed in the remote computer. RMI is implemented as three layers:

- A stub program in the client side of the client/server relationship, and a corresponding skeleton at the server end. The stub appears to the calling program to be the program being called for a service. (Sun uses the term proxy as a synonym for stub.)
- A Remote Reference Layer that can behave differently depending on the parameters passed by the calling program. For example, this layer can determine whether the request is to call a single remote service or multiple remote programs as in a multicast.
- A Transport Connection Layer, which sets up and manages the request.

Java Remote Method Invocation (RMI) enables a programmer to create distributed Java-based applications, in which the methods of remote Java objects can be invoked from other Java virtual machines, possibly on different hosts. RMI inherits basic RPC design in general, it has distinguishing features that reach beyond the basic RPC. With RMI, a program running on one JVM can invoke methods of other objects residing in different JVMs. The main advantages of RMI are that it is truly object-oriented, that it supports all the data types of a Java program, and that it is garbage collected. These features allow for a clear separation between caller and callee. Development and maintenance of distributed systems become easier. Java's RMI provides a high-level programming interface that is well-suited for Grid computing. Some examples are Jini [52] [53], JXTA [54], Ibis [55], Albatorss project [56] and GMI [57].

2.2.1.3 Distributed object model

Distributed object model such as OMG CORBA and DCOM, have been introduced to handle problems inherent in distributed computing on a heterogeneous environment. Distributed object model consists of objects which interact via method invocation, while message-passing model consists of processes which communicate with each other via message-passing. Distributed object model has several benefits over message-passing model. It provides an easy programming environment by supporting transparency of distributed objects, plug and play of software as well as the advantages of object-oriented programming such as reusability, extensibility, and maintainability through abstraction, encapsulation and inheritance. However, it lacks some functionalities for parallel applications, since they are based on client-server model. It does not support group operations, and has some difficulty in implementing efficient parallelisms by using asynchronous communications.

The Common Object Request Broker Architecture (CORBA) is an open distributed object-computing infrastructure being standardised by the Object Management Group (OMG) [OMG]. CORBA automates many common network programming tasks such as object registration, location, and activation; request de-multiplexing; framing and error-handling; parameter marshalling and de-marshalling; and operation dispatching. Although CORBA provides a rich set of services, it does not contain the Grid level allocation and scheduling services found in Globus, however, it is possible to integrate CORBA with the Grid. The OMG has been quick to demonstrate the role of CORBA in the Grid infrastructure. Apart from providing a well-established set of technologies that can be applied to e-Science, CORBA is also a candidate for a higher level conceptual model. It is language-neutral and targeted to provide benefits on the enterprise scale, and is closely associated with the Unified Modelling Language (UML). One of the concerns about CORBA is reflected by the evidence of intranet rather than Internet deployment, indicating difficulty crossing organisational boundaries; e.g. operation through firewalls. Furthermore, real-time and multimedia support was not part of the original design.

Distributed Component Object Model (DCOM) is the distributed version of Microsoft's COM technology which allows the creation and use of binary objects/components from languages other than the one they were originally written in, it currently supports Java(J++),C++, Visual Basic, JScript, and VBScript. DCOM works over the network by using proxy's and stubs. When the client instantiates a component whose registry entry suggests that it resides outside the process space, DCOM creates a wrapper for the component and hands the client a pointer to the wrapper. This wrapper, called a proxy, simply marshals methods calls and routes them across the network. On the other end, DCOM creates another wrapper, called a stub, which unmarshals methods calls and routes them to an instance of the component.

2.2.1.4 Common component model

The Common Component Architecture [59] is a minimal set of standard features that a high performance component framework would need to provide, or can expect, in order to be able to use components developed within different frameworks. Like CORBA it supports component programming, but it is distinguished from other component programming approaches by the emphasis on supporting the abstractions necessary for high-performance programming. The core technologies described in the previous section, Globus or Legion could be used to implement services within a component framework.

The idea of using component frameworks to deal with the complexity of developing interdisciplinary high performance computing applications is becoming increasingly popular. Such systems enable programmers to accelerate project development by introducing higher-level abstractions and allowing code reusability. They also provide clearly defined component interfaces, which facilitate the task of team interaction: such a standard will promote interoperability between components developed by different teams across different institutions. These potential benefits have encouraged research groups within a number of laboratories and universities to develop, and experiment with prototype systems. There is a need for interoperability standards to avoid fragmentation.

2.2.1.5 Service oriented model

Grid technologies are evolving toward an Open Grid Services Architecture (OGSA[10]) [58] in which a grid provides an extensible set of services that virtual organizations can aggregate in various ways. OGSA defines a uniform exposed service semantics (the so called grid service) based on concepts and technologies from both the Grid and Web services communities. OGSA defines standard mechanisms for creating, naming, and discovering transient grid service instances, provides location transparency and multiple protocol bindings for service instances, and supports integration with underlying native platform facilities.

The OGSA effort aims to define a common resource model that is an abstract representation of both real resources, such as nodes, processes, disks, file systems, and logical resources. It provides some common operations and supports multiple underlying resource models representing resources as service instances. OGSA abstractions and services provide building blocks that developers can use to implement a variety of higher-level Grid services, but OGSA services are in

[10] http://www.globus.org

principle programming language- and programming model-neutral. OGSA aims to define the semantics of a grid service instance: how it is created, how it is named, how its lifetime is determined, how to communicate with it, and so on.

OGSA does not, however, address issues of implementation programming model, programming language, implementation tools, or execution environment. OGSA definition and implementation will produce significant effects on grid programming models because these can be used to support and implement OGSA services and higher-level models could incorporate OGSA service model offering high level programming mechanisms to use those services in grid applications. The Globus project is committed to developing an open source OGSA implementation by evolving the current Globus Toolkit towards an OGSA-compliant Globus Toolkit 3.0. This new release will stimulate the research community in developing and implementing OGSA oriented programming models and tools.

Web Services Resource Framework (WSRF[11]) is a kind of grid computing based on web services. WSRF defines conventions for managing 'state' so that applications can reliably share changing information. In combination with WS-Notification and other WS-* standards, the result is to make grid resources accessible within a web services architecture. Coupled with WS-Notification, the specification is a response to, and supersedes, the grid community's own first effort to converge grid and web services, the Open Grid Service Infrastructure (OGSI), which the Global Grid Forum (GGF) and others released in 2003. Announced by the Globus Alliance and IBM (with contributions from HP, SAP, Akamai, Tibco and Sonic) in January 2004, WSRF is due to be implemented in version 4.0 of the open source Globus Toolkit for grid computing, as well as several commercial packages. It consists of several component specifications, including WS-Resource Properties, WS-Resource Life time, WS-Service Group and WS-Base Faults.

2.2.1.6 Hybrid model

The inherent nature of grid computing is to make all manner of hosts available to grid applications. Hence, some applications will want to run both within and across address spaces. That is to say, they will want to run perhaps multithreaded within a shared-address space, and also by passing data and control between machines. Such a situation occurs in clumps (clusters of symmetric multiprocessors) and also in grids. A number of programming models have been developed to address this.

The combination of both OpenMP and MPI within one application to address the clump and grid environment has been considered by many groups [60]. A prime consideration in these application designs is “who’s on top”. OpenMP is essentially a multithreaded programming model. Hence, OpenMP on top of MPI

[11] http://www.globus.org/wsrf/

requires MPI to be thread-safe or requires the application to explicit manage access to the MPI library. MPI on top of OpenMP can requirement additional synchronization and limit the amount of parallelism OpenMP can realize, which approach actually works out best is typically application dependent.

OmniRPC [170] was specifically designed as a threadsafe RPC facility for clusters and grids. OmniRPC uses OpenMP to manage thread-parallel execution while using Globus to manage grid interactions. Rather than using message-passing between machines, however, it provides RPC. OmniRPC is, in fact, a layer on top of Ninf. Hence, it uses the Ninf machinery to discover remote procedure names, associate them with remote executables, and retrieve all stub interface information at run-time. To manage multiple RPCs in a multi-threaded client, OmniRPC maintains a queue of outstanding calls that is managed by a scheduler thread. A calling thread is put on the queue and blocks until the scheduler thread initiates the appropriate remote call and receives the results.

The argument for MPJ [61] is that many applications naturally require the symmetric message-passing model, rather than the asymmetric RPC/RMI model. Hence, MPJ makes multithreading, RMI and message-passing available to the application builder. MPJ message-passing closely follows the MPI-1 specification. This approach, however, does present implementation challenges. Implementation of MPJ on top of a native MPI library provides good performance but breaks the Java security model and does not allow applets. A native implementation of MPJ in Java, however, usually provides slower performance. Additional compilation support may improve overall performance and make this single language approach more feasible.

2.2.1.7 Coordination model

The purpose of a coordination model is to provide a means of integrating a number of possibly heterogeneous components together, by interfacing with each component in such a way that the collective set forms a single application that can execute on parallel and distributed systems [62]. Coordination models can be used to distinguish the computational concerns of a distributed or parallel application from the cooperation ones, allowing separate development but also the eventual fusion of the these two development phases.

The concept of coordination is closely related to those of heterogeneity. Since the coordination interface is separate from the computational one, therefore, the actual programming languages used to write computational code play no important role in setting up the coordination mechanisms. Furthermore, since the coordination component offer a homogeneous way for inter-process communication and abstracts from the architecture details, coordination encourages the use of heterogeneous ensembles of machines.

A coordination language offers composing mechanism and imposes some constraints on the forms of parallelism and on the interfacing mechanisms used to compose an application. Coordination languages for grid computing generally are orthogonal to sequential or parallel code used to implement the single modules that must be executed, but provide a model for composing programs and should implement inter-module optimizations that take into account machine and interconnection features for providing efficient execution on grids. Some recent research activities in this area use XML-based [63] [64] or skeleton-based models for grid programming. Another potential application domain for grid coordination tools is workflow [65], a model of enterprise work management where work units are passed between processing points based on procedural rules.

2.2.3 Grid projects

Grid is a generalized network computing system that is supposed to scale to Internet levels and handle data and computation seamlessly. Designing a Grid architecture that will meet these requirements is challenging due to several issues. Some of these issues are: (1) supporting adaptability, extensibility, and scalability, (2) allowing systems with different administrative policies to inter-operate while preserving site autonomy, (3) co-allocating resources, (4) supporting quality of service, and (5) meeting computational cost constraints. Many Grid projects have been proposed to solve those issues presented above and we will introduce some famous Grid projects in the following parts.

2.2.3.1 Globus

The Globus [66] [67] [68] project is a multi-institutional research effort that seeks to enable the construction of computational grids providing pervasive, dependable, and consistent access to high-performance computational resources, despite geographical distribution of both resources and users. A central element of the Globus system is the Globus Toolkit (GT), which defines the basic services and capabilities required to construct a computational grid. The toolkit consists of a set of components that implement basic services, such as security, resource location, resource management, and communications.

Globus is: A community of users and developers who collaborate on the use and development of open source software, and associated documentation, for distributed computing and resource federation; The software itself, the Globus Toolkit (GT): a set of libraries and programs that address common problems that occur when building distributed system services and applications; The infrastructure that supports this community-code repositories, email lists, problem tracking system, and so forth: all accessible at globus.org.

The software itself provides a variety of components and capabilities, including the following: 1) a set of service implementations focused on infrastructure management. 2) tools for building new web services, in Java, C, and Python. 3) a powerful standards-based security infrastructure. 4) both client APIs (in different languages) and command line programs for accessing these various services and capabilities. 5) detailed documentation on these various components, their interfaces, and how they can be used to build applications.

Globus has evolved from its original first generation incarnation as I-WAY, through version 1 (GT1) to version 2 (GT2). The protocols and services that Globus provided have changed as it has evolved. The emphasis of Globus has moved away from supporting just high performance applications towards more pervasive services that can support virtual organisations. GT4 makes extensive use of Web services mechanisms to define its interfaces and structure its components. Web services provide flexible, extensible, and widely adopted XML-based mechanisms for describing, discovering, and invoking network services; in addition, its document-oriented protocols are well suited to the loosely coupled interactions that many argue are preferable for robust distributed systems. These mechanisms facilitate the development of service-oriented architectures—systems and applications structured as communicating services, in which service interfaces are described, operations invoked, access secured, etc., all in uniform ways.

2.2.3.2 Apples

The AppLeS project [69] [70] [71] focuses on the design and development of Grid-enabled high performance schedulers for distributed applications. The goals of the AppLeS project have been twofold. The first goal has been to investigate adaptive scheduling for Grid computing. The second goal has been to apply research results to applications for validating the efficacy of our approach and, ultimately, extracting Grid performance for the end-user. The first generation of AppLeS schedulers demonstrated that simultaneously taking into account application- and system level information makes it possible to effectively schedule applications onto computational environments as heterogeneous and dynamic as the Grid. However, each scheduler was embedded within the application itself and thus difficult to re-use for other applications. The next logical step was to consider classes of applications that are structurally similar and to develop independent software frameworks that can schedule applications within a class.

AppLeS agents use application and system information to select a viable set of resources. AppLeS uses the services of other RMSs such as Globus, Legion, and NetSolve to execute application tasks. Applications have embedded AppLeS agents that performing resource scheduling on the Grid. AppLeS has been used in several applications areas including gene sequence comparison, satellite radar images visualization, and tomography. The AppLeS framework contains templates that can be applied to applications that are structurally similar and have the same computational model. The templates allow the reuse of the application specific schedulers in the AppLeS agents. Templates have been developed for parametric and master slave type of applications.

The focus of AppLeS is on scheduling and thus it follows the resource management model supported by the underlying Grid middleware systems. An AppLeS scheduler is central to the application that performs mapping of jobs to resources, but the local resource schedulers perform the actual execution of application units. AppLeS schedulers do not offer QoS support. AppLeS can be classified with a predictive heuristic state estimation model, online rescheduling and fixed application oriented scheduling policy.

2.2.3.3 Legion

Legion [72] [73] [74] is an object-based meta-system developed at the University of Virginia. Legion provides the software infrastructure so that a system of heterogeneous, geographically distributed, high performance machines can interact seamlessly. Legion attempts to provide users, at their workstations, with a single, coherent, virtual machine. In the Legion system: 1) everything is an object – Objects represent all hardware and software components. Each object is an active process that responds to method invocations from other objects within the system. Legion defines an API for object interaction, but not the programming language or communication protocol; 2) classes manage their instances – every Legion object is defined and managed by its own active class object. Class objects are given system-level capabilities; they can create new instances, schedule them for execution, activate or deactivate an object, as well as provide state information to client objects; 3) users can define their own classes – As in other object-oriented systems users can override or redefine the functionality of a class. This feature allows functionality to be added or removed to meet a user's needs.

Legion core objects support the basic services needed by the metasystem. The Legion system supports the following set of core object types: 1) Classes and Metaclasses; 2) Host objects; 3) Vault objects; 4) Implementation Objects and Caches; 5) Binding Agents; 6) Context objects and Context spaces. Legion objects are independent, active, and capable of communicating with each other via unordered non-blocking calls. Like other object-oriented systems, the set of methods of an object describes its interface. The Legion interfaces are described in an Interface Definition Language (IDL). As the Legion system uses object-oriented approach, which potentially make it ideal for designing and implementing a complex distributed computing environments. However, using an object-oriented methodology does not come without a raft of problems, many of these is tied-up with the need for Legion to interact with legacy applications and services.

2.2.3.4 Netsolve/Gridsolve

Netsolve [75] [76] is a client–agent–server paradigm based network enabled application server. It is designed to solve computational science problems in a distributed environment. The Netsolve system integrates network resources including hardware and computational software packages into a desktop application. Netsolve clients can be written in C, FORTRAN, Matlab or Web pages to interact with the server. A Netsolve server can use any scientific package to provide its computational software. Communications between Netsolve clients, agents, and servers are performed using TCP/IP sockets. Netsolve agents can search for resources on a network, choose the best one available, execute the client request, and then return the answer to the user.

The Netsolve system is a computational Grid with hierarchical based machine organization. Netsolve agents maintain information about resources available in the network. The Netsolve servers which are the resources in a Netsolve Grid are responsible for making their existence aware to Netsolve Agents and thus use a push protocol for resource dissemination. Netsolve Agents also perform resource discovery and scheduling. An agent may request assistance of other Agents in identifying the best resources and scheduling. Thus Netsolve can be considered to use a network directory implemented by the Netsolve agents, decentralized scheduling with a fixed application oriented scheduling policy.

GridSolve [77] is a stubless RPC-based client-agent-server system for remotely accessing hardware and software resources. GridSolve emphasizes ease-of-use for the user and includes resource monitoring, scheduling and service-level fault-tolerance. In addition to providing Fortran and C clients, GridSolve enables SCEs (such as Matlab) to be used as clients, so domain scientists can use Grid resources from within their preferred environments. GridSolve is a more highly evolved version of the earlier NetSolve project, and it is based on the emerging GridRPC standard.

2.2.3.5 Diet

DIET [78 [79] (Distributed Interactive Engineering Toolbox) project is to develop a set of tools to build, deploy, and execute computational server daemons. It focuses on the development of scalable Middleware with initial efforts concentrated on distributing the scheduling problem across multiple agents. DIET consists of a set of elements that can be used together to build applications using the GridRPC paradigm. This Middleware is able to find an appropriate server according to the information given in the client’s request (e.g. problem to be solved, size of the data involved), the performance of the target platform (e.g. server load, available memory, communication performance) and the local availability of data stored during previous computations. The scheduler is distributed using several collaborating hierarchies connected either statically or dynamically (in a peer-to-peer fashion). Data management is provided to allow persistent data to stay within the system for future re-use. This feature avoids unnecessary communications when dependencies exist between different requests. The component of DIET Dashboard can provide the users with a complete, modular, portable, and powerful way to manage grid resources of the applications that run on it.

The DIET framework comprises several components. A Client is an application that uses the DIET infrastructure to solve problems using an RPC approach. Clients access DIET through various interfaces: web portals or programs using C, C++, or Java APIs. A SeD, or server daemon, acts as the service provider, exporting a functionality through a standardized computational service interface. A single SeD can offer any number of computational services (depending on the capacity of the machine). A SeD can also serve as the interface and execution mechanism for either a stand-alone interactive machine or a parallel supercomputer (or cluster) using an interface with a batch scheduler. Agents which are the third component of the DIET, can help to facilitate the service location and invocation interactions between clients and SeDs. Collectively, a hierarchy of agents provides higher-level services such as scheduling and data management. These services are made scalable by distributing them across a hierarchy of agents composed of a single Master Agent and several Local Agents. In DIET system, the scheduler/agent is scattered across a hierarchy of Local Agents and Master Agents. The motivation for this architecture was that it was more scalable and solved the problem of bottlenecks in a centralised agent/scheduler when many clients try to access several servers. In addition, the DIET system employed direct communication between servers and data persistency. Where a dependency existed between tasks, this output would remain on the source server. When the destination task is called for execution, this data would be pulled from the source server. If the source server is the same as the destination server this output would be stored and retrieved locally (data persistency).

2.2.3.6 Ninf/Ninf-G

Ninf [81] is a client–server based network infrastructure for global computing. The Ninf system functionality is similar to NetSolve. Ninf allows access to multiple remote compute and database servers. Ninf clients can semi-transparently access remote computational resources from languages such as C and FORTRAN. Programmers can build a global computing application by using the Ninf remote libraries as its components, without being aware of the complexities of the underlying system they are programming. Procedural parameters, including arrays, are marshaled and sent to the Ninf server on a remote host responsible for executing the requested library functions and returning the results. The Ninf client library calls can be synchronous or asynchronous in nature.

Ninf-G [80] is a Grid-enabled RPC (remote procedure call) implementation that allows existing libraries to be used as RPC calls rather than local calls. Ninf-G can interpose itself between the library and the calling application, allowing the library code to be run on a remote system, potentially in parallel with other systems, while the application runs locally. Neither the library nor the calling application need to be changed. Ninf-G can even be used with precompiled libraries for which the source code is not available. Ninf-G is most useful in cases where the library provides a computationally-intensive service where the application would benefit if the computation were performed on a remote system. Useful scenarios include: when several computations are needed and can be performed in parallel, or when the user interface allows the user to do other things while the computation is performed. In both of these cases, the calling application would be modified to invoke the library call (using Ninf-G) in a separate process or thread so that the application can proceed while the library call is executed remotely.

Ninf-G2 [82] is a reference implementation of the GridRPC API, a proposed GGF standard. Ninf-G2 aims to support development and execution of Grid applications which will run efficiently on a large-scale computational Grid. Here, we use the term "large-scale computational Grid" as a cluster of about ten geographically distributed cluster systems, each of which consists of tens to hundreds of processors.

2.2.3.7 SmartGridSolve

SmartGridRPC [83] [84] is an extension of the GridRPC model, which aims to achieve higher performance. As well known to us all, 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, the GridRPC model can only support 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. Another important aspect of the GridRPC model is its communication model. The communication model of GridRPC is based on the client-server model or star network topology. This means that tasks can be executed on any of the servers and inputs/outputs can only traverse the client-server links.

SmartGridRPC model has extended the GridRPC model to support collective mapping of a group of tasks by separating the mapping of tasks from their execution. This allows the group of tasks to be mapped collectively and then executed collectively. In addition the traditional client-server model of GridRPC has been extended so that the group of tasks can be collectively executed on a network topology which is fully connected. This is a network topology where all servers can communicate directly or servers can cache their outputs locally.

Key benefits of SmartGridRPC can be described as follows through direct communication between clients: 1) Improved balancing of computation load; 2) Reduced volume of communication; 3) Improved balancing of communication load; 4) Increased parallelism of communication; 5) Reduced memory usage and paging.

2.2.3.8 DataCutter

DataCutter [85] [86] is a middleware infrastructure that enables processing of scientific datasets stored in archival storage systems across a wide-area network. DataCutter provides support for subsetting of datasets through multi-dimensional range queries, and application specific aggregation on scientific datasets stored in an archival storage system.

DataCutter provides a core set of services, on top of which application developers can implement more application-specific services or combine with existing Grid services such as metadata management, resource management, and authentication services. The main design objective in DataCutter is to extend and apply features of the Active Data Repository (ADR), namely support for accessing subsets of datasets via range queries and user-defined filtering operations, for very large datasets in a shared distributed computing environment. In ADR, data processing is performed where the data is stored (i.e. at the data server). To make efficient use of distributed shared resources within the DataCutter framework, the application processing structure is decomposed into a set of processes, called *filter*s. DataCutter uses these distributed processes to carry out a rich set of queries and application specific data transformations. Filters can execute anywhere (e.g., on computational farms), but are intended to run on a machine close (in terms of network connectivity) to the archival storage server or within a proxy server.

DataCutter also provides common support for subsetting very large datasets through multi-dimensional range queries. Very large datasets may result in a large set of large data files, and thus a large space to index. A single index for such a dataset could be very large and expensive to query and manipulate. To ensure scalability, DataCutter uses a multi-level hierarchical indexing scheme.

2.2.3.9 DAGMan

Directed Acyclic Graph Manager (DAGMan) [90] [91] is a meta-scheduler for the execution of programs (computations). DAGMan submits the programs to Condor in an order represented by a DAG and processes the results. A DAG input file describes the DAG, and further submit description file(s) are used by DAGMan when submitting programs to run under Condor. DAGMan is itself executed as a scheduler universe job within Condor. As DAGMan submits programs, it monitors log file(s) to enforce the ordering required within the DAG. DAGMan is also responsible for scheduling, recovery, and reporting on the set of programs submitted to Condor.

DAGMan allows the expression of dependencies. Each job is a node of the graph and the edges identify their dependencies. Each job can have any number of "parent" or "children" nodes. Childrens are only executed once their parents have completed their execution. DAGMan does not allow cycles in the graph to prevent dead-locks. Data migration is not automated by DAGMan and must be explicitly defined by the user. The mapping of the workflow to the computing resource is done during the execution of the workflow by the condor job scheduler.

2.2.3.10 Kepler

Kepler [92] [93] is an open-source scientific workflow engine with contributors from a range of application-oriented research projects. Kepler built upon the Ptolemy II system based at the University of California at Berkeley, is a dataflow oriented mature workflow architecture. The Kepler system aims at supporting very different kinds of workflows, ranging from low-level 'plumbing' workflows of interest to Grid engineers, to analytical knowledge discovery workflows for scientists, and conceptual-level design workflows that might become executable only as a result of subsequent refinement steps .Kepler is used in several large Grid projects where the management of biological data analysis workflows is critical. The approach Kepler takes is based on an actor-oriented model which allows hierarchical modeling and dataflow semantics. The Kepler tools support a well-designed graphical composition interface that is very intuitive and easy to use. To support the interaction with web services Kepler uses a form of actor proxy for each web services that is invoked. In addition they have created a set of Grid actors for doing GridFTP file management and Globus GRAM execution.

Using Kepler, scientists can capture workflows in a format that can easily be exchanged, archived, versioned, and executed. Both Kepler's intuitive GUI (inherited from Ptolemy) for design and execution, and its actor-oriented modeling paradigm make it a very versatile tool for SWF design, prototyping, execution, and reuse for both workflow engineers and end users. Kepler workflows can be exchanged in XML using Ptolemy's own Modeling Markup Language (MoML). Kepler actors run as local Java threads by default (from Ptolemy), but are extended to spawn distributed execution threads via web and Grid services, as well as through Java's foreign language interface (Java Native Interface). Kepler currently provides the following features: Prototyping workflows; Distributed Execution (Web and Grid-Services); Database Access and Querying; Kepler includes a suite of data transformation actors (XSLT, XQuery, Perl, etc.); Supporting foreign language interfaces (Matlab actor, Python actor, etc) via the Java Native Interface (JNI).

2.3 Desktop Grid Computing

2.3.1 Introduction

The world's computing power and disk space is no longer primarily concentrated in supercomputer centers and machine rooms, but is distributed in hundreds of millions of personal computers belonging to the general public. Desktop Grids use these resources to do scientific computing. The number of Internet-connected PCs is indeed growing rapidly, and is projected to reach 1 billion by 2015. Together, these PCs could provide many Petaflops of computing power. The public resource approach applies to storage as well as computing. If 100 million computer users each provide 10 Gigabytes of storage, the total would exceed the capacity of any centralized storage system.

A Desktop Grid computing environment mainly consists of client, volunteer, and server. A client is a parallel job submitter. A volunteer is a resource provider that donates its computing resources when idle. A server is a central manager that controls submitted jobs and volunteers. A client submits a parallel job to a server. A job is divided into sub-jobs that have their own specific input data. The sub-job is called a task. The server allocates tasks to volunteers using scheduling mechanisms. Each volunteer executes its task when idle, while continuously requesting data from its server. When each volunteer subsequently finishes its task, it returns the result of the task to the server. Finally, when the server collects all results of tasks from volunteers, it returns the final result of the job back to the client.

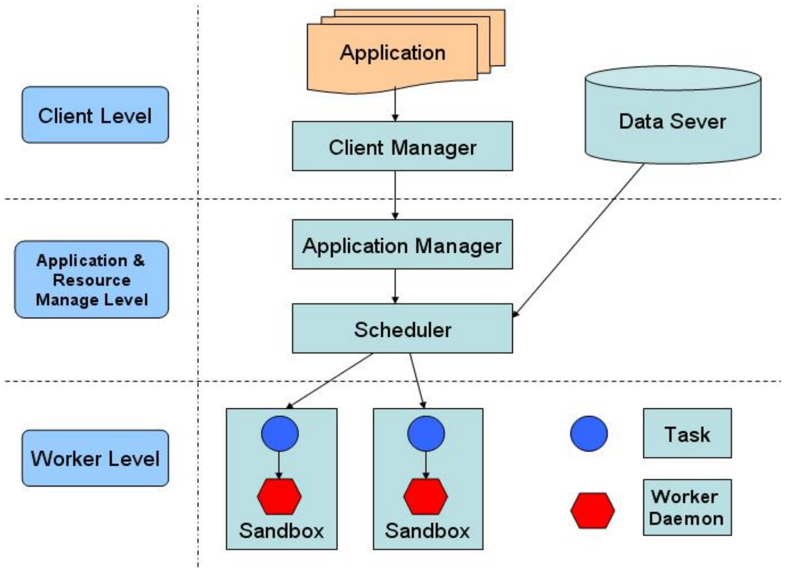


Figure.2.6 Common architecture of Desktop Grid systems

Just as mentioned above, desktop grid systems can harvest the idle cycles of PC's in Internet environments and/or enterprise environments. These systems share many features of architectural design and organization, and we give an overview of the anatomy of those kinds of systems, identifying commonalities and important differences at the client, application and resource management, and worker levels (Note that the physical organization may be different than what is shown in Figure 2.6. For example, components of the client level often reside on the same host as the worker.) At the Client Level, a user submits an application to a desktop grid, using tools for controlling the application's execution and monitoring its status. At the Application and Resource Management Level, the application is then scheduled on workers and appropriate data will be sent to workers. At the Worker Level, the worker ensures the application's task executes transparently with respect to other user processes on the hosts.

The ideal desktop grid system would have the following characteristics:

- Scalability: The throughput of the system should increase proportionally with the number of resources and the computing resources can be joined in or left from the desktop grid platform automatically without influencing the performance of the platform.
- Fault tolerance: The system must be tolerant of both server failure and worker failure. Traditionally, the term failure refers to a defect of hardware or software. We use the term failure broadly to include all causes of task failure, including not only failure of the host's hardware or worker software, but also keyboard/mouse activity that causes the worker to kill a running task.
- Security: The machine including its data, hardware, and processes must be protected from a misbehaving desktop grid application. Conversely, the application's executable, input, and output data, which may be proprietary, must be protected from user inspection and corruption.
- Easy to manage: To use public computing resources, client program should be easy to install and manage. Systems should provide tools for installing and updating workers very easily, and also tools for managing applications and resources, and monitoring their progress.
- Unobtrusiveness: Since the desktop grid application shares the system with the user, the user processes must have priority over the client's. When the worker detects user activity, the task should be suspended temporarily until the activity subsides, or the task should be killed and restarted later when the host becomes available again.
- Usability: Integration of an application within a desktop grid system should be as transparent as possible; in many cases, the complexity of the (legacy) program or the fact that the source code is proprietary and is not available makes it difficult to modify the code to use a desktop grid system.

Some challenges have to be mentioned when building real desktop grid systems and those challenges can be summarized as follows:

- Volatility (non-dedication): Since the computing resources of Desktop Grid are mainly from personal computers, it should respect the autonomy of resource providers. I.e., volunteers can leave arbitrarily even during the process of public execution, and they are allowed to execute private execution at any time, thus causing interrupting the public execution. Accordingly, they have various volunteering time (that is, the time of donation and regularity of utilization from users). The various occurrence rate and form of volatility directly affect the execution of tasks. So the scheduler must take volatility into account in order to provide good performance and reliable computation.
- Dynamic environment: Resource's owners can configure its preference and

can control its machine in Desktop Grid. They can freely join and leave in the middle of the executions without any constraints. Thus, the state of system (that is, load, availability, volatility, latency, bandwidth, trust, etc.) is continuously changing over time during the public execution. A scheduler should adapt to such a dynamic environment.

- Lack of trust: In Desktop Grid, anonymous nodes can participate as a resource provider. Some malicious volunteers tamper with the computation and then return corrupted results. The desktop grid system should guarantee the correctness of results.
- Failure: In Desktop Grid, volunteers are connected through Internet/low speed network, so they are exposed to crash and link failures. In addition, since volunteers are not dedicated to public execution and freely leave during public execution, the execution is delayed, blocked, and even lost. The desktop grid system should tolerate the failures and volatility.
- Heterogeneity: Desktop Grid is based on volunteer computers at the edge of Internet. Volunteers have heterogeneous properties each volunteer has a various occurrence rate of failures and volatility, availability, and trust according to its execution behavior. The heterogeneity, which can cause delaying the overall completion time, makes scheduling decision more difficult.
- Scalability: Scalability is character of Desktop Grid system by nature. The Desktop Grid system must adapt to this feature. This is to say, the scheduler must allocate tasks to appropriate computing resources according to theirs properties such as availability, stability and trust.
- Participants: In Desktop Grid, resource providers are mainly voluntary participants without any reward for their donation of resources. In order to encourage resource providers to reliably and eagerly donate their resources for a long time, the Desktop Grid system may consider reputation and incentive mechanisms.

In Desktop grid system, the resources are often ***volatile*** due to user activity, machine hardware failures, and network failures. The high volatility of Desktop Grid system makes the differences between Grid and Desktop Grid in programming method, scheduling mechanism, resources management and so on. Next, we will present some general programming methods on Desktop Grid systems.

2.3.2 Programming method on Desktop Grids

A key mechanism in Desktop Grid is the volatility tolerance. Every Desktop Grid platform uses resources that are not fully manageable by the Desktop Grid administrator. A very common situation is the resource owner disconnecting its resource from the Desktop Grid without prior notice. Volatility tolerance systems are also used to cope with the unavoidable failures appearing in Desktop Grid: resource hardware of software failures, networking Failures. Like cycle stealing, volatility tolerance is the key problem of designing a Desktop Grid system. Two kinds of programming methods are proposed to deal with specific situation in Desktop Grid.

2.3.2.1 Volatility based message passing model

MPI is the most widely used programming environment for high performance applications. MPI in its specification and most deployed implementations follows the fail stop semantic (specification and implementations do not provide mechanisms for fault detection and recovery). Thus, MPI applications running on a large cluster may be stopped at any time during their execution due to an unpredictable failure. The MPICH-V [12] project focuses on designing, implementing and comparing several automatic fault tolerance protocols for MPI applications.

MPICH-V [94] [95] environment encompasses a communication library based on MPICH and a runtime environment. The MPICH-V library can be linked with any existing MPI program as usual MPI libraries. MPICH-V runtime is a complex environment involving several entities: Dispatcher, Channel memories, Checkpoint servers, and computing/communicating nodes

MPICH-V is a MPI fault tolerant implementation based on uncoordinated checkpointing and distributed pessimistic message logging. MPICH-V relies on the MPICH-V runtime to provide automatic and transparent fault tolerance for parallel applications on large scale parallel and distributed systems with volatiles nodes. By reusing a standard MPI implementation and keeping it unchanged, MPICH-V allows to execute any existing MPI application and only requires to relink them with the MPICH-V library. MPICH-V architecture gathers several concepts: Channel Memory for implementing message relay and repository, Checkpoint Servers for storing remotely the context of MPI processes.

Currently, MPICH-V proposes 7 protocols: MPICH-V1, MPICH-V2, MPICH-Vcl, MPICH-Pcl and 3 algorithms for MPICH-Vcausal. MPICH-V1 implements an original fault tolerant protocol specifically developed for Desktop Grids relying on uncoordinated checkpoint and remote pessimistic message logging. It uses reliable nodes called Channel Memories to store all in transit messages. MPICH-V2 is designed for homogeneous networks like clusters where the number of reliable component assumed by MPICH-V1 is too high. It reduces the fault tolerance overhead and increases the tolerance to node volatility. This is achieved by implementing a new protocol splitting the message logging into message payload logging and event logging. These two elements are stored separately on the sender node for the message payload and on a reliable event logger for the message events. The third protocol, called MPICH-Vcl, is derived from the Chandy-Lamport global snapshot algorithm. It implements coordinated checkpoint without message logging. This protocol exhibits less overhead than MPICH-V2 for clusters with low fault frequencies. MPICH-Pcl is a blocking

[12] http://mpich-v.lri.fr/

implementation of Chandy-Lamport algorithm. It consists in exchanging messages for emptying every communication channel before checkpointing all processes. MPICH-Vcausal concludes the set of message logging protocols, implementing a causal logging. It provides less synchrony than the pessimistic logging protocols, allowing messages to influence the system before the sender can be sure that non deterministic events are logged, to the cost of appending some information to every communication.

2.3.2.2 Fault tolerance based RMI/RPC model

The concept of Remote Procedure Call (RPC) has been used for a long time in distributed computing as it provides a simple way to allow communication between distributed components. Most of the previous works have focused on the development of high performance RPC mechanisms and RPC/RMI for the Grid. GridRPC is a proposal to standardize a RPC/RMI mechanism for Grid computing, but it does not encompass fault tolerance mechanisms. Ninf and Ninf-G employ a client-server model but do not provide fault detection nor recovery. NetSolve uses a client-agent-server paradigm and provides two levels of fault tolerance for the servers: a) inter-server fault tolerance moves the computation to another server when the failure of a server is detected by an agent; b) usual techniques of coordinated checkpointing and rollback recovery ensure intra-server fault tolerance. Agent and client fault tolerance is not supported.

The characteristics of Internet connected Desktop Grids derive from the combination of best-effort networks and infrastructures gathering a large number of weakly controlled and volatile computing nodes. *Volatility*, The size of large scale computing infrastructure makes the node disconnection probability not a rare event; *no stable component*, the unpredictability of component volatility precludes to considering the existence of stable components in the system; *intermittent crashes*, components may leave the system for any period of time without prior notification. *connection-less interactions*. In Internet connected Desktop Grids, it is likely that many clients will be connected to a server at a given time. Reciprocally, it is likely that many servers will be connected to a client launching a large number of non blocking RPCs and no communication exists between clients.

Some global computing platforms like SETI and BOINC use process replication to recover from computing resources crash. In replication, the same job is running on several hosts and an atomic broadcast based protocol ensures consistency of replicas. When a failure hits some processes, the recovery procedure is to rebuild new replicas. Because of the voluntary social model of the global computing platform, overall available computing power far oversize the application needs and replication comes at no cost. So multi-replications for a task are possible and reasonable.

RPC-V [96] [97] is an automatic and transparent fault-tolerant environment for RPC programming on Internet connected Desktop Grids. The main features of

RPC-V (API, virtualization, basic fault management policies, scheduling and data communication modes) have been designed according to the context of large scale Grids. RPC-V design follows a new fault tolerant architecture based on 1) a three-tier architecture (clients, Coordinator, servers) 2) message logging on all system components 3) fault detector on all components and 4) passive replication of the coordinators. RPC-V can be integrated into some desktop grid middleware, such as Xtremweb.

2.3.3 Desktop Grid projects

To know the Desktop Grid system in detail, some representative Desktop Grid projects will be presented in this section.

2.3.3.1 Boinc

BOINC[13] [98] [99] (Berkeley Open Infrastructure for Network Computing) is an open source platform for Desktop Grid computing. BOINC is being developed at U.C. Berkeley Spaces Sciences Laboratory by the group that developed and continues to operate SETI@home. Other projects such as Predictor@Home, Folding@Home, Climatepredication.net, Climate@Home, LHC@Home and Einstein@Home are based on BOINC.

BOINC consists of server and client. A server has a task server that dispatches tasks and collect the results of tasks, a data server that handles file transfer, a database that stores descriptions of applications, volunteers, scheduling, etc., and web interfaces for account management, message boards, etc. A client (that is, worker) runs projects' applications. It can participate in several projects. BOINC is mainly based on voluntary participants connected through Internet .When a client sends a request to a task server (pull mode based), the server allocates a list of new tasks to the client. Additionally, BOINC supports locality scheduling. In addition, a client performs local scheduling on its computer, which decides which task to run among multiple projects' applications, when to ask a server for more tasks, which project to ask, and how much tasks to ask for. BOINC clients can join and leave freely, so scheduling in BOINC should be dynamic.

The applications suiting for BOINC are mainly compute-intensive and independent. BOINC scheduler distributes a task to a worker only if the client is likely to complete the task by its deadline. If a task's deadline passes or if a task fails, a server marks it as time-out and redistributes a new instance of the task. With the local scheduling, BOINC client may preempt applications either by suspending them or by instructing them to quit if it participates in multiple projects. That is, one task is preempted by another task. BOINC also provides checkpoint API for applications.

[13] http://boinc.berkeley.edu/

To prevent erroneous computational results, BOINC uses a technique called redundant computing. In particular, a project can specify that N results should be created for each task. Once M (M is less than or equal to N) of these have been distributed and completed, an application-specific function is called to compare the results and possibly select a canonical result. If no consensus is found, or if results fail, BOINC creates new results for the task, and continues this process until either a maximum result count or a timeout limit is reached.

As presented above, scalability is a problem that must be consider in Desktop Grid computing projects. Specially, we may have millions of participants and a relatively modest server complex. As a matter of fact, if all the participants simultaneously try to connect to the server, a disastrous overload condition will happen. BOINC has a number of mechanisms to prevent this.

2.3.3.2 Entropia

Entropia [100] [101] is a middleware for commercial Desktop Grid. The Entropia system architecture consisted of three layers: physical node management, scheduling, and job management. The physical node management layer, provided basic communication and naming, security, resource management, and application control. The second layer was resource scheduling, providing resource matching, scheduling, and fault tolerance. Users could interact directly with the resource scheduling layer through the available APIs or alternatively through the third layer management, which provides management facilities for handling large numbers of computations and files. Entropia provided a job management system, but existing job management systems can also be used. The three-layer architecture of Entropia can provide end users with a wealth of benefits in system capability, ease of use by users and IT administrators, and internal implementation.

The key advantages of the Entropia system are the ease of application integration and a new model for providing security and unobtrusiveness for the application and client machine. The approach is to automatically wrap an application in a virtual machine technology. When an application program is registered or submitted to the Entropia system, it is automatically wrapped inside the virtual machine. This isolation is called sandboxing. The application is contained within a sandbox and is not allowed to modify resources outside the sandbox. The application is fully unaware of being running within a sandbox, since its interaction with the OS is automatically controlled by the virtual machine. The virtual machine intercepts system calls the application makes. This ensures that the virtual machine has complete control over the applications. The Entropia system also can aggregate the raw desktop resources into a single logical resource. This logical resource is reliable, secure and predictable despite the fact that the underlying raw resources are unreliable, insecure and unpredictable. This logical resource provides high performance for applications through parallelism while always respecting the desktop user and his or her use of the desktop machine. Furthermore, this logical resource can be managed from a single administrative console. Addition or removal of desktop machines from the Entropia system is easily achieved, providing a simple mechanism to scale the system as the organization grows or as the need for computational cycles grows.

The disadvantage of the Entropia system was that it did not support heterogeneous systems. The only platform was Windows that limited the usability of this system in a research environment.

2.3.3.3 Condor

Condor [104] [103] [102] [89], developed at the department of Computer Science, University of Wisconsin, Madison, is a High Throughput Computing (HTC) environment that can manage very large collections of distributive owned workstations.The Condor environment provides a powerful resource management by match-making resource owners with resource consumers. Condor provides a job queuing mechanism, scheduling policy, priority scheme, resource monitoring, and resource management. Users submit their serial or parallel jobs to Condor, Condor places them into a queue, chooses when and where to run the jobs based upon a policy, carefully monitors their progress, and ultimately informs the user upon completion.

Condor implements ClassAds to simplify the user's submission of jobs. ClassAds work in a fashion similar to the newspaper classified advertising want-ads. All machines in the Condor pool advertise their resource properties, both static and dynamic, such as available RAM memory, CPU type, CPU speed, virtual memory size, physical location, and current load average, in a resource offer ad. A user specifies a resource request ad when submitting a job. The request defines both the required and a desired set of properties of the resource to run the job. Condor acts as a broker by matching and ranking resource offer ads with resource request ads, making certain that all requirements in both ads are satisfied. During this match-making process, Condor also considers several layers of priority values: the priority the user assigned to the resource request ad, the priority of the user which submitted the ad, and desire of machines in the pool to accept certain types of ads over others.

Condor also provides DAGMan for executing dependable jobs. Condor enables preemptively resume scheduling on dedicated compute cluster resources. It can preempt a low-priority task in order to immediately start a high-priority task.

2.2.3.4 BitDew

BitDew [87] [88] framework is a programmable environment for management and distribution of data for Desktop Grid. It aims to break the "data wall" by adopting the key P2P technologies (DHT, BitTorrent). BitDew can offer programmers a simple API for creating, accessing, storing and moving data with an easy way, even on highly dynamic and volatile environments. The BitDew programming model relies on 5 abstractions to manage the data : *1) replication* indicates how many occurrences of a data should be available at the same time on the network, *2) fault-tolerance* controls the policy in presence of machine crash, *3) lifetime* is an attribute absolute or relative to the existence of other data, which decides the life cycle of a data in the system, *4) affinity* drives movement of data according to dependency rules, *5) protocol* gives the runtime environment hints about the protocol to distribute the data (http, ftp or bittorrent).

The BitDew runtime environment is a flexible environment implementing the API. It relies both on centralized and distributed protocols for indexing, storage and transfers data. The architecture follows a classical three-tiers schema commonly found in Desktop Grids: it divides the world in two sets of nodes: stable nodes and volatile nodes. Stable nodes run various independent services which compose the runtime environment: Data Repository, Data Catalog, Data Transfer and Data Scheduler. We call these nodes the *service hosts*. Volatile nodes can either ask for storage resources or offer their local storage. Usually, programmers will not use directly the various D* services; instead they will use the API which in turn hides the complexity of internal protocols. The Bitdew runtime environment delegates a large number of operation to third party components : 1) Meta-data information are serialized using a traditional SQL database, 2) data transfer are realized out-of-band by specialized file transfer protocols and 3) publish and look-up of data replica is enabled by the means of DHT protocols.

2.3.3.5 Javalin

Javelin [105] [106] [107] is a Java-based infrastructure for parallel Internet computing. Applications run as Java applet or screen saver. Applications are mainly compute-intensive and independent. Javelin consists of three entities: broker, client, and host (volunteer in this thesis). A client registers its tasks to a broker, a host offers computing resources and a broker coordinates the supply and demand for computing resources. When a host contacts a broker, the broker adds the host to a logical tree structure. A broker maintains the organized tree of hosts.

A client registers with a broker. If a host requests tasks to the broker, the broker informs the host of client ID and application information. Then the host executes tasks. At this time, work stealing and advanced eager scheduling are performed. With worker stealing, when a host finish a task, it request tasks from other host in two ways: deterministic or probabilistic approaches. In a deterministic approach, a host asks tasks from its children or its parents on the basis of tree structure. In a probabilistic approach, the host selects the target randomly from the list of hosts it currently knows. With advanced eager scheduling, the client selects the next task marked undone and reissues it to another host. The advanced eager scheduling is invoked only when worker stealing fails. It also provides fault tolerant mechanism, that is, how to fix tree in the presence of host's failure. The failed work is redistributed by eager scheduling, in the sense that eager scheduling guarantees that the undone works will be rescheduled to different hosts eventually. In Javelin, the work stealing is performed at a host, and eager scheduling is performed at a client. A broker is a simple mediator between clients and hosts.

Javelin 2.0 [108] is a branch-and-bound computational model through extending javelin and it facilitates aggregating larger sets of host processors through improving the following aspects: a branch-and-bound computational model, a scalable task scheduler using distributed work stealing, a distributed eager scheduler implementing fault tolerance. Javelin 2.0 frees application developers from concerns about complex inter-processor communication and fault tolerance among Internet worked hosts. When all or part of their application can be cast as a piecework or a branch-and-bound computation, Javelin 2.0 allows developers to focus on the underlying application.

2.3.3.6 Webcom

WebFlow [109] [110] is a computational extension of the Web model that can act as a framework for wide-area distributed computing. The main design goal of WebFlow was to build a seamless framework for publishing and reusing computational modules on the Web, so that end users, via a web browser, can engage in composing distributed applications using WebFlow modules as visual components and editors as visual authoring tools.

WebCom uses a variant of the client/server paradigm to distribute instructions for execution over the web. These instructions are arbitrarily complex and are specified in the application programming environment. When the Condensed Graphs instruction constructor is used in conjunction with WebCom, instructions are typically composed of both sequential programs (also called atomic instructions) and Condensed nodes encapsulating graphs of interacting sequential programs. In effect, a Condensed Graph on WebCom represents a hierarchical job control and specification language. Execution begins on a single server (also called a master) and instructions are distributed to clients (which are known as slaves or potential masters, depending on the type of instruction they can receive) via TCP/IP socket connections. Typically, clients which are going to act as slaves connect to WebCom via a web server. They then receive atomic instructions by way of a Java applet, execute these instructions within the confines of their browser and return results to the WebCom server via a dedicated TCP/IP socket connection. Clients which act as potential masters are implemented as Java applications and are capable of executing atomic and Condensed Node instructions. On receipt of a Condensed Node, a potential master is promoted to be a fully fledged master, is assigned a number of clients to help in executing its atomic instructions and becomes the root of another computation tree distributed over the web. WebCom employs a fault survival mechanism to counteract the volatile nature of its underlying platform. Typical web based faults include the failure of TCP/IP sockets due to normal network failure. Also, client machine users may wish to exit from WebCom by killing their browser or pointing it to a different location.

WebCom-G [112] [111] is the evolution of WebCom for producing a grid-enabled middleware. The aim of the WebCom-G OS is to hide the low level details from the programmer while providing the benefits of distributed computing. The WebCom-G System is proposed as a Grid Operating System. It is modular and constructed around a WebCom kernel, offering a rich suite of features to enable the Grid. WebCom-G utilizes the tested benefits of the WebCom metacomputer and leverages existing grid technologies such as Globus and MPI. To WebCom-G, multiple grids are themselves viewed as independent WebCom-G units. When an instruction is sent to WebCom-G all the information is supplied to either dynamically create or invoke an RSL script or to execute the job directly. When a WebCom-G unit receives an instruction it is passed to the grid engine module. This module unwraps the instruction, creates the RSL script and directs the gatekeeper to execute it. Once the Gatekeeper has completed execution, the result is passed back to the unit that generated the instruction. As WebCom-G uses the underlying grid architectures, failures are detected only at the higher level. In this case WebCom-G.s fault tolerance will cause the complete job to be rescheduled.

2.4 Grids versus Desktop Grids

Desktop Grid and Grid computing share the same goal of better utilizing existing computing resources. However, there are key differences between the two paradigms. As a matter of fact, Grid computing involves organizationally-owned resources: supercomputers, clusters, and PCs owned by universities, research labs, and companies. These resources are centrally managed by IT professionals, are powered on most of the time, and are connected by full-time, high-bandwidth network links. Furthermore, there is a symmetric relationship between organizations: each one can either provide or use resources. In contrast, Desktop Grids involve an asymmetric relationship between projects and participants. Projects are typically in a special interests based academic research groups. Most participants are individuals who own Windows, Macintosh and Linux PCs, connected to the Internet by telephone or cable modems or DSL, and often behind network-address translators (NATs) or firewalls. The computers are frequently turned off or disconnected from the Internet. Participants are not computer experts, and participate in a project only if they are interested in it. Projects have no control over participants, and cannot prevent malicious behavior.

The second difference is that, Grid computing has many requirements that Desktop Grid computing does not. Grids architecture must accommodate many existing commercial and research-oriented academic systems, and must provide a general mechanism for resource discovery and access. In fact, it must address all the issues of dynamic heterogeneous distributed systems, an active area of computer science research for several decades. This has led to architecture such as OGSA, which achieves generality at the cost of complexity and even performance. In contrast, the main characteristic of Desktop Grid systems is the unobtrusiveness because the resource used are installed and designed for purposes other than distributed computing, thus the resource must be exploited without disturbing their primary use. Moreover, the machine including its data, hardware, and processes must be protected from a misbehaving of Desktop Grid applications. Analogously, the application's executable, input, and output data, which may be proprietary, must be protected from user inspection and corruption.

More detail comparison between Grids and Desktop Grids can be summarized using Table 2.2.

Table 2.2 Grids versus Desktop Grids

Items	Desktop Grids	Grids
Initial ideas	To collect idle cycle of PCs connected by Internet to do scientific computing	To harness more clusters, supercomputers, devices into a meta-computer which can provide users on-demand process power
Resources	Supercomputer, cluster, scientific instruments and so on.	Mainly PCs, it also can be workstation, supercomputers…
Connection	Internet based Poor bandwidth; low speed	LAN based high speed network
Task management	Pull mode	Push mode
Dedication	Non-dedicated High volatile	Dedicated Utilize based on “reservation”
Trust	Low confidence Results need certification	High confidence
Stable	Failure is common Fault tolerance mechanism is necessary	More stable

Application	high throughput computing (mainly);
	Job independent
		High performance computing;
		High throughput computing;
		Job independent or Job dependent.

Scalability	Easy to scale with low cost Large scale based systems	Very easy to scale with no cost Very large scale based systems
Quality of service	High Or service level agreement based services	Low
Heterogeneity	High heterogeneous	Heterogeneous

2.5 Conclusion

In this chapter, we briefly presented the evolution of high performance systems. Generally speaking, its development experienced two stages. The first is to improve the process power of a single computer through adopting new materials of chips (from electron tube to transistor) and start-of-art technology (digital large scale integrated circuit, for example). But as well known to us all, the signal couldn't be transmitted faster than the speed of light and perhaps Moore's law doesn't work in future. So the second method of improving process power is to harness lots of CPU/computers to finish tasks cooperatively. Two kinds of methods can be divided into according to what is harnessed (CPU or computer). In 80's, the main way is to harness many CPUs to improve the performance of supercomputer. Then the cluster emerged in 90s and it became the main stream of high performance systems. Up to the middle of 90s, the architecture of high performance systems is mostly homogeneous.

With the development of programming language and emergence of Java, virtual technology make programming interface be platform independent and easy of networking. So the concept of Grids and Desktop Grids are proposed and they become the main stream in high performance systems. Lots of Grid systems have been utilized in real society, such as LCG project for LCH, EGEE for high energy physics and TeraGrid for scientific computing. At the same time many Desktop Grids, Seti@home, Condor, XtremWeb for example, also have been widely used in scientific computing. Some famous Grid systems and Desktop Grid systems have been described in this chapter and some comparisons between Grids and Desktop Grids are also made in this chapter.

Though there are many successful applications on Grids and Desktop Grids. Some drawbacks still exist in those Grid and Desktop Grid projects. Also from the description of Grids/Desktop Grids, we also know different Grid/Desktop Grid systems suit for different applications, this is to say, no a general Grid/Desktop Grid systems can suit for all the applications/situations. So the aim of this chapter is to analyze different Grid/Desktop Grid systems and find an appropriate way to make scientific computing more efficiently. Our research is for not big enterprises or research institutes whose users are non professional computer users. So we should try to find a way to make scientific computing more simply. At the same time, we will try to utilize the computing resources we can use. According to the research context, we choose appropriate experiment tools and environments to make our experiments. Detail on experimental tools and environments will be introduced in chapter three.

Chapter 3

Experimental Tools and Platforms

This chapter presents our experimental tools and platforms. We choose the OmniRPC (Grid middleware), XtremWeb (Desktop Grid middleware) and YML (workflow language based middleware) as our software/tools; Grid5000 (Grid environment), Polytech Lille (Desktop Grid environment) and Hohai platform (Desktop Grid environment) as our platforms/environments. The reason why we choose the softwares and platforms presented above is from the background of the research in the dissertation.

The aim of this dissertation is to find an easy of use way to make large scale scientific computing for non professional computer end users. This is to say, the new solution will try to provide higher level interface to end users and it is better for users to develop using the interface without knowing much about programming knowledge. To achieve this goal, a component based framework YML is presented in this chapter. YML an open source middleware, is a workflow based framework which can help provide end users high level programming interface and users also can extend YML by modifying its components or adding a new component.

As well known to us all, volunteer computing resources can be harnessed into a powerful platform and Desktop Grids become more and more important in scientific computing area. So an open source desktop middleware XtremWeb is selected to testify the performance of volunteer computing resources based platform. The reason of using XtremWeb can be described as follows: firstly, it is a java based Desktop Grid middleware which can provide three kinds clients program separately suiting for Windows OS, Linux OS and Mac OS. Secondly, XtremWeb can support different scale applications. It can be used to make large scale scientific computing such as high energy physics and small or middle scale applications also can be dealt with using Xtremweb. Thirdly, it is open source software.

In scientific computing areas, dedicated computing resources (for example, supercomputer, workstation, cluster, servers) always play an important role. Recently, more and more scientific computing applications are based on distributed computing. So this dissertation chooses OmniRPC to harness dedicated computing resources. The reason of choosing it can be summarized as follows: firstly, it is RPC based middleware and it has more efficiency than virtual technology based middleware (for example, RMI based middleware); secondly, OmniRPC can support a local environment with "rsh" and remote hosts with "ssh"; thirdly, it can provide SMP environment and a Grid environment with Globus. It also supports programming in Fotran and OmniRPC API; fourthly, it provides end users with several kinds of scheduling mechanism and user can choose appropriate scheduling strategy according to real situation.

As to platform, Grid5000 a national wide Grid platform in France and two Desktop Grid platforms in Polytech Lille, France and Hohai platform, China are also presented. Our experiments are based on those platforms.

3.1 Experimental platforms

3.1.1 Grid5000

Grid'5000[14] [113, 114] is national wide experimental set of clusters which aims at providing a strong reconfiguration, control and monitoring infrastructure, transforming the full system into a scientific instrument. Actually, its reconfiguration capability allows a large variety of configurations. The full software stack between the hardware and the Grid user can be configured on all processors, at a nation wide scale. Grid'5000 can be used and shared by many researchers. Its control infrastructure allows researchers running interactive as well as batch experiments. Experiments in batch mode follow a complex sequence of operation including the following steps: reservation, reconfiguration, run preparation, run and post run resource liberation.

Grid'5000 is implemented as a nation wide cluster of clusters. Nine sites in France are equipped with clusters ranging from 100 to 1000 CPUs. The sites are connected by RENATER through VLANS implemented by MPLS at level 2. The clusters are not connected to the rest of the Internet. From the Internet, accesses to clusters are made through a first access to the laboratory hosting the cluster. Strong authentication and authorization verification procedures check the access rights of users. When a user is logged in Grid'5000, he can see all the Grid'5000 processors without restriction. A reservation tool called OAR allows to reserve

[14] https://www.grid5000.fr/mediawiki/index.php/Grid5000:Home

nodes on several sites. After the reservation, the user can deploy a specific stack of software and reboot the reserved nodes. After the reboot, the user can start the experiment. A set of probes inside RENATER allows capturing and recording the inter-site network traffic during the experiment. Grid 5000 will offer a set of tools controlling the experimental conditions during the execution of the experiment. Basically, the user will be able to start and stop every Grid 5000 nodes, on demand. Grid'5000 is used for many experiments in all levels of the Grid software stack: resource provisioning systems, network protocols, operating systems mechanisms, middleware, runtime and applications. Researchers can study issues (performance, scheduling, fault tolerance, QoS, etc) on the Grid5000 platform.

3.1.2 Polytech Lille and Hohai platforms

Polytech Lille platform: its machines are teaching machines at the university school of engineer of Lille. There are total 128 machines which are non-dedicated machines under Linux OS. The bandwidth varies from 10 to 100MB/S in the LAN and it's about 1GB/S between those switches.

Hohai platform: this platform also belongs to non-dedicated platform which are from 2 labs. Those PCs are utilized when there is an experimental course and if no course in the labs, users/clients can utilize the machines in the labs. The bandwidth between machines is based on 100M/S and the bandwidth between labs is 1GB/S. There are total 108 machines.

3.2 Experimental tools

3.2.1 XtremWeb

XtremWeb[15] [115] [116] an open source middleware to form global computing platform for a multi-user and multi-parallel programming context, intends to distribute applications over dynamic resources according to their availability and implements its own security and fault tolerance policies. XtremWeb relies on the notion of services to deploy a Desktop Grid based on a three-tiers based architecture. This architecture gathers tree main services: Clients, Coordinators and Workers. Detail architecture can be described using Figure 3.1.

[15] http://www.xtremweb.net/

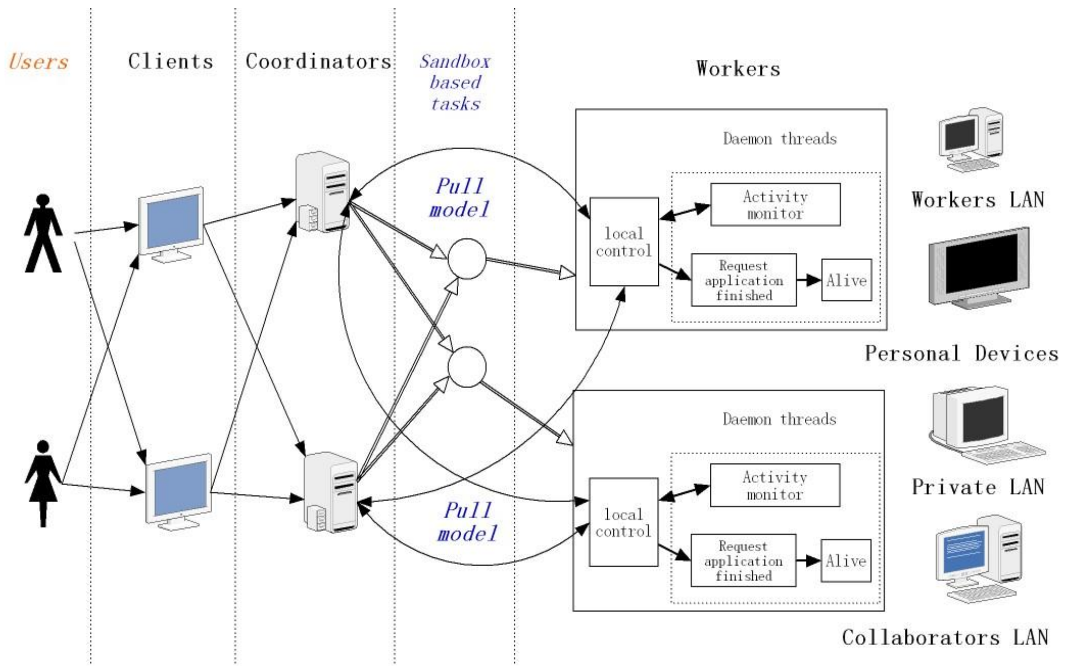


Figure 3.1. Architecture of XtremWeb

- The coordinator. The coordinator manages a bag of tasks provided by clients and coordinates their scheduling among a set of workers that are volunteers provided by institutional or private users. But those workers are not under the control of the coordinator and they can join in or leave from the platform at any time. Under this situation, each action is initiated by workers and this behavior model is commonly known as pull model. Tasks are scheduled to workers on their specific demand only since they may appear (connect to coordinator) and disappear (disconnect from coordinator) with no predictable pattern (a worker is then said connected as long as it periodically contacts the coordinator). Scheduling mechanism in XtremWeb is based on FIFO (first in, first out) mode. Any scheduled task is expected to be computed by a worker and have its results sent back to the coordinator; once the task is failure, it will be rescheduled to another worker to get correct results. XtremWeb can schedule applications as jar files (binary files) to ensure its security.
- The clients. Clients are distributed to authorized users only to make them able to submit tasks to the coordinator as transactions. Before submitting any task, the client contacts the coordinator to fetch any previous submitted ones. This ensures that when the client restarts from a fault or any other reason, it does not resubmit previously submitted tasks. Results are managed according to the user needs. They can be discarded immediately after fetch or kept by the

coordinator until the end of the session. So on client failure, it is the responsibility of the client programmer to fetch relevant results. An API is implemented to provide such secure implementations.

- The workers. Workers are distributed entity to volunteer institutional or individual PCs, which aim to use CPU accordingly to a local user customizable policy (available scheduling time, CPU usage conditions...) to compute tasks provided by the coordinator. A worker requests task to compute accordingly to its own local policy; it downloads task software and all expected objects (input file, arguments...), stores them on reliable media and starts computing the provided tasks. Computation goes on locally until it ends or dies for any reason, including due to host utilization policy rules. As computation is started, the worker periodically signals the coordinator, so that coordinator knows the computation goes on well. The coordinator manages its tasks using transactions and stores them in reliable media (disk) so that the full system integrity is preserved even if the coordinator shuts down for any reason. At starting time, the coordinator reads the information stored on disk to set up its proper state. The client submits tasks and the worker fetches tasks using transactions; this ensures a consistent state when the coordinator restarts from fault and the client/worker have not failed. Workers failures are detected (by the alive signal) and their tasks may be rescheduled on another available worker. Also, to avoid redundant task and result overwriting, a worker can be brought to stop its current task if it has been disconnected for a long time.

3.2.2 OmniRPC

OmniRPC[16] [16] is designed to allow easy development and implementation of parallel scientific applications for distributed and Grid environments. OmniRPC is an evolution of Ninf, since it inherits the API and basic structure from it, and thus the OmniRPC programming model is very similar to the GridRPC one. It is composed of a client application and various remote computational hosts, which execute the remote procedures. Remote locations can be connected via a local area network or over a wide-area network. The client application can be written in various different languages, such as FORTRAN, C and C++, and the parallel execution in the client can be obtained by using direct thread libraries, such as the POSIX thread, or the OpenMP API. The interface to a remote function is described by the Ninf IDL. In OmniRPC, the remote executions are managed by the use of remote shell (rsh) for local distributed environments and by the use of Globus and ssh for Grid environments.

OmniRPC follows a master-worker structure. It consists of three parts which are client, agent and computational hosts. Its architecture can be described through figure 3.2.

[16] http://www.omni.hpcc.jp/OmniRPC/

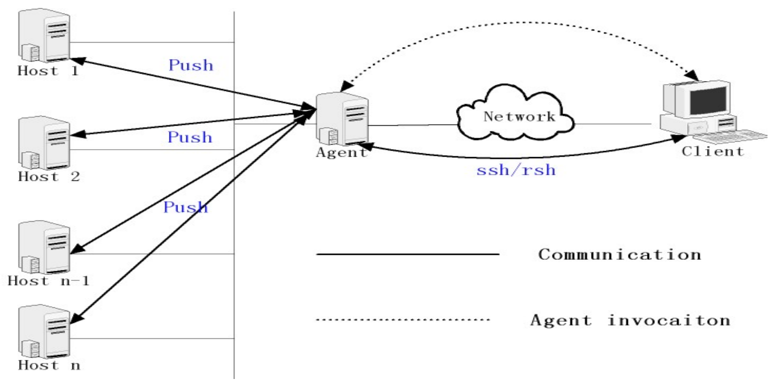


Figure 3.2 General architecture of OmniRPC

The executable process can be described as follows: when the OmniRPC client program starts, the initialization function of the OmniRPC system invokes the OmniRPC agent program omrpcagent in the remote hosts listed in the host file. In order to invoke the agent called omrpc-agent, the user can use rsh in a local-area network, the Globus Resource Allocation Manager of the Globus toolkit in a Grid environment, or ssh in a private Grid environment. The user can switch the configurations only by changing the host file.

OmniRpcCall is a simple client programming interface for calling remote functions. When OmniRpcCall makes a remote procedure call, the call is allocated to an appropriate remote host. When the client issues the RPC request, it requests that the agent in the selected host submit the job of the remote executable with the local job scheduler specified in the host file. If the job scheduler is not specified, then the agent executes the remote executable in the same node by using the fork system call. The client sends the data of the input arguments to the invoked remote executable and receives the results upon return of the remote function. Once a remote executable is invoked, the client attempts to use the invoked remote executable for subsequent RPC calls in order to eliminate the cost of invoking the same remote executable again. RPCs are properly allocated in the idle node. When the agent and the remote executables are invoked, the remote programs obtain the client address and port from the argument list and reconnect to the client by direct TCP/IP or Globus-IO for data transmission. Because the OmniRPC system does not use any fixed service ports, the client program allocates unused ports dynamically in order to wait for a connection from the remote executables. This avoids possible security problems and allows the user to install the OmniRPC system without a privileged account.

In our experimental environment, the Grid resource is based on cluster of geographically distributed PC clusters. For PC clusters on a private network, an OmniRPC agent process on the server host functions as a proxy to relay communications between the client and the remote executables by multiplexing the communications using a single connection. This feature, called multiplex IO (MXIO), allows a single client to use up to 1,000 remote computing hosts. When the PC cluster is located inside a firewall, the port forwarding of SSH enables the node to communicate to the outside via MXIO.

3.2.3 YML

YML[17] [117] [118] [119] is a framework, developed at PRiSM laboratories in collaboration with Laboratoire d'Informatique Fondamentale de Lille (Grand Large Team, INRIA Futurs). The aim of YML is to define an abstraction for middlewares, hiding differences among them and using this abstraction to remain portable over multiple middlewares. The user can easily develop a complex parallel application which may transparently execute on multiple middlewares during one application execution. The framework can be divided into three parts which are "end-users interface", "frontend" and "backend".

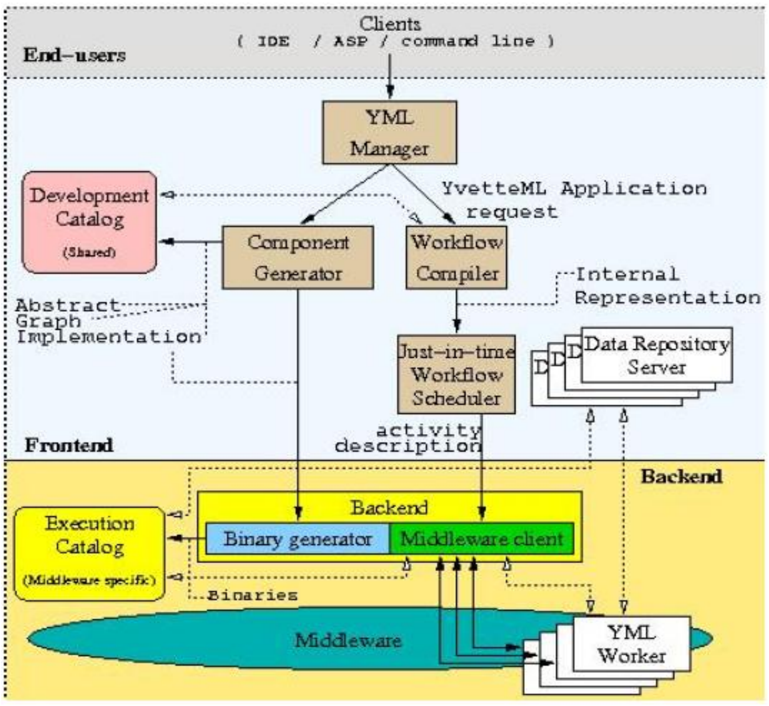
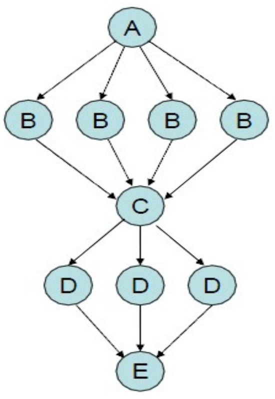


Figure 3.3 General architecture of YML

[17] http://yml.prism.uvsq.fr/

“End-users interface” is used to provide an easy-to-use and intuitive way to submit their applications and applications can be described using a workflow language YvetteML. YvettML is a dedicated language embedded in XML which helps to make the writing of application description of application graph more human friendly. It can support a ‘graph component’ based program (See an example in Figure3.4). The YvetteML language is used to control the activity of YML. The main role of the language is to describe component and execution. It focuses on the control provided to the user to manage the creation of an application and it allows the coordination and the description of the dependencies between the various component calls composing the graph of an application. The ‘graph component’ based interface is separation of the control or communication of the application and the computation. Several structures in YvetteML can be described as follows:

- Component call: It corresponds to the execution of a component on the remote peers. The component call is composed of a scheduling policy that can be used to invoke the name of an abstract component and a list of parameters. Parameters can be a data or a constant value.
- Parallel section: In YvetteML like most programming language the control flow is sequential until you explicitly mark section as parallel. YvetteML supports two kind of parallel sections. Parallel section are separated by a “//” separator. Multiple statements can compose a parallel section. “Par” and “Endpar” are keywords of parallel section.
- Sequential loop: You can also express sequential loop controlled by an iterator using a form similar to the parallel iterated loop described previously. The example does the same as before but sequentially.
- Conditional: YML allows the definition of conditional branches based on a boolean condition. The condition can use the operator available in the language as well as any user defined function.
- Events manipulation: The events can be explicitly managed using two constructions called notify and wait. The first construction set a list of events. The second one is used to wait until a boolean expression of events is true. “wait()” and “notify()” are two keys to execute dependable tasks.



```
Sequential
     A
//
   Parallel
     B
   End Parallel
//
    C
//
   Parallel
    D
   End Parallel
//
    E
End Sequential
```

Figure 3.4 Sample of YvettML based interface

"Frontend" is the main part of YML which includes compiler, scheduler, data repository and Development Catalogs (in this part, two components will be developed or invoked based on reuse: abstract component and implementation component). Abstract components and implementation components based on function can be reused very easily and we will explain it in the fifth chapter.

- Compiler: It translates applications described using the YvetteML language to a set of components calls. Its component call is decorated with two pieces of information. The execution condition is a boolean expression which determines whether the component can be executed or not. The post condition is a list of boolean flags used to describe the state of the application. Execution conditions are evaluated based on the events.
- Scheduler: It manages application executions. It acts as a client for underlying middleware accurately requiring computing resources. During the application execution the scheduler detects task ready for execution solving dependencies at runtime. Each scheduling iteration may or not generate a set of parallel tasks which are translated in computing requests to middleware through dedicated back-ends
- Data Repository: The YML framework implies a lot of data exchange through the network. The Data Repository server acts as a resource provider and delivers data to each component on demands.
- Development Catalogs: The YML framework stores components in Catalogs. The Development Catalog stores the information used only during the development stage. The middleware independent part relies only on this catalog. The Development Catalog store components information and data type information used to validate the YvetteML input program. This catalog is not required for executing an application. Two kinds of components are needed to develop which are abstract component and implemental component.
    1. Abstract component: an abstract component defines the communication interface with the other components. This definition gives the name and the communication channels with other components. Each channel corresponds to a data input, data output or both and is typed. This component is used in the code generation step and to create the graph. This part of work can be developed by computer engineering in advance according to users' requirement.
    2. Implementation component: an implementation component is the implementation of an abstract component. It provides the description of computations through YvetteML language. The implementation is done by using common languages like C or C++. They can have several implementations for a same abstract component. This part of work also can be developed by computer engineers in advance.

"Backend" is the part to connect different Grid and peer to peer middleware. This layer is responsible for the interaction between YML and middleware. Its goal is to allow YML to execute the YML service on a worker. In order to accomplish this, the back-end layer defines three components: the back-end, the worker and the data repository. These components interact with the middleware to provide a consistent execution environment to YML services.

- The back-end component is a client of middleware services. It permits YML to request the execution of a work on a peer. It also notifies of work terminations. The application executed on peers is the YML worker component.
- The worker component is a service container. It defines a consistent environment for the execution of a service. Among the tasks assumed by the worker, the most significant is data management. The worker is responsible of importing data for the service and exporting its results. In order to do that, the worker component interacts with data repositories.
- Data repository components are responsible for all data exchanges between two peers and with the other components. A data repository component provides two services to its local or distant clients: the publication of a resource and its retrieval.

3.2.4 Relation between XtremWeb/OmniRPC and YML

Just as presented above, XtremWeb is a middleware for Desktop Grid environment and it is based PULL model. If the worker has CPU idle, it will take the initiative to get tasks from coordinator and then send its result to coordinator after finishing it. Its programming interface is based Java based API; OmniRPC is a middleware for cluster and Grid environment and it is based on PUSH model. Users will set the computational environment by themselves and agent will push tasks to related workers according to users' configuration. Its programming interface is based GridRPC API; YML is a framework aiming at making users program in cluster, Grid and Desktop Grid environments more simply. It is based on workflow technology and suits for large scale scientific computing. Users can program in pseudo-code mode based YvettML language and YML will parse the pseudo-code program and generate "Task Table". Then those tasks will be allocated to YML workers. YML workers are interfaces to different middleware which can be XtremWeb or OmniRPC or both. Their relations can be described using Figure 3.5.

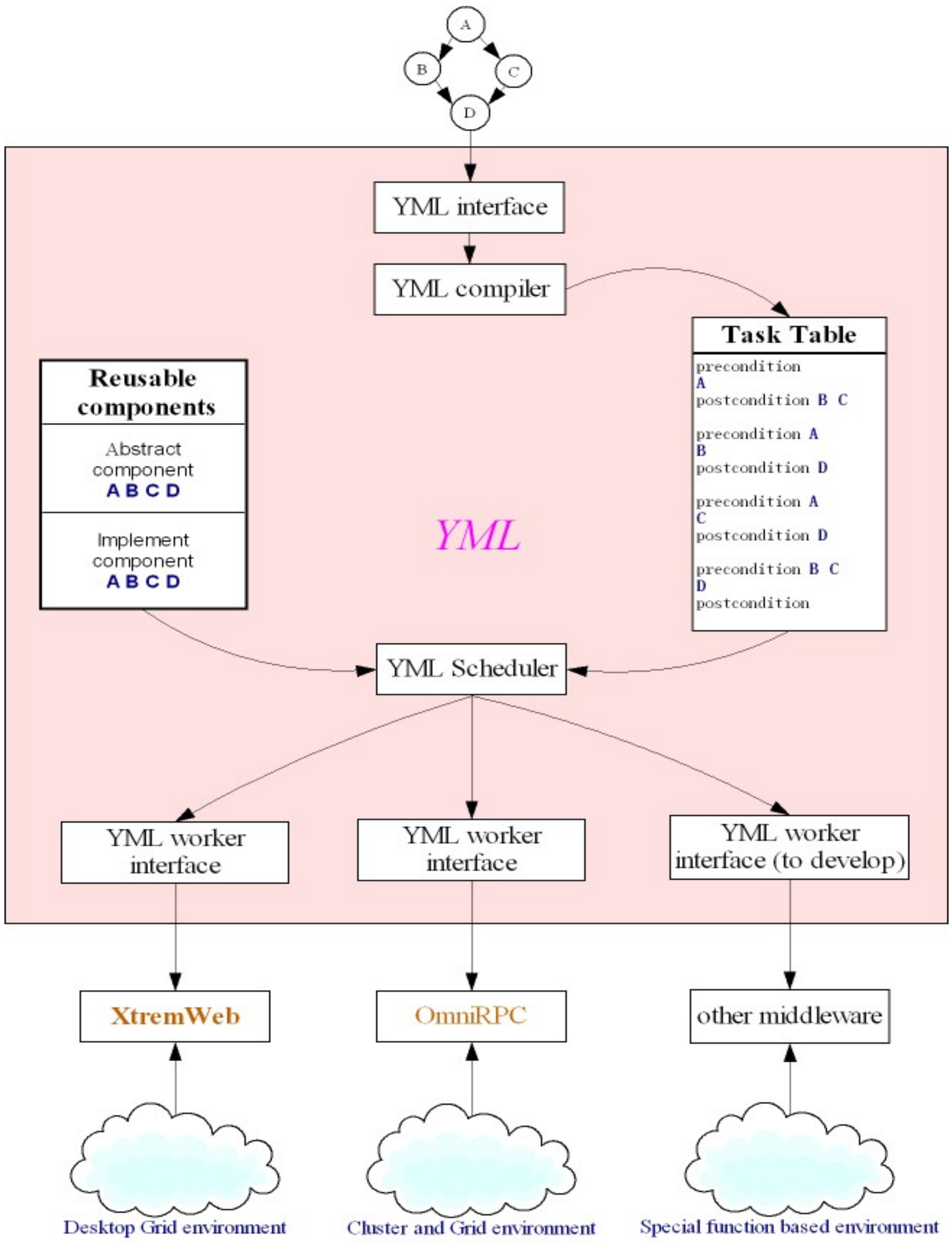


Figure 3.5 Relation between XtremWeb/OmniRPC and YML

Chapter 4

A New Parallel Programming Adapted Version for Block Based Gauss Jordan Algorithm

4.1 Motivation

Large scale matrix inversion has been widely used in many scientific research domains. As a classical method of large matrix inversion, block-based Gauss-Jordan (BbGJ) algorithm has aroused great concerns among many researchers. Many people have developed parallel versions of BbGJ [122] [121] [120] [123]. But large granularity based task parallelism (intra-steps data dependence based tasks parallelism) disables mapping more tasks on computing resources simultaneously. As a result, the performance of those parallel versions degrades when running on the Grid platform consisting of a lot of heterogeneous PCs and workstations. So this chapter will make further research on BbGJ and analyze all the data dependences between operations. Then a series of executable rules on when each operator can be executed are made. Finally a fine-grained task based parallel adapted version of BbGJ algorithm in which both intra-steps and inter-steps based data dependences are considered, is presented in this chapter.

4.2 Sequential algorithm of block based Gauss Jordan

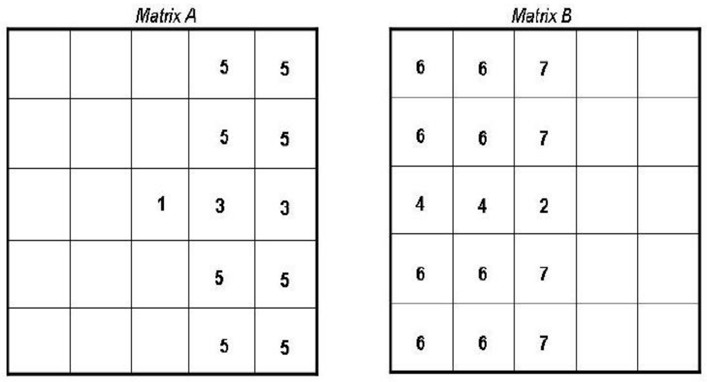
The sequential BbGJ algorithm is a classical linear algebra method to get a large scale matrix inversion. The algorithm can be described as follows: Let A and B be two dense matrices of dimension N, and let B be the inverse of A, i.e. AB=BA=I. Let A and B be partitioned into a matrix of q×q blocks of dimension n which n = N/q.

Sequential algorithm of BbGJ

```
    Input: A,B ← I (n, n), q
    Output: $B=A^{-1}$
-----------------------------------------------------------------
    For k=1 to q do
        $A^{k}_{k,k} \leftarrow (A^{k-1}_{k,k})^{-1}$                         (1)
        $B^{k}_{k,k} \leftarrow A^{k}_{k,k}$                                  (2)
        For j=k+1 to q do
            $A^{k}_{k,j} \leftarrow A^{k}_{k,k} A^{k-1}_{k,j}$                (3)
        End For
        For j=1 to k-1 do
            $B^{k}_{k,j} \leftarrow A^{k}_{k,k} B^{k-1}_{k,j}$                (4)
        End For
        For j=k+1 to q do
            For i=1 to q and i≠k do
                $A^{k}_{i,j} \leftarrow A^{k-1}_{i,j} - A^{k-1}_{i,k} A^{k}_{k,j}$    (5)
            End For
        End For
        For j=1 to k-1 do
            For i=1 to q and i≠k do
                $B^{k}_{i,j} \leftarrow B^{k-1}_{i,j} - A^{k-1}_{i,k} B^{k}_{k,j}$    (6)
            End For
        End For
        For i=1 to q and i≠k do
            $B^{k}_{i,k} \leftarrow - A^{k-1}_{i,k} A^{k}_{k,k}$                (7)
        End For
    End For
```

4.3 Parallelism in the algorithm

Through the sequential BbGJ algorithm, we can find that each iterative step of algorithm is made up of five loops (the third, fourth, fifth, sixth and seventh operation marked in sequential BbGJ algorithm) and two other operations (the first and second ones in sequential BbGJ algorithm). In the iterative step k (k = 1,…,q), the first operation is used to get the inverse of pivot block $A^{k-1}_{k,k}$; the second operation will assign the sub-block of matrix A to corresponding sub-block of matrix B; the third operation computes the blocks of the row k of matrix A with subscripted variables j above k while the fourth operation acts on the row k of matrix B with subscripted variables j below k; the fifth operation calculates the blocks of all columns of the matrix A with subscripted variables i above and below k and subscripted variables j above k; while the sixth operation calculates the blocks of all columns of the matrix B with subscripted variables i above and below k and subscripted variables j below k. At last the seventh operation is used to compute the blocks of the column number k of matrix B except $B_{k,k}$. Refer to Figure 4.1.



			5	5
			5	5
		1	3	3
			5	5
			5	5

6	6	7		
6	6	7		
4	4	2		
6	6	7		
6	6	7		

Figure 4.1. Operations in the 3rd iterative step with q is 5

So, the parallelization of the sequential BbGJ algorithm consists of exploiting two kinds of parallelism: the inter-steps based parallelism and the intra-step based parallelism. The intra-step parallelism aims at exploiting the parallelism involved in each of the five loops (operation ‘3’ to operation ‘7’ in sequential BbGJ algorithm). It falls into two categories: the inter-loops parallelism and the intra-loop parallelism. See Figure.4.2.

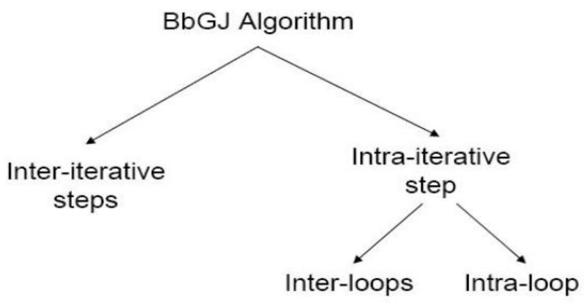


Figure 4.2. Parallelism in the block-based Gauss-Jordan algorithm

Next, we will analyze those parallel relationships one by one.

4.3.1 Intra-step based parallelism

4.3.1.1 The Inter-loops based parallelism

The inter-loops parallelism refers to operations between loops. There are dependable relationships which determine when those operations between loops can be executed. In the sequential BbGJ algorithm, we identify three ones: 1) the second, third, fourth and seventh operation (see the number marked in sequential BbGJ algorithm) can't be executed before knowing the value of block $A^k_{k,k}$; 2) the loop of the fifth operation depends on the third one because its input (the block $A^k_{k,j}$) is the output of the third operation; 3) the sixth operation is dependent on the fourth operation because its input (the block $B^k_{k,j}$) is the output of the fourth operation.

4.3.1.2 The Intra-loop based parallelism

The intra-loop parallelism consists of executing each of the five loops (operations '3', '4', '5', '6' and '7') of the algorithm in parallel. They all belong to SPMD (single program multi data) paradigm. The work units for these loops are of two kinds: matrix product and matrix triadic respectively X×Y and Z-X×Y, where X, Y and Z are sub-matrix blocks. Moreover, to get higher performance in a certain platform, the granularity of tasks is very important. Here we can balance the computing time and communication time through choosing appropriate block-size of sub-matrix. The experiment part of this chapter will show us the influence on the efficiency of algorithm caused by changing the block-size and block-count of matrix.

4.3.2 Inter-steps based parallelism

Next, we will take the data-dependence between step k and step k+1 as example to show the inter-step parallelism in the BbGJ algorithm. Getting the value $A_{k,j}$ (k=1…q; j=k+1…q) (respectively $B_{k,j}$ in which k=1…q; j=1…k-1) of k+1 step in the third operation (respectively the fourth) needs get the value of block $A_{k,j}$ (k =1…q; j=k+1…q) (respectively $B_{k,j}$ in which k=1…q; j=1…k-1) of k step. As to the fifth operation, we must know that block $A_{i,j}$ (i=1…q and $i\neq k$; j=k+1…q) and $A_{i,k}$ (i=1…q and $i\neq k$; k=1…q) of k step before we compute the value $A_{i,j}$ (i =1…q and $i\neq k$; j= k+1…q) of k+1 step. The situation in the sixth operation is just like that in the fifth, we can compute the value $B_{i,j}$ (i=1,…,q and $i\neq k$; j=1…k-1) of step k+1 after we getting the value $B_{i,j}$ (i =1…q and $i\neq k$; j=1…k -1) and $A_{i,k}$ (i =1…q and $i\neq k$; k=1…q) of step k. As to the operation '7', to get the value $B_{i,k}$ (i=1…q and $i\neq q$; k=1…q) of k+1 step, we must know the value $A_{i,k}$ (i=1…q and $i\neq q$; k=1…q) of step k.

4.4 Description of data dependence

We have analyzed all the possible data dependence in the BbGJ algorithm, but it is hard to understand and find the regularity. So in this part an intuitive method (using tables) is presented to express data dependence.

4.4.1 Description of Intra-step based data dependence

Then we can describe the data dependence in sequential BbGJ algorithm using Figure 4.3 and Figure 4.4.

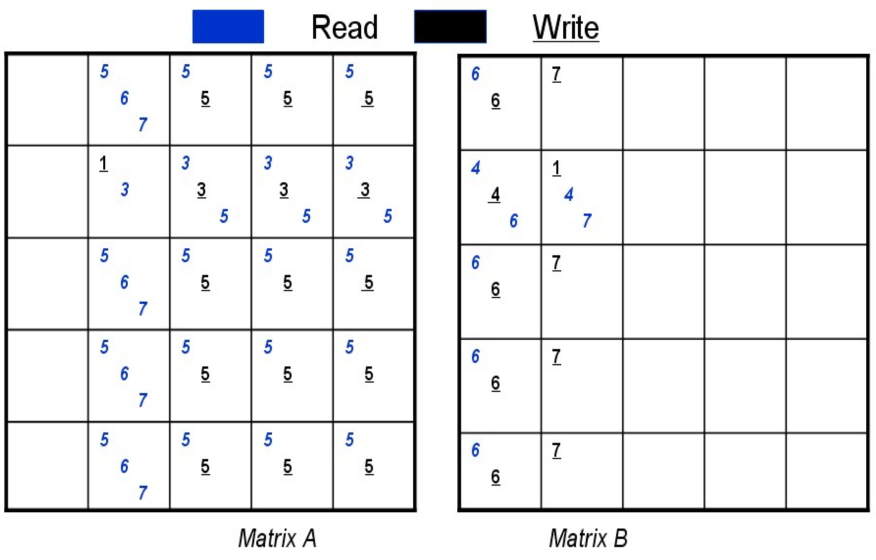


Figure 4.3. Block based operation at the 2ed iterative step with q is 5

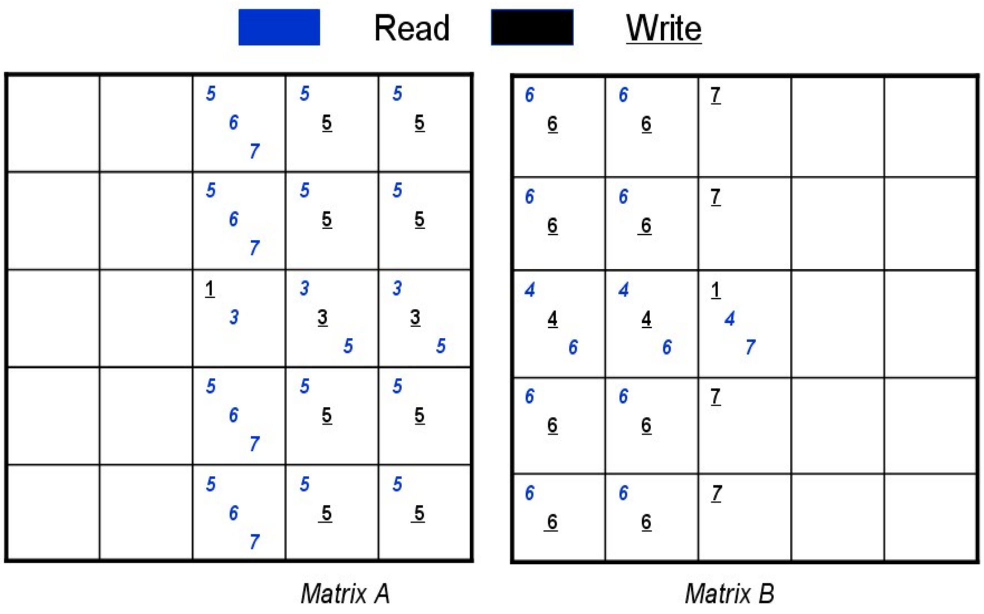


Figure 4.4. Block based operation at the 3rd iterative step with q is 5

In those figures the italic font (in blue) represents the read operation (a value will be read from the block as data output) and black font with underline donates the write operation (a value will be written into the block as data input). For example, the italic font '5' in the first row, second column of matrix A in Figure 4.3

represents that: to this block, the first operation is ‘5’ (see the number marked in sequential BbGJ algorithm) which is a read operation, i.e. operation ‘5’ will read the data from this block as its input. The same way we can know that the black font with underline ‘1’ in the second row, second column of matrix A in Figure 4.3 represents that: the operation ‘1’ will assign a new value to this block. Next we will take the block in the second row, first column of matrix B in Figure 4.3 as example to show all the related operations executed in this block. From the Figure 4.3, we can know three operations executed in this block, of which italic fonts ‘4’ and ‘6’ are read operations and black font with underline ‘4’ is write operation. To this block, operations will be executed from up to down, i.e. the read operation ‘4’ will read from this block as its data input at first, then write operation ‘4’ assigns a new value to this block as its data output. Finally, read operation ‘6’ will read the value of this block as its data input. After that, no more operations are made on this block in this iterative step. Figure 4.3 shows us all the operations happened at the second iterative step with q is 5 in the BbGJ algorithm. We can also get all the operations information at the third iterative step with q is 5 from the Figure.4.4. Through the analysis, we can summarize the intra-step based data dependence using Figure4.5.

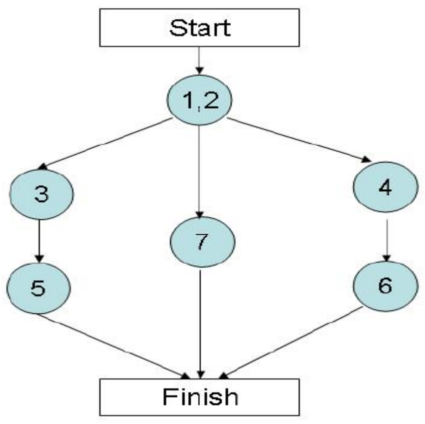


Figure 4.5. Intra-step based data dependence

4.4.2 Description of inter-steps based data dependence

We have presented the intra-step data dependence through the Figure.4.3 and Figure.4.4. Now we will analyze the inter-steps based data dependence. To do that, we define “directed arc” as follows: a directed arc from block “X” to block “Y” (X, Y are blocks of matrix) signifies that the operation on block “Y” can’t be executed before the operation on block “X” finishes. See directed arc in Figure.4.6.

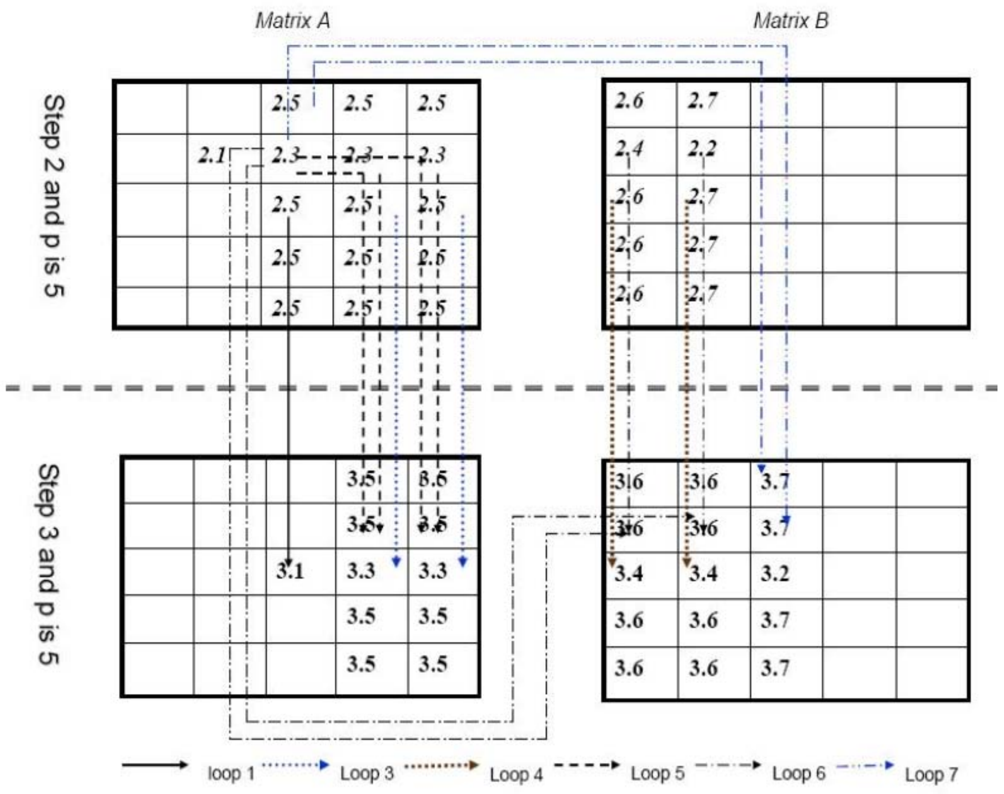


Figure 4.6. Inter-steps based data dependence

There are two operations in ‘loop 1’, which are ‘2.5’ and ‘3.1’. ‘2.5’ represents the fifth operation in second iterative step of BbGJ, while ‘3.1’ stands for the first operation in the third iterative step of BbGJ. ‘Loop 1’ in the Figure.4.6 shows us data dependence between operation ‘1’ at second iterative step and operation ‘5’ at the third iterative step. And it means the operation ‘5’ at third iterative step can’t be executed until the finishes of operation ‘1’ at second iterative step. In the same way, we can analyze the data dependence represented by ‘loop n’ (n is 1, 3, 4, 5, 6, 7). Figure.4.6 shows us all the inter-steps based data dependence existing between step 2 and step 3 with q is 5.

4.4.3 Description of all the data dependences

From the Figure.4.5, we can know all the intra-step based data dependence and Figure.4.6 show us all the inter-steps based data dependence. Then we can summarize all the data dependence existing in BbGJ algorithm using the Figure.4.7.

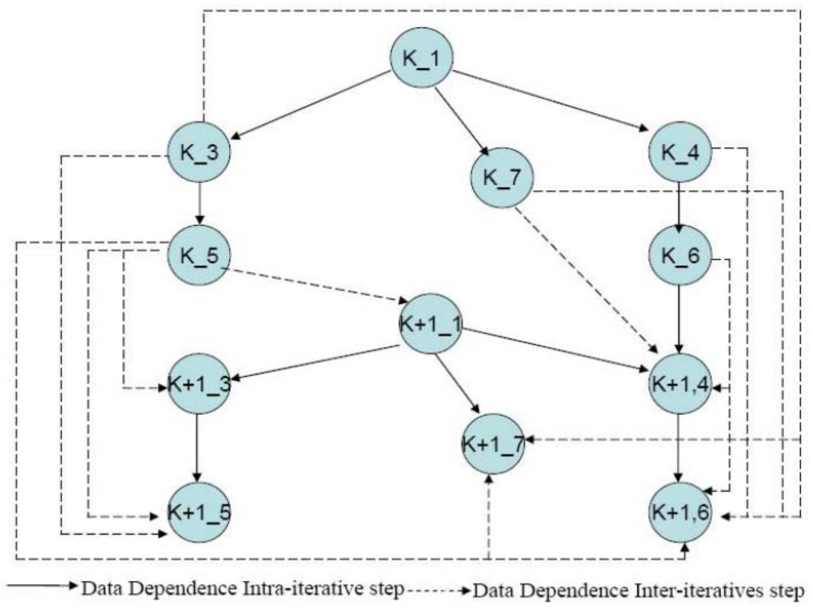

K_1
K_3
K_7
K_4
K_5
K_6
K+1_1
K+1_3
K+1,4
K+1_7
K+1_5
K+1,6
Data Dependence Intra-iterative step
Data Dependence Inter-iteratives step

Figure 4.7. All the data dependences in the Algorithm

4.5 Formal description of data dependence

From Figure 4.3 and Figure 4.4, we can find that each iterative step of the algorithm has the same number of write operations and the number is q. To the matrix A ,the write operation on it are operations ‘1’, ‘3’ and ‘5’, while operations ‘4’, ‘6’ and ‘7’ act on matrix B. It is the similarities existing in write and read operation on matrix A and matrix B that we can take matrix A and matrix B together into account as an augmented matrix. The method/idea of formal description can be refered to [122].

Definition 1:

Let

$AB(i,j) = A(i,j) \quad i=1\ldots q; \quad j=1\ldots q$
$AB(i,j+q) = B(i,j) \quad i=1\ldots q; \quad j=1\ldots q$

$AB^{k}_{i,j}$ represents the block $AB_{i,j}$ at step k for k = 1,…, q. In the BbGJ algorithm, one block ($AB^{k}_{i,j}$) is modified once and only once per iterative step. Let's use a three-tuple (k,i,j) to represent $AB^{k}_{i,j}$. Then we can use the three-tuple (k,i,j) {k,i=1,...,q and j=k+1,...,k+q} to mark the status of sub-matrix's operations (operation '1' to '7' in sequential BbGJ algorithm), i.e. the three-tuple (k,i,j) means that at the iterative step k, the operation will be made on the row i and column j of the augmented matrix $AB_{i,j}$.

For a fixed step, a set of three-tuples can be utilized to represent the status of operations on augmented matrix AB. Consequently we can use those three-tuples as global signals to control all the operations decided by data dependence in the algorithm.

Definition 2:

Let the binary relation be defined as follows: $X \succ Y$: (X, Y is the block of matrix) if and only if there is an edge from X to Y in the Figure 4.7. The data dependence can thus be represented by the three-tuple presented in definition 1. For all the data dependence existing in Figure 4.7, we have:

Intra-step based data dependence can be written as follows:

✧ $(k, k, k) \succ (k, k, j)$ $\quad k=1\ldots q; \quad j=k+1\ldots q$
✧ $(k, k, k) \succ (k, k, j+q)$ $\quad k=1\ldots q; \quad j=1\ldots k-1$
✧ $(k, k, j) \succ (k, i, j)$ $\quad k=1\ldots q; \quad j=k+1\ldots q; \quad i=1\ldots q$ and $i \neq k$
✧ $(k, k, j+q) \succ (k, i, j+q)$ $\quad k=1\ldots q; \quad j=1\ldots k-1; \quad i=1\ldots q$ and $i \neq k$
✧ $(k, k, k) \succ (k, i, k+q)$ $\quad k=1\ldots q; \quad i=1\ldots q$ and $i \neq k$

Inter-steps based data dependence can be summarized as follows:

✧ $(k-1, k, k) \succ (k, k, k)$ $\quad k=2\ldots q$
✧ $(k-1, k, j) \succ (k, k, j)$ $\quad k=2\ldots q; \quad j= k+1\ldots q$

- ✧ (k-1, k, j+q)≻(k, k, j+q) k=2… q; j=1…k-1
- ✧ (k-1, i, k)≻(k, i, j) k=2…q; j=k+1…q; i=1…q and i≠k
- ✧ (k-1, i, j)≻(k,i, j) k=2…q; j= k+1… q; i=1…q and i≠k
- ✧ (k-1, i, j+q)≻(k, i, j+q) k=2…q; j=1… k-1; i=1… q and i≠k
- ✧ (k-1, i, k)≻(k, i, j+q) k=2… q; j=1…k-1; i=1… q and i≠k
- ✧ (k-1, i, k)≻(k, i, k+q) k=2…q; i=1…q and i≠k

Now we will summarize binary relation obtained above according to operation marked in sequential algorithm of BbGJ and make the executable rules for each operation as follows:

- ✧ (k, k, k)≻(k, k, j) *k=1… q; j=k+1…q*

  $\boldsymbol{A^{k}_{k,k} \leftarrow (A^{k-1}_{k,k})^{-1} \quad B^{k}_{k,k} \leftarrow A^{k}_{k,k}}$
- ✧ (k, k, k)≻(k, k, j) and (k-1, k, j)≻(k, k, j) *k=2…q; j=k+1… q*

  $\boldsymbol{A^{k}_{k,j} \leftarrow A^{k}_{k,k}\, A^{k-1}_{k,j}}$
- ✧ (k, k, k)≻(k, k, j+q) and (k-1, k, j+q)≻(k, k, j+q) *k=2… q; j=1… k-1*

  $\boldsymbol{B^{k}_{k,j} \leftarrow A^{k}_{k,k}\, B^{k-1}_{k,j}}$
- ✧ (k, k, j)≻(k, i, j) and (k-1, i, k)≻(k, i, j) and (k-1, i, j)≻(k,i, j) *k=2…q; j=k+1…q; i=1…q and i≠k*

  $\boldsymbol{A^{k}_{i,j} \leftarrow A^{k-1}_{i,j} - A^{k-1}_{i,k}\, A^{k}_{k,j}}$
- ✧ (k, k, j+q)≻(k, i, j+q) and (k-1, i, j+q)≻(k, i, j+q) and (k-1, i, k)≻(k, i, j+q) *k=2…q; j=1…k-1; i=1…q and i≠k*

  $\boldsymbol{B^{k}_{i,j} \leftarrow B^{k-1}_{i,j} - A^{k-1}_{i,k}\, B^{k}_{k,j}}$
- ✧ (k, k, k)≻(k, i, k+q) and (k-1, i, k)≻(k, i, k+q) *k=2…q; i=1…q and i≠k*

  $\boldsymbol{B^{k}_{i,k} \leftarrow -\, A^{k-1}_{i,k}\, A^{k}_{k,k}}$

4.6 A new parallel programming adapted version of BbGJ

There is no simple recipe for designing parallel algorithms. However, it can benefit from a methodological approach that maximizes the range of options, that provides mechanisms for evaluating alternatives, and that reduces the cost of backtracking from wrong choices [124]. The design methodology allows the programmer to focus on machine independent issues such as concurrency in the early stage of design process, and machine-specific aspects of design are delayed until late in the design process. As suggested by Ian Foster [125], this methodology organizes the design process into four distinct stages which are: partitioning, communication, agglomeration and mapping. Next we will design the parallel paradigm of Gauss-Jordan algorithm according to those four steps presented above.

Because large scale matrix inversion is hard to be executed directly, so partitioning is the first thing to do. In this chapter, large scale Matrix A has been partitioned into q*q blocks. After partitioning, the objects of all the operations in the algorithm focus on sub-matrices, which make a more complex problem to be decomposed into some easy-to-solve sub-problems. No communication between different small sub-problems exists during the process of program execution. The communication is just between matrix A and its sub-matrix. The operation on sub-matrix can read from (or write back to) the corresponding block of Matrix A. We can design this stage of paradigm according to parallel divide and conquer model in which the sub-problems can be solved at the same time, giving sufficient parallelism.

Analysis in the Section 4.2 tells us that inter-steps and intra-step parallelism exist in this algorithm and these parallelisms rest on the data dependence. In this algorithm, data dependence on different blocks determines the sequence of different operations on a sub-matrix. As to inter-step data dependence in Matrix A, there are (n-1)*(n-k) blocks (k is the number of steps) which have the same write-operation (operation ‘5’). They have the same operation and sequence of that and there is no communication between them. This means that the sub-tasks of the algorithm are independent and each processor can execute one part of them. So this kind of operations can use single program multi-data paradigm model to deal with. As to Figure 4.6, the operation represented by “loop 1” shows us that before the execution of operation ‘1’ in step 3 begins, the operation ‘5’ in step 2 must finish. This kind of parallelism we can use data pipelining paradigm to deal with them.

All the write-operations of the algorithm are based on sub-matrix and the results will be returned to the Matrix A. So when all the operations on sub-matrix are end, agglomeration is finished. Hybrid parallel paradigm can generate sub-tasks as many as possible, so you can use any kind of schedule stratagem to map the sub-tasks to computing resources.

Next, a Flow-chart of hybrid parallel paradigm can be seen in Figure 4.8.

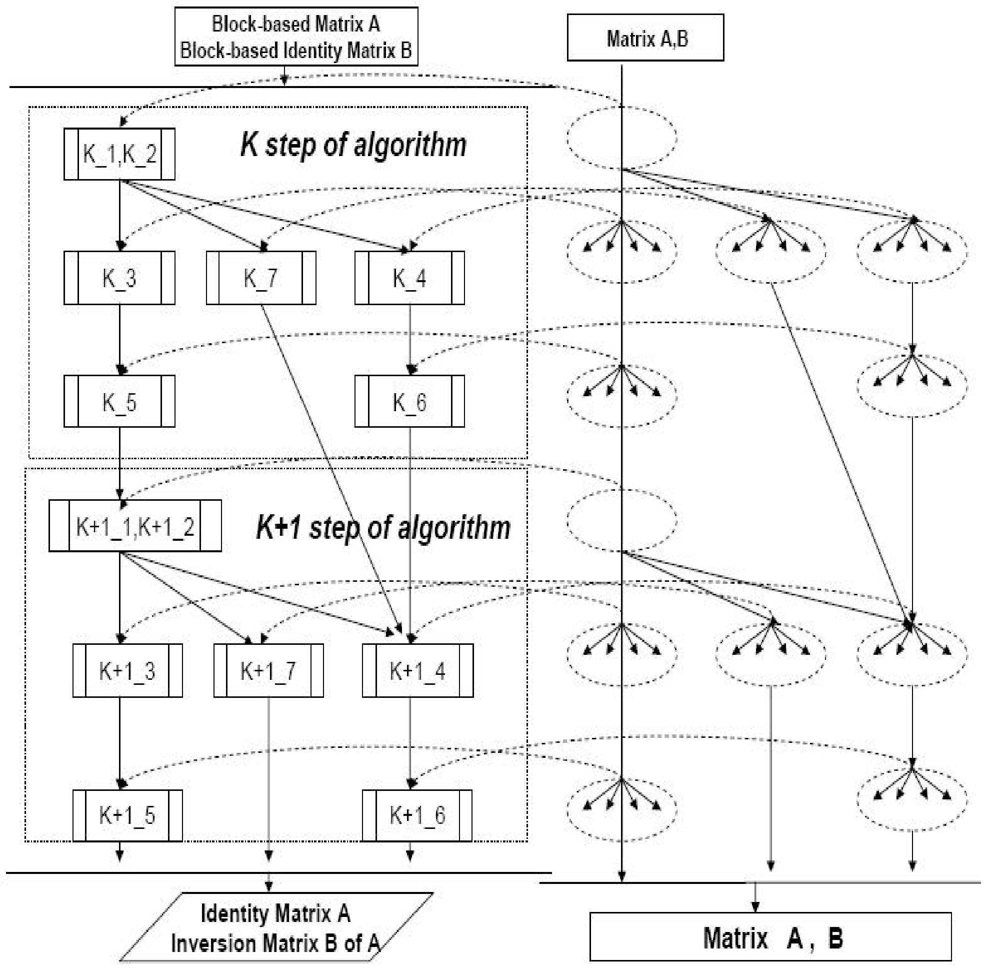


Figure 4.8. Flowchart of new parallel paradigm of BbGJ

Left part is the flowcharts of the paradigm execution and the right part is the corresponding data block operation. Q steps needed to execute in the paradigm and the execution between step 'k' and step 'k+1' is described. In the right parts, 'many arrows' represents these data can be executed simultaneously.

According to formal description, a set of three-touples as the global signals are used to control the execution of algorithm. Through the Figure 4.8, we can know the flowchart of algorithm execution. Then the hybrid parallel paradigm of BbGJ can be presented as follows: (the signal // is meaningless in the algorithm. We just want to use this signal to divide the whole program into several sub-program sections. And those subprogram sections can be executed in parallel on the basis of their conditions are met.)

New Parallel Paradigm of BbGJ

```
    Input: $A,B \longleftarrow I\ (n,n),\ q$
        the logical value of (0,1,1) is set to be true;
          for(i=1;i<=q;i++)    the logical value of (0,1,i) is set to be true;
          for(i=1;i<=q;i++)    the logical value of (0,i,1) is set to be true;
          for(i=1;i<=q;i++)
              for(j=1;j<=q;j++)
                   the logical value of (0,i,j) is set to be true;
     Output: $B=A^{-1}$
   ----------------------------------------------------------------------
     if the logical value of (k, k, k) is true
        then   $A^{k}_{k,k} \leftarrow (A^{k-1}_{k,k})^{-1}$
               the logical value of (k, k, j) is set to be true;
     end if
//
     if the logical value of both (k, k, k) and (k-1,k,j) are true
        then   $A^{k}_{k,j} \leftarrow A^{k}_{k,k} A^{k-1}_{k,j}$   (k=2,...,q; j=k+1,...,q)
               the logical value of (k, k, j) is set to be true;
     end if
//
     if the logical value of both (k, k, k) and (k-1,k,j+q) are true
        then   $B^{k}_{k,j} \leftarrow A^{k}_{k,k} B^{k-1}_{k,j}$   (k=2,...,q; j=1,...,k-1);
                the logical value of (k, k, j+q) is set to be true;
     end if
//
     if the logical value of (k, k, j) and (k-1,i,k) and (k-1,i,j) all are true
         then   $A^{k}_{i,j} \leftarrow A^{k-1}_{i,j} - A^{k-1}_{i,k} A^{k}_{k,j}$  (k=2,...,q;  j=k+1,...,q;  i=1,...,q and i$\neq k$)
                 the logical value of (k, i, j) is set to be true;
     end if
//
     if the logical values of (k, k, j+q) and (k-1,i,j+q) and (k-1,i,k) all are true
         then   $B^{k}_{i,j} \leftarrow B^{k-1}_{i,j} - A^{k-1}_{i,k} B^{k}_{k,j}$  (k=2,...,q; j=1,...,k-1; i=1,...,q and i$\neq k$)
                the logical value of (k, i, j+q) is set to be true.
     end if
//
     if the logical value of both (k, k, k) and (k-1,i,k) are true
         then   $B^{k}_{i,k} \leftarrow -A^{k-1}_{i,k} A^{k}_{k,k}$  (k=2,...,q; i=1,...,q and i$\neq k$)
                 the logical value of (k, i, k+q) is set to be true.
     end if
```

4.6.1 Comparison of different parallel versions of BbGJ

In order to state conveniently we call our new parallel paradigm of BbGJ as "Max-par BbGJ" for its achieving **max**imum-degree **par**allelism in the **BbGJ** algorithm and Melab's and Aouad's version of BbGJ [121] [120] [127] [123] [126] as "Par-par BbGJ" for its just considering **par**tial degree **par**allelism in the **BbGJ** algorithm.

1. Max-Par BbGJ is a new parallel programming adapted version for block based gauss Jordan algorithm and it can adapt to any kinds of programming interface very easily. For example, it can suit for MPI, RPC based interface, high level programming interface and workflow based programming interface. Two kinds of programming methods which adapt to RPC based interface and workflow based interface are adopted in this dissertation. This is the key point in which Max-par BbGJ is different from algorithm in [122] [121] [120]. [122] suits for MIMD based supercomputer architecture and [121][120] suits for MARS which is a kind of network of workstations.

2. Max-Par BbGJ achieves the maximum-degree parallelism of each operator in the BbGJ algorithm. Just described in section 4.2, two kinds of parallelisms which are inter-steps based parallelism and intra-step based parallelism are considered. To algorithm in [121] [120] [127] [123] [126], they just take intra-step based parallelism into consideration.

Intra-step based parallelism means the operations in next iterative step can't be executed before any operation in this iterative step finishes, i.e. if the "operation 5" finishes, but "operation 1" in next iterative step must wait for "operation 7" and "operation 6" until they both finish (see Figure 4.5). But to Max-par BbGJ algorithm, "operation 1" in next iterative step can be executed immediately once the "operation 5" finishes (see Figure 4.7). So less wait/synchronization time is needed in Max-par BbGJ algorithm. As a result, the greatest degree of parallelism on tasks is achieved in Max-par BbGJ algorithm.

4.6.2 Theoretical analysis on BbGJ

In Max-par BbGJ, matrix A is partitioned into a matrix of q*q blocks of dimension n and n=N/q. There are q+1 operations in each iterative step and n*n elements in each sub-matrix. The nature of the Gauss Jordan algorithm makes that even if the initial matrix to be inverted is sparse and it becomes dense matrix after no more than one step [120]. Each sub-matrix has n^2 elements and its storage space should be n^2*64 bits. The tasks in BbGJ is based on 2 or 3 or 4 sub-matrix, so data migration from one task to another is $2n^2$ or $3n^2$ or $4n^2$ (64 bits).

Next we will analyze the number of "operation *l*" (*l* =1, 2, 3, 4, 5, 6, 7 see the detail numbers in sequential algorithm of BbGJ) in the k iterative step (k=1, … , q) during the process of algorithm execution.

- The number of "operation 1" is 1.
- The number of "operation 2" is 1.
- The number of "operation 3" is q-k.
- The number of "operation 4" is k-1.
- The number of "operation 5" is (q-k)*(q-1).
- The number of "operation 6" is (k-1)*(q-1).
- The number of "operation 7" is q-1.

According to Figure 4.8, there are data dependence between "operation 3" and "operation 5". So under the condition of ensuring full concurrency of "operation 3" and "operation 5", the number of computing peers needed is Max {(q-k); (q-1)*(q-k)} = (q-1)*(q-k). In the same way we can get the number of computing peers needed for other operations. The number of computing peers needed can be summarized as follows:

- To "operation 1" and "operation 2", the number of computing peers needed is 1.
- To "operation 3" and "operation 5", the number of computing peers needed is Max {(q-k); (q-1)*(q-k)} = (q-1)*(q-k).
- To "operation 4" and "operation 6", the number of computing peers needed is Max {(k-1); (k-1)*(q-1)} = (k-1)*(q-1).
- To "operation 7", the number of computing peers needed is (q-1).

According to the analysis above, we can get N_cp (the minimum number of computing peers) and V_dm (total volume of data migration during the process of algorithm execution; here, V_dm is got without the help of optimal technology on data migration).

$$\begin{aligned} N\_cp &= \mathrm{Max}\ \{(q-k); (q-1)*(q-k)\} + \mathrm{Max}\ \{(k-1); (k-1)*(q-1)\} + (q-1) \\ &= (q-1)*(q-k) + (k-1)*(q-1) + (q-1) \\ &= (q-1)*(q-k+k-1+1) \\ &= q*(q-1) \end{aligned}$$

$$\begin{aligned} V\_dm &= q*2n^2 + q*(q-k)*3n^2 + q*(k-1)*3n^2 + q*(q-1)*(q-k)*4n^2 + \\ &\quad q*(k-1)*(q-1)*4n^2 + q*(q-1)*\ 3n^2 \\ &= n^2 * (2q + (q-k)\ (3q+4q*(q-1)) + (k-1)\ (3q + 4q*(q-1)) + 3*q*(q-1)) \\ &= n^2 * (2q + q*(q-k)\ (3+4(q-1)) + q*(k-1)\ (3 + 4*(q-1)) + 3*q*(q-1)) \\ &= n^2 * q\ (2 + 3*(q-k) + 3*(k-1) + 4*(q-1)*(q-k) + 4*(k-1)*(q-1) + 3*(q-1)) \\ &= n^2 * q\ (4*(q-1)^2 + 6*(q-1) + 2) \\ &= 2*n^2 *q\ (2*(q-1)^2 + 3*(q-1) + 1) \\ &= 4*q^2*(q-1)*\ n^2\ (64\ \text{bits}) \end{aligned}$$

4.7 Evaluation of Max-par BbGJ

This section will evaluate the performance of Max-par BbGJ and we will compare Max-par BbGJ with Par-par BbGJ [121] [127] [126].The experimental environments can be described as follows:

Table 4.1. Parts of resources in Grid'5000 platform.

Site	Nodes	CPU/Memory
Nancy	120	2 × Inter xeon , 1.6GHz/2GB
Nancy	47	2 × AMD opteron, 2GHz/2GB
Lyon	70	2 × AMD opteron, 2.4GHz/2GB
Bordeaux	93	2 × AMD opteron, 2.6GHz/2GB
Orsay	216	2 × AMD opteron, 2.0GHz/2GB
Rennes	99	2 × AMD opteron, 2.0GHz/2GB

4.7.1 Block-size fixed and block-count changed

Experiment motivation: test the performance of Max-par BbGJ with block-size fixed and block-count changed. Experiment environments: 100 nodes used of cluster bordereau in Bordeaux site, France. Experiment data: the block-size of matrix is fixed as 500*500 and 1500*1500. We change the block-count as follows: 2*2, 3*3, 4*4, 5*5, 6*6, 7*7 and 8*8.

This experiment is based on the situation that there are enough computing resources to be used. This is to say, all the tasks can be executed immediately when its executable condition is met. 100 nodes are enough for the situation of block-count is 8*8 and the reason is that 8*(8-1) < 100. The Figure 4.9 and Figure 4.10 show us the relationship between the elapsed time and the different block-counts of sub-matrix. We can conclude that the elapsed time of Max-par BbGJ is shorter than that of Par-par BbGJ whether the block-size of sub-matrix is fixed on 500*500 or 1500*1500. We also can know that with the increase of block-count, the better performance can be obtained using Max-par BbGJ than that using Par-par BbGJ. The reason is that with the increase block-count of matrix, the number of iterative steps augments. More wait time is needed to synchronize at the end of each iterative step in the Par-par BbGJ, while no wait time is needed in Max-par BbGJ Algorithm. From those figures, we also know that with the increase of block-size, the better performance can be achieved for

Max-par BbGJ. We can explain that with the increase of block-size, the computing time of processing those sub-matrix becomes longer and more time is needed to wait for synchronize at the end of each iterative step.

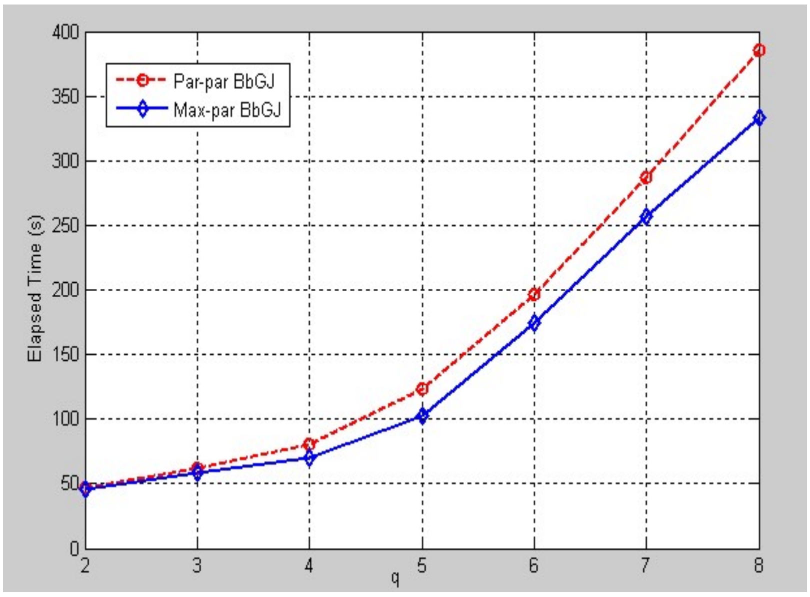


Figure 4.9. Elapsed time with block-count changed and block-size fixed as 500*500

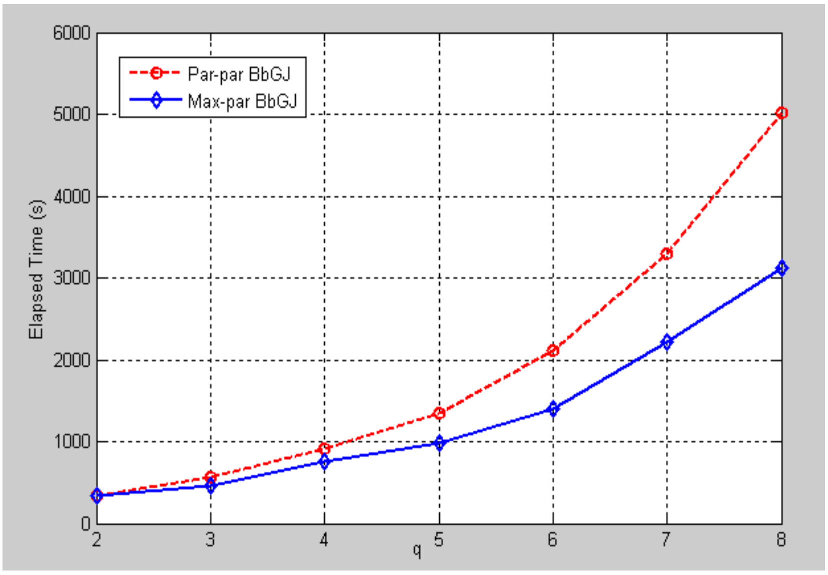


Figure 4.10. Elapsed time with block-count changed and block-size fixed as 1500*1500

4.7.2 Block-count fixed and block-size changed

Experiment motivation: test the performance of Max-par BbGJ with block-size changed and block-count fixed. Experiment environments: 100 nodes used of cluster of grelon in Nancy site, France. Experiment data: the block-count of matrix is fixed on 6*6. We change the block-size of sub-matrix from 250*250, 500*500, 1000*1000, 1250*1250 and 1500*1500.

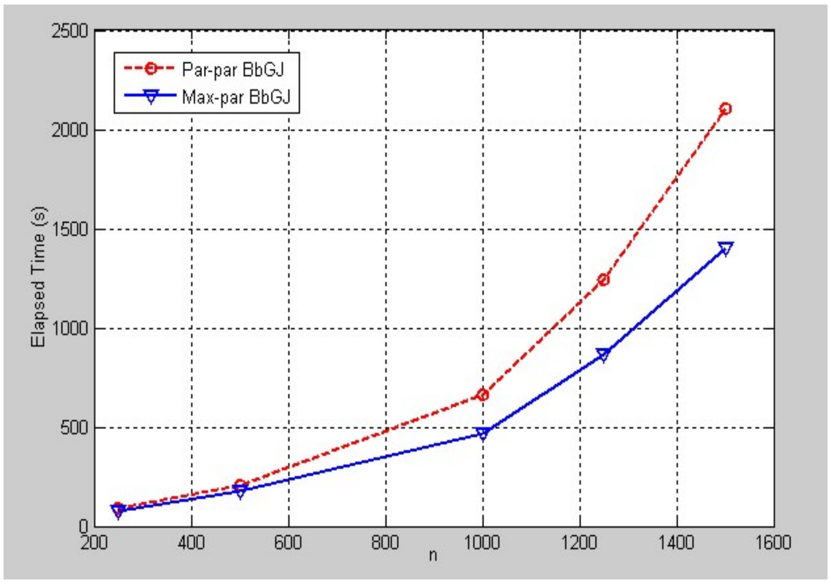


Figure 4.11. Elapsed time with block-size changed and block-count fixed as 6*6

This experiment is based on the situation that there are enough computing resources to be used. This is to say, all the tasks can be executed immediately when its condition is met. 100 nodes are enough for the situation of block-count is 8*8 and the reason is that 8*(8-1) < 100. From the Figure 4.11, we can find when the block-size is not large, the elapsed time of Max-par BbGJ is almost the same with that of Par-par BbGJ. The reason is that when the block-size is small, communication time plays an important role in the overall consumed time. Less time for synchronization is needed and the advantage of Max- par BbGJ algorithm is not obvious. But when the block-size is more than 500*500, the performance of Max-par BbGJ is better than Par-par BbGJ and with the increase of block-size, the advantage of Max-par BbGJ becomes more obvious. The reason is that, less wait time is needed in Max-par BbGJ algorithm comparing to that in Par-par BbGJ algorithm. (refer to analysis in the second point of section 4.6.1)

4.7.3 Block-count changed and block-size changed

Experiment motivation: test the performance of Max-par BbGJ with block-size changed and block-count changed. Experiment environments: 100 nodes used of cluster of grelon in Nancy site, France. Experiment data: change the block-count of sub-matrix from 2*2, 3*3, 4*4, 5*5, 6*6, 7*7 and 8*8 and change the block-size of sub-matrix from 500*500, 1000*1000 and 1500*1500.

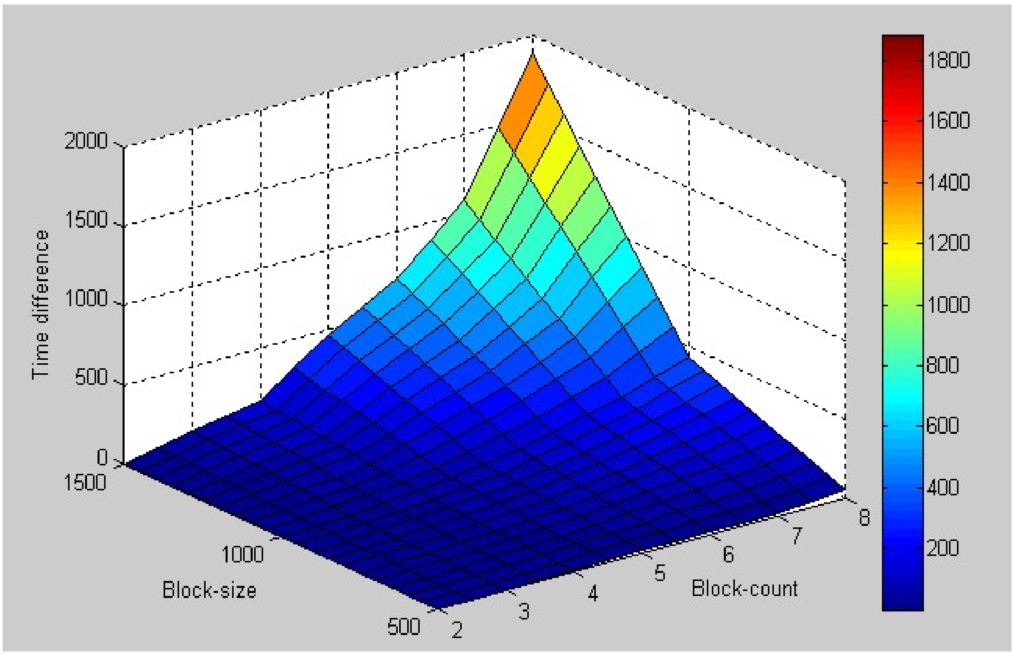


Figure 4.12. Time difference between two algorithms with block-size changed and block-count changed

This experiment is based on the situation that there are enough computing resources to be used. This is to say, all the tasks can be executed immediately when its condition is met. 100 nodes are enough for the situation of block-count is 8*8 and the reason is that 8*(8-1) < 100. The Figure 4.12 shows us the general trends of time difference with changing block-size and block-count of sub-matrix. With the same block-size, the time difference becomes larger with increasing block-count of sub-matrix. The reason is analyzed through Figure 4.11. Another trend is that the time difference also becomes larger with the increase of block-size of sub-matrix. The reason is that the larger dimension of sub-matrix will cost more computing time. As a result, more wait time is needed to synchronize in Par-par BbGJ. Through the Figure 4.12 we can know the gap becomes larger with the increase of block-count and block-size. What we want to emphasize here is when the dimension of matrix is very large (whether through adding block-count or enlarging block-size of sub-matrix), the advantage of Max-par BbGJ becomes more obvious.

4.7.4 Situation of no enough computing resources

Experiment motivation: test the performance of Max-par BbGJ under the situation of no enough computing resources. Experiment environments: 20 nodes used of cluster of grelon in Nancy site, France. Experiment data: change the block-count of sub-matrix as follows: 2*2, 3*3, 4*4, 5*5, 6*6, 7*7 and 8*8 and block-size of sub-matrix is fixed on 500*500.

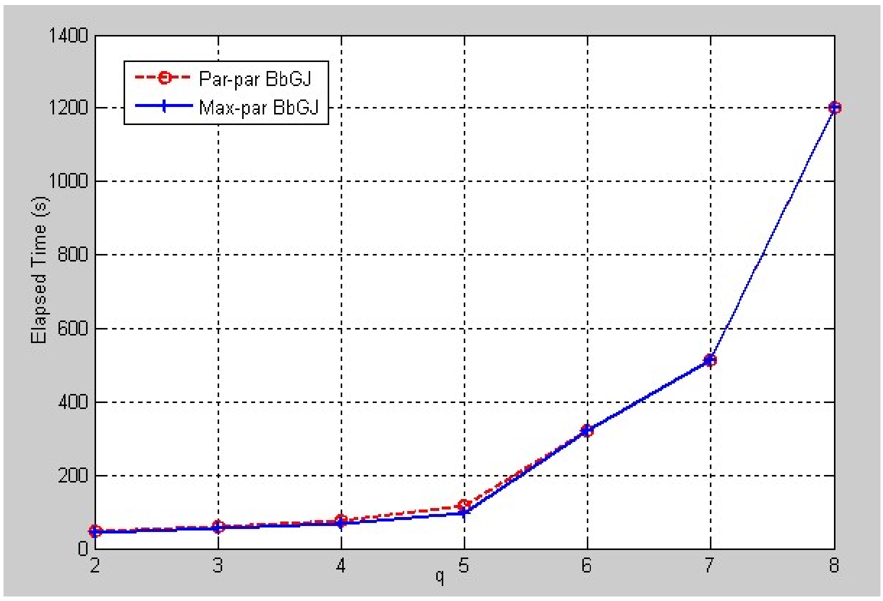


Figure 4.13. Situation of no enough computing resources

This experiment is based on the situation that there are not enough computing resources available to be used. This is to say, not all the tasks can be executed immediately when its condition is met. 20 nodes are enough for the situation of block-count is below 6*6 and the reason is that 6*(6-1) > 20 and 5*(5-1) ≤ 20.

The Figure 4.9 shows us the Max-par BbGJ has better performance that Par-par BbGJ under the condition of that there are enough computing resources can be used. The Figure 4.13 demonstrates the comparison between Max-par BbGJ and Par-par BbGJ under the situation of no enough computing resources. When there are not enough computing resources, the advantage of Mar-par BbGJ which is achieving maximum parallelism between tasks, can't be made full use of. Thus the performance of Max-par BbGJ is almost the same with that of Par-par BbGJ.

4.7.5 Performance in Grid environment

Experiment motivation: test the performance of Max-par BbGJ algorithm in the grid environment. Experiment environments: 100 nodes used. Grelon cluster, Nancy site; Paravent cluster and Paraquad cluster in Rennes site; Sagittaire cluster in Lyon site; Bordereau cluster in Bordeaux site. 20 nodes cores are used in each cluster. Experiment data: the block-size of matrix is fixed on 1500*1500 and we change block-count as follows: 2*2, 3*3, 4*4, 5*5, 6*6, 7*7 and 8*8.

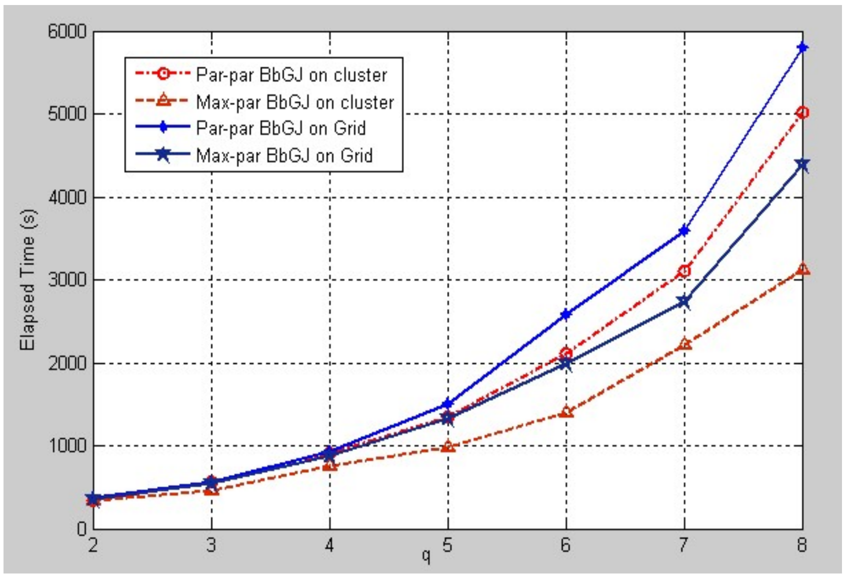


Figure 4.14. Performance comparison between cluster and Grid environment

From the Figure4.14, we can know clearly that the same conclusion can be made in the Grid environment, and the reason is the same. But the elapsed time is longer on Grid than that on cluster, which is determined by the character of Grid environments (lower speed network than cluster, more complex schedule stratagem than cluster). An interesting case is that computation time of Par-par BbGJ based on cluster is longer than that of Max-par BbGJ on Grid. Two reasons can explain that. On one hand, the performance of Max-par BbGJ is far better than that of Par-par BbGJ. On the other hand, it is that high speed network between sites in Grid'5000 platform makes Max-par BbGJ algorithm have better performance.

4.8 Conclusion

This Chapter makes further research on parallelism of BbGJ based on former researches. The key point of our research is that we analyze all the possible parallelisms in BbGJ and based on those available parallelisms, a new parallel programming adapted version of BbGJ which can adapt to any programming interface very easily and make full use of computing resources, is proposed in this chapter.

The detail of how to realize that can be described as follows: the chapter takes all the parallelisms which are intra-iterative step based parallelism and inter-iteratives step based parallelism, into consideration. Analysis has been made on data dependence between operations. Based on works done above, we propose a three-tuple to describe those data dependences. According to those three-tuples, a series of executable rules for each operation in BbGJ are made. Finally, we present a parallel paradigm according to rules made above and design model of parallel programming. The advantage of our parallel paradigm is that it can make operations in BbGJ can be concurrent fully and thus improve the efficiency of parallel BbGJ. The parts of experiment testify our viewpoint.

Block based Gauss Jordan algorithm are widely used in large scale scientific computing and it will be used as ***an example*** to test our experiment environment in the following chapter of this dissertation.

Chapter 5

Large scale scientific computing on Grid and Desktop Grid environments
—— with block based Gauss Jordan as an example

5.1 Motivation

This chapter is to make further research on large scale scientific computing on Grid and Desktop Grid environments. The related issues include programming method, overhead of middleware, data anticipate migration, the influence from parallel granularity of tasks on different experiment environments and schedule mechanism. Block based Gauss Jordan algorithm as a real example of large scale scientific computing, is used to evaluate those issues presented above. Three middlewares which are OmniRPC for Grids, XtremWeb for Desktop Grids and YML a workflow based high level programming framework, are adopted.

5.2 Programming model

As well known to us all, programming method is a key issue of scientific computing. In fact, early supercomputer has its owner programming interface and it is hard to transplant developed code to other supercomputers. The emergence of MPI library has changed the programming method of high performance systems. It is MPI that makes code migration be possible. Then with the development of programming language, RPC based programming interface are popular for making large scale scientific computing. The section will present three kinds of programming interfaces and compare with them.

5.2.1 Programming with YML

YML has a simple and high level interface. The development of an YML application is based on components approach and three kinds of components are needed to develop when running an application program based on YML. The three components are “Graph component”, “abstract component” and “implement component”. “Abstract component” defines the communication interface with the other components. This definition gives the name and the communication channels with other components. Each channel corresponds to a kind of data operation type which can be data input, data output or both. “Implementation component” is the implementation of an “abstract component”. The “implementation component” is developed using common languages like C or C++ or Java. Those two parts of works can be done by computer engineers in advance or invoke public-domain libraries such as ScaLAPACK, LAPACK and BLAS3. Once those components are registered in YML, they can be reused at any time. Graph component carries a graph expressed in YvetteML and it provides the parallel and sequential parts of an application and the synchronization events between dependent components. It is pseudo code based program interface for end users and it is very easy way for scientific researchers to develop their application.

To understand the description above, we will use block based Gauss Jordan algorithm as a real example to explain it. There are seven operations in BbGJ algorithm. To adapt BbGJ algorithm to YML, three kinds of components for the seven operations are needed to develop and those operations can be divided into four kinds:

- inversion: to inverse one matrix block. (Operation 1)
- prodMat: to compute the two blocks product.(operation 3; operation 4)
- mProdMat: to compute the negative of two blocks product (operation 7)

- ProdDiff: to compute the difference between one block and a block matrix product (operation5; operation6).

We will take the "matrix multiplication" (operation 3 and operation 4 in sequential BbGJ algorithm) as an example to show that.

As just described, abstract component is just to define the communication channels with other components and in this program, suiting for YML can be written as follow:

```
<?xml version="1.0" ?>
      <component type="abstract"    name="prodMat"      description="Compute the product of two matrix"
>
            <params>
                  <param name="matrixBkk" type="Matrix" mode="in" />
                  <param name="matrixAki" type="Matrix" mode="inout" />
                  <param name="blocksize" type="integer" mode="in" />
            </params>
      </component>
```

From the example, we can know more details in how to develop abstract component. The operation of "matrix multiplication" has three parameters which are "matrixBkk", "matrixAki" and "blocksize". The parameter modes of "matrixBkk" and "blocksize" are input. The mode of parameter "matrixAki" is input and output. The type of parameters is also needed to mark and it should be supported by YML. In summary, abstract component is just to tell YML the properties of operation and doesn't show how to execute the operation in detail.

As to how to execute the operation, it can be dealt with by "implementation component". There is an "implementation component" which is one-to-one correspondence with "abstract component". An "implementation component" can be developed using C, C++ or java. An example of "matrix multiplication" developed using C can be written as follows:

```
<?xml version="1.0"?>
<component   type="impl"    name="prodMat"    description="Implementation component of prodMat"
          abstract="prodMat">
      <impl lang="CXX">
            <header>   </header>
              <source>
                <![CDATA[
                The detail program developed using special programming language such as C, C++ or
Java
                 ]]>
                </source>
            <footer >
      </impl>
</component>
```

In the framework of "implementation component", we should point out its type and name. Here what we should stress is the parameter of "abstract" and it must be the name of the parameter of "name" in "abstract component". You should point out what language you will choose to develop the component in the parameter of "lang". Then, you can develop the detail program using the language you chose. Here is an example about matrix multiplication developed using c.

```
int i,j,k;
double ** tempMat;
tempMat = (double **)malloc(blocksize * sizeof (double *));
            for (i = 0 ;i < blocksize ; i++)
      {   tempMat[i] = (double *)malloc( blocksize *sizeof (double));
                     for (j = 0 ;j < blocksize ; j++)
                             tempMat[i][j] = 0.0;    }
   for(k = 0 ; k< blocksize ; k++)
        for (i = 0 ;i <blocksize ; i++)
              for (j = 0 ;j <blocksize ; j++)
               tempMat[i][j] = tempMat[i][j] + matrixBkk.data[i][k] * matrixAki.data[k][j];
         for (i = 0 ;i < blocksize ; i++)
                     for (j = 0 ;j < blocksize ; j++)
                             matrixAki.data[i][j] = tempMat[i][j];
for (i = blocksize-1 ; i>=0 ; i--)
       free(tempMat[i]);
       free(tempMat);
```

So far, we have developed "abstract component" and "implementation component". But we want to stress two points:

1. The developed components can be reused very easily. It can be used not only in block based Gauss Jordan algorithm, but also in other algorithm which includes matrix multiplication. This is to say, the developed components have nothing to do with application program.
2. If you are not good at programming, you can let programmer help to develop those components. Or you also can invoke the related functions from public/ third party libraries.

We have developed the "abstract component" and "implementation component". Those two components are just for "operation 3" and "operation 4" of BbGJ algorithm. Other three kinds of components in BbGJ can be developed in the same way. The next thing to consider is the executable way of those operations. This is to say, those operations should be executed in parallel or in sequence and if in parallel, how and when to execute each operation is what we must consider.

The "Graph component" is the interface for users to control how to execute the application program. The interface is based on YvettML language which is a XML based description language. It can control when to execute the "operations" in BbGJ algorithm through "wait" and "notify" events. YML tells computer how to execute the "operations" in BbGJ algorithm through its series of control structures such as “structure parallel”, “structure sequence”, “condition” and “structure loop”. An example of “matrix multiplication” developed using YvettML can be written as follows:

```
<?xml version="1.0"?>
<application name="gauss_jordan">
<graph>
par
          par(k:=1; blockcount-1) (j:=0; blockcount-1)
             do
                if (k gt j) then
                     wait(flag[k][k][k] and flag[k-1][k][j+blockcount]);
                     compute prodMat(A[k][k],B[k][j],blocksize);
                     notify(flag[k][k][j+blockcount]);
                endif
            enddo
endpar
</graph>
</application>
```

In fact, users just use the statement “**compute prodMat(A[k][k], B[k][j], blo cksize);**” to invoke “implementation component”. It is a very simple task to finish and users even need know nothing about programming knowledge. The event “wait” and “notify” is decided by parallel algorithm of BbGJ algorithm. Different parallel algorithm has different events which are used to decide how to execute “operations” in BbGJ algorithm. Figure 5.1 shows us all the events of Par-par BbGJ during its executable process and Figure 5.2 describes all the events of Max-par BbGJ during its running process. According to those events and “control structure” provided by YvettML, users can program very easily. See pseudo code based program developed using YML in appendix A (at the end of this dissertation). Through comparison the program in appendix A and new parallel programming version in Chapter 4.6, you can find the advantage of programming in higher level interface. Next, we will show some other program interfaces provided by XtremWeb and OmniRPC.

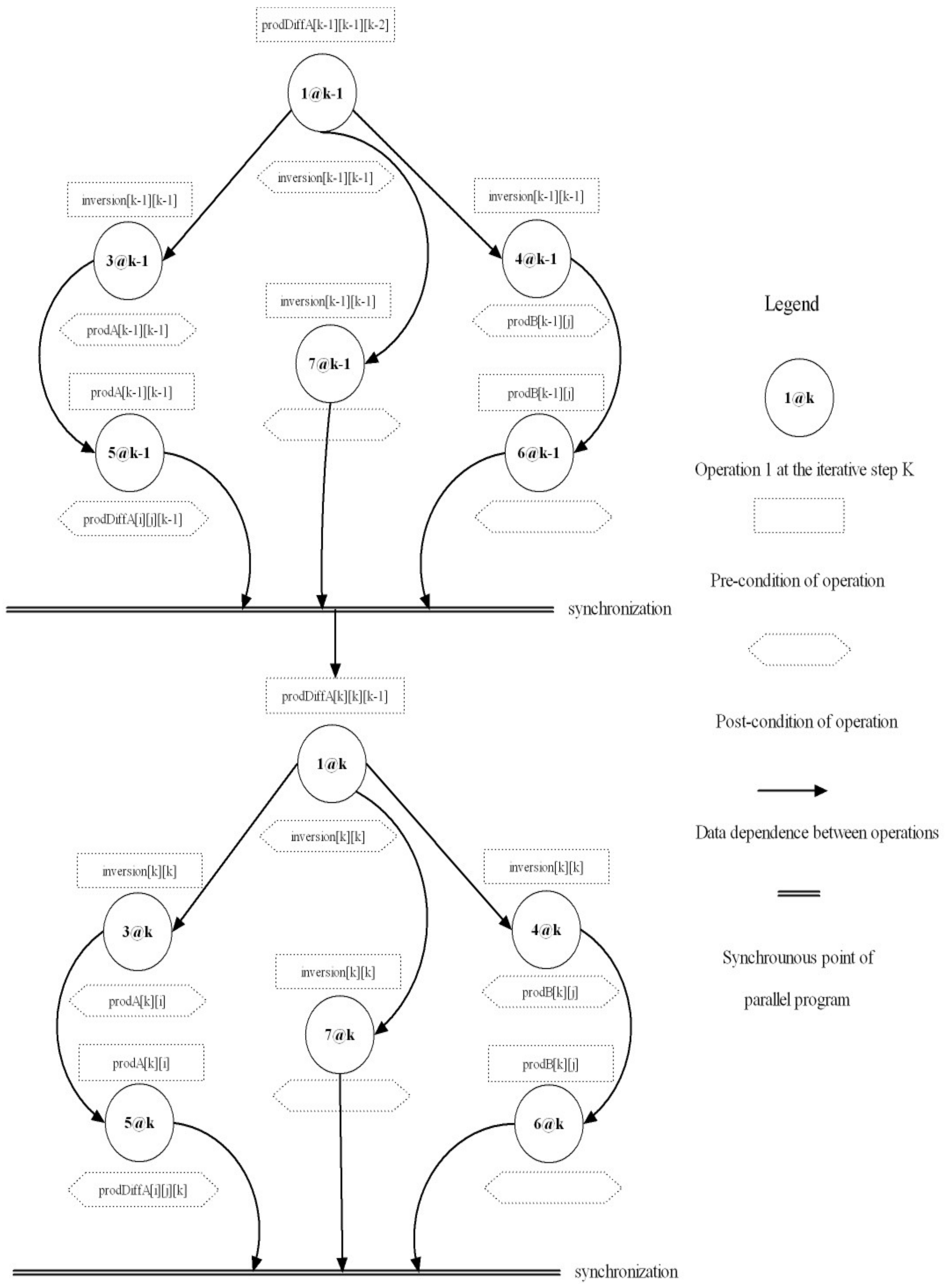


Figure 5.1. Events and execution in Par-par BbGJ

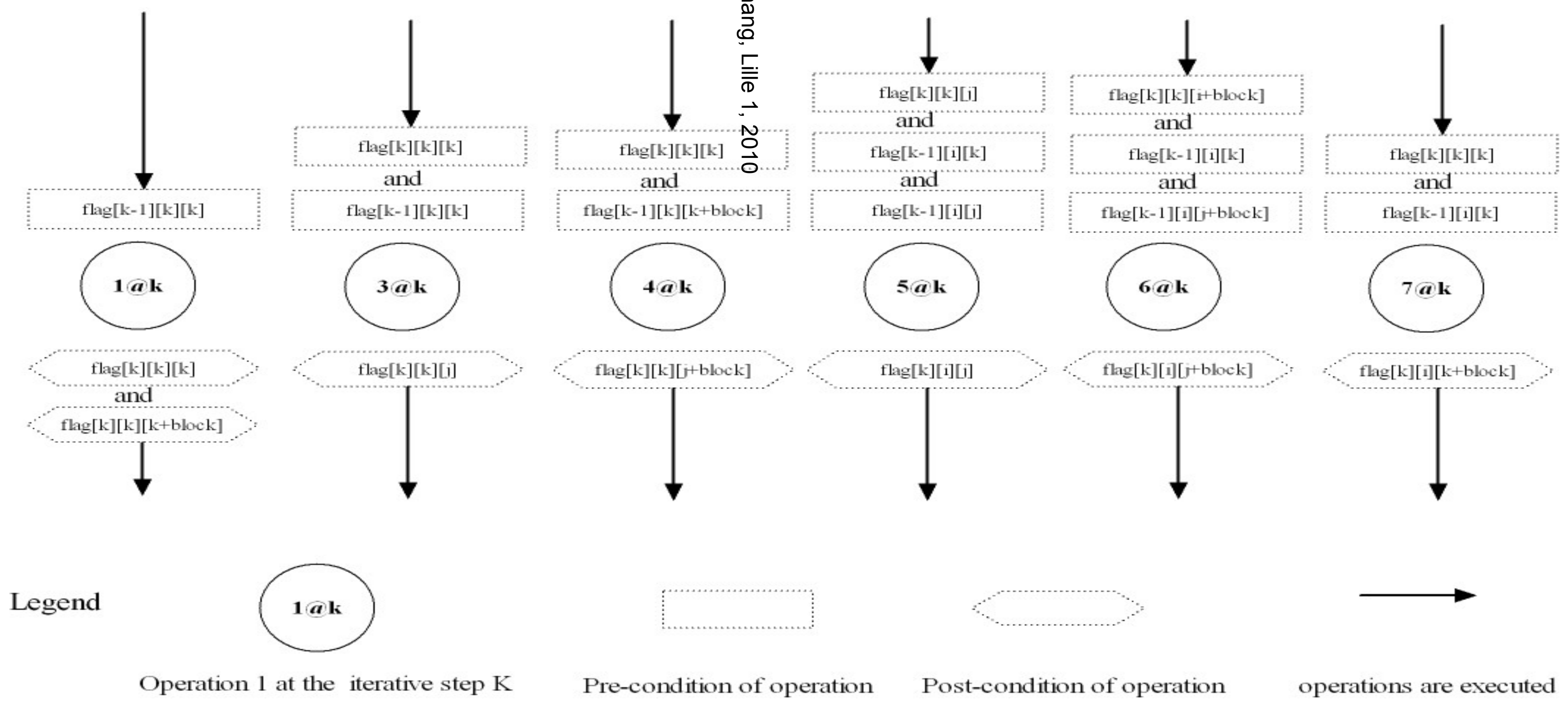


Figure 5.2. Events and execution in Max-par BbGJ

5.2.2 Programming with XtremWeb

Desktop Grids are to collect idle cycle of Internet connected computers which may be widely distributed across the world and more and more scientific researchers tend to use Desktop Grids for its huge process power and lower cost. XtremWeb is a middleware for Desktop Grid environment. It is based on Client-Coordinator-Worker model. Client is to submit requests, worker is for executing them and coordinator plays the role of intermediary between clients and workers. To XtremWeb, coordinator encapsulates different services (scheduler, results server, applications repository). To make application run on the XtremWeb, users need to know the following information:

1. Application management with the XtremWeb client
wapps: list the applications present in the platform. This parameters need no argument.
--xwaddapp <Application Name> <CPU> <OS> <binary> insert a new application binary in the platform;
--xwrmapp <Application Name> removes an application and all its associated job from the platform;

2. Jobs management with the XtremWeb client.
Here are the client parameters dedicated to job management.
--xwstatus [jobUID, jobUID...] : list job status.
--xwtasks [jobUID, jobUID...] : list status of job instances.
--xwsubmit <Application Name> [--xwenv <zip file>] [--xwlabel <label>] [application parameters] [<input file>] : insert a new job.
--xwrm [jobUID, jobUID...] : delete jobs.

3. Results management with the XtremWeb client.
Here follow the client parameters dedicated to results management.
--xwresult [jobUID, jobUID...]: retrieve job results.

4. Jobs management with the XtremWeb client.
Here follow the client parameters dedicated to job management.
--xwstatus [jobUID, jobUID...] : list job status.
--xwtasks [jobUID, jobUID...] : list status of job instances (i.e. job and associated running worker).
--xwsubmit <Application Name> [--xwenv <zip file>] [--xwlabel <label>] [application parameters] [<input file>] : insert a new job.
--xwrm [jobUID, jobUID...] : delete jobs.

So many parameters for submit tasks and much information of platform are necessary to know about for users. So it is complex for users to run their applications on XtremWeb based platform. Also developing programming based on XtremWeb is not easy. We still want to take "operation 3" in BbGJ algorithm as example to show that. The program can be written as follows:

```
Int nojob=0
For (int l=step+1; l<block-size; l++)
   String[] inresults=new String[] {"A_"+step+1, "B_"+step+step};
   Zipper.setFilename ("A_"+step+1+".zip");
   Zipper.zip(inresults);
   String cmdline= "A_"+step+1+ " " + "B_"+step+step+ "" + "A_" +step+1;
   Nt[nojob]= new MobileWork ("prodMat _" + nojob +step);
   Nt[nojob].setServer (GaussJordan.config.getCurrentServer());
   Nt[nojob].serApplicationName ("prodMat");
   Nt[nojob].setCmdLine (cmdline);
   Nt[nojob].setDirin ("./", "A_" +step+1+ ".zip");
   If (GaussJordan.comm.submitJob(nt[nojob], session.getName(), group.getName()) == Null)
      { System.out.println ("can't submit " + nt[nojob].getUID());
        Debug.Info ("job" + nt[nojob].getUID() + " : can't submit ");
      }
   Else
      { System.out.println("job" + nt[nojob].getUID() + " submitted ");
        Debug.Info ("submitted" + nt[nojob].getUID());
        Nojob++;
      }
```

From the example above, we can find it is not easy for end users to develop application program using XtremWeb interface directly. And to YML, users just write an invoke function and it is very simple for users to use, especially for non-professional computer users.

5.2.3 Programming with OmniRPC

OmniRPC is designed to allow easy development and implementation of parallel scientific applications for distributed and Grid environments. OmniRPC is an evolution of Ninf, since it inherits the API and basic structure from it. It is mainly designed for multi-threaded clients based on a master-worker structure. Its implementation is through a thread-safe based remote procedure call.

The architecture of OmniRPC is composed of a client application and various remote computational hosts, which execute the remote procedures. Remote locations can be connected via a local area network or over a wide-area network. The client application can be written in various different languages, such as FORTRAN, C and C++, and the parallel execution in the client can be obtained by using direct thread libraries, such as the POSIX thread, or the OpenMP API. The interface to a remote function is described by the Ninf IDL. In OmniRPC, the remote executions are managed by the use of remote shell (rsh) for local distributed environments and by the use of Globus and ssh for Grid environments.

To program with OmniRPC, the following steps have to done:

- On the remote host, create a remote executable program of the remote function and register it. It can be divided into 3 parts which are:
    1. create the IDL file which defines interfaces.
    2. generate a remote executable module from the IDL file with the "omrpc-cc" program.
    3. register with omrpc-register.

- On the client host, create hosts.xml which describes the remote host.
- Write the client program, and compile with omrpc-cc.
- Execute the client program, specifying hosts.xml.

We still use the matrix multiplication as an example to show how to program with OmniRPC.

First, write remote executable program:

1. Write a IDL for matrix multiplication as follows:

```
Define prodMat(IN int size, INOUT double C[size][size],IN double A[size][size], IN double B[size][size])
{
   int i = 0;    int j = 0;    int k = 0;
   tempMat = (double **)malloc(size * sizeof (double *));
       for (i = 0 ; i < size ; i++)
       {    tempMat[i] = (double *)malloc(size * sizeof (double));
                for(j=0; i < size ; j++)
                    tempMat[i][j] = 0;
       }
   for(k = 0 ; k < size ; k++)
       {    for(i = 0 ; i < size ; i++)
                {    for(j = 0 ; j < size ; j++)
                     {
                     tempMat[i][j] = A[i*size+k] * B[k*size+j] + tempMat[i][j] ;
                     }
                }
       }
     for (i = 0 ; i < size ; i++)
              for (j = 0 ;j < size ; j++)
                     C[i*size+j] = tempMat[i][j];
     for(i=size-1; i>=0; i--)
              free(tempMat[i]);
   free(tempMat);
}
```

2. generate a remote executable module from the IDL file:

   omrpc-cc prodMat.idl –lm
   Then generate the file "prodMat.rex"

3. register with omrpc-register

   omrpc-register -register prodMat.rex

Second, write client program:

To write the client program, it is necessary for users to learn and use those OmniRPC APIs. Some APIs are:

```
void OmniRpcInit(int *argc, char **argv[]);
void OmniRpcFinalize(void);
int OmniRpcCall(char *entry_name,...);
int OmniRpcCallV(char *entry_name,va_list ap);
OmniRpcRequest OmniRpcCallAsync(char *entry_name,...);
void *OmniRpcCallAsyncV(char *entry_name,va_list ap);
void OmniRpcWait(OmniRpcRequest req);
int OmniRpcProbe(OmniRpcRequest req);
void OmniRpcWaitAll(int n, OmniRpcRequest reqs[]);
int OmniRpcWaitAny(int n, OmniRpcRequest reqs[]);
```

Then, the client program will be developed based on those APIs. The client program can be:

```
int main(int argc, char * argv[])
{
        OmniRpcRequest reqsProdMatA[blockcount];
        OmniRpcRequest reqsProdMatB[blockcount];
            typedef  double Matrix[blocksize][blocksize];
               omrpc_debug_flag = 0
                for(i=k+1; i < blockcount; i++)
                {    if(i==(k+1))
                          {OmniRpcWait(reqsInvers[k]);}
                             reqsProdMatA[i]=OmniRpcCallAsync("prodMat",blocksize,B[k][k],A[k][i]);
                }
              fprintf(stderr,"End Gauss-Jordan\n");
              end=time(NULL);
         fprintf(stderr,"Time of run %f\n",difftime(end,begin));
             for( i=blockcount-1; i >=0; i--)
               {    free(B[i]);    free(A[i]);
                 }
             free(B); free(A);
        OmniRpcFinalize();
        return 0;
}
```

Then, compile the program with command “omrpc-cc”

```
omrpc-cc –o prodMat.exe prodMat.c
```

Third, specify hosts.xml and execute the client program:

hosts.xml can be created as follows:

```
<?xml version="1.0" ?>;
<OmniRpcConfig>;
    <Host name=" " />;
</OmniRpcConfig>;
```

Then use the following command to execute the program:

```
prodMat.exe --hostfile hosts.xml
```

From the process of programming with OmniRPC, we can find that it still difficult for end users to program using OmniRPC directly. Before using OmniRPC, users need cost long time to learn how to use it. But YML provide an intuitive way to program and users can adapt their application program to Grids/Desktop Grids environments more easily.

5.2.4 Summary and conclusion

From the description, we can summarize that:

Programming with OmniRPC and XtremWeb require users to know about their APIs and computing environments they want to use. This is to say, user have to deal with something before gridificating their application. Firstly, users must know how to adapt their application to Grid /Desktop Grid environment through APIs OmniRPC/ XtremWeb provided. Secondly, users also must know more information about platforms. They need to know the status of computing resources and how to allocate tasks to related computing resources. The process of using XtremWeb/ OmniRPC is complex for end users. Last, but not the least, it is hard to reuse the developed code.

YML provides end users a higher level programming interface which is pseudo-code based. The advantage of YML is that it succeeds in separating “operation functions” from “control flow”. “Operation functions” (for example “operation 3” in BbGJ algorithm) can be developed by third party or invoke related function from common libraries. At the same time, this separation make those “operation functions” can be reused very easily. End users need not know about how to program those “operation functions”. The interface of YML is just to describe the “control flow” of application program and it is independent of program language and underlying execution environments. So, if users know more detail about application itself, it is very easy for end users to program with YML which provides a description based programming interface.

From the analysis above, we can conclude that YML provide end users with a very easy-of-use interface. The components developed in YML can be reused very easily. Programming with YML can save lot of costs in time (high level interface makes program be very easier) and money (reused component, once develop and many use) for end users. Here we also have to point out that, YML is based on OmniRPC/ XtremWeb and some overheads are added to the platform. In the next section, we will discuss the overhead by adopting YML framework.

5.3 Overhead of middleware

From the description in the last section, we can know about two points:

- YML supports the separation of “control flow” and “executable functions” and it helps end users just focus on parallel algorithm itself without considering how to adapt their application to detail executable environments. Based on xml based description programming language, YML provides a high level programming interface which is very easy to use.
- YML is based on some middleware. YML compiler will generate a schedule table through parse pseudo code based application program developed using YvetteML. Then, YML scheduler will allocate appropriate tasks to available YML workers. YML worker will put available tasks to related computing resources according to its local scheduler. Now it can support two middlewares: XtremWeb and OmniRPC.

From the summary, we can know that it is reasonable for YML to have some overhead. The overhead comes from two aspects:

- YML need to invoke related “implementation components” from YML server. While, OmniRPC invoke their related “implementation functions” from local server.
- YML server has to deal with “scheduler table” when each event happens. Even when the scheduler table is very big, the overhead will become larger.

Paper [126] had some basic experiments based on Par-par BbGJ algorithm to evaluate the overhead of YML. Its conclusion shows that the overhead of YML was not important when the scheduler is not overloaded to solve the data dependences. Also YML's overhead can be reduced through some technology such as out of core.

This section will present another case to show that YML can be a good choice for end users to reduce time to solution. Here we will make a series of experiments to testify that.

Experiment environments: 100 nodes used in cluster of grelon, Nancy site, France. Experiment data: we change the block-count of sub-matrix from 2*2, 3*3, 4*4, 5*5, 6*6, 7*7 and 8*8. We also change the block-size of sub-matrix from 500*500, 1000*1000 and 1500*1500.

From Figure5.3, we can find the overhead of YML on OmniRPC through comparing Par-par BbGJ algorithm on YML and on OmniRPC. Its conclusion is the same with that of paper [126]. But here, what I want to emphasize is that, the performance of Max-par BbGJ algorithm on YML is very close to that of Par-par BbGJ algorithm on OmniRPC. At the same time, we know programming Max-par OmniRPC with YML is very easy (see the detail program in appendix A) and it is more difficult to program Max-par algorithm on OmniRPC. The reason is that there are more complex control events which are used to deal with concurrency of application program, during the process of execution. The complex control events make programming using OmniRPC become more difficult. In other words, YML can reduce the time to solution of running a new algorithm through its easy-to-use interface and it also can reduce cost to solution through components reuse. So YML can be a good choice for end users making large scale scientific computing.

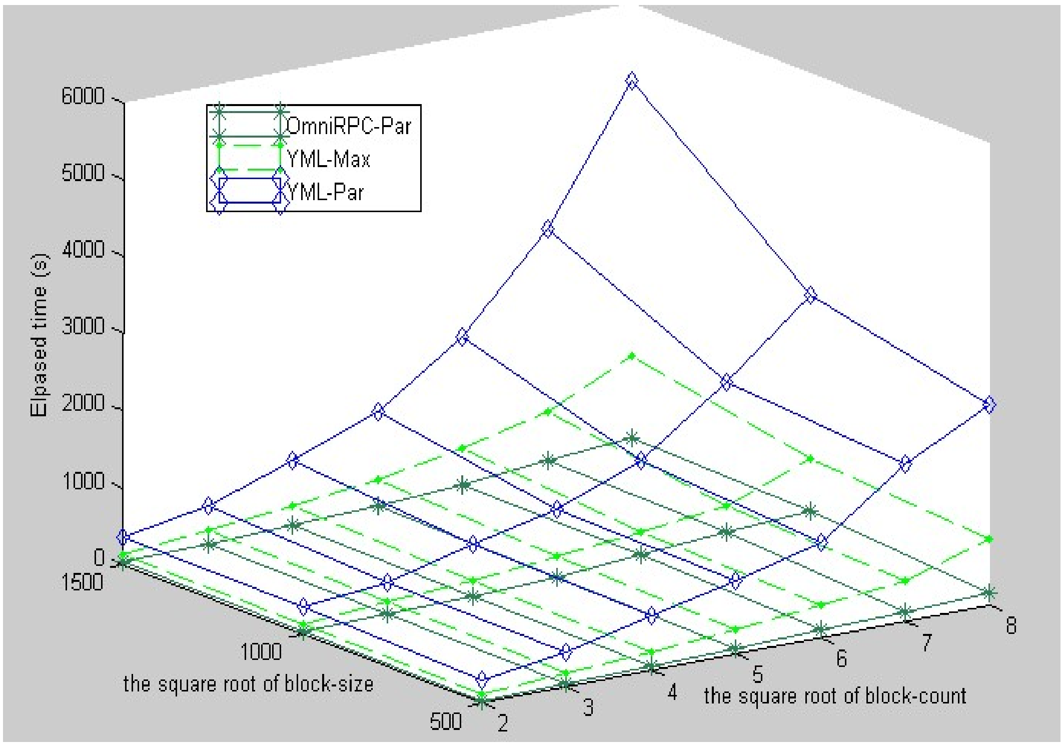


Figure 5.3. Overhead of YML on OmniRPC

Table 5.1 Computing resources used in Hohai platform

Site	Nodes	Bandwidth	CPU/Memory
Lab 303	16	100MB/s	Inter , 2.66GHz/512M
Lab 101	64	100MB/s	AMD, 1.8GHz/512M

Table 5.2. Overhead of YML on XtremWeb

	XtremWeb	YML+ XtremWeb	overhead
1500*2	608	727.88	119.88
1500*4	3675	3943.7	268.7
1500*6	8943.67	9704.47	760.8
1500*8	17633.4	19736.2	2102.8

We discussed that YML can be a good choice of achieving less time to solution. Now we want to emphasize another feature of YML which is its portability between different kinds of platforms. This feature is unique and little middleware can posse this kinds of capability and it can make sure the program develop with YML can be run on Grid platform and Desktop Grid platform without any change. It is appealing for scientific researchers who want to use volunteer computing resources to reduce their costs. So we made some experiments based on XtremWeb using Par-par BbGJ algorithm on Hohai platform.

Experiment data: we change the block-count of sub-matrix from 2*2, 4*4, 6*6 and 8*8. We also change the block-size of sub-matrix from is fixed on 1500*1500. The bandwidth between lab1 and lab3 are 1GB/s. The detail environment can be described using Table 5.1.

Through running the program for XttremWeb based Desktop Grid platform, we can get the results presented in Table 5.2. Here we don't want to compare the middleware XtremWeb with the middleware of OmniRPC, because there is no comparability between the two middlewares which suit for different environments. What we want to emphasize is that YML can adapt its program to different environments (Grid environment or Desktop Grid environment) and **this feature is specia l for YML** . With the success of Seti@home, more and more scientific computing will try to use volunteer computing resources for its lower costs and huge processing power. So YML is a very successful attempt to make program migrate between different environments without any change. Through the experiments, we also conclude that YML has an acceptable overhead and it can help to reduce costs on time and money for users through high level programming interface and reusable components.

To summary, YML can be a good choice for scientific researchers to make large scale computing and the excuses can be described as follows:

- YML has an acceptable overhead and it can help users to reduce time to solution through its high level program interface.
- The separation of "implementation component" from "control flow" make developed code reuse very easily, which help users to reduce cost-to-solution through components reuse.
- YML support program migration between different executable environments (for example Grid, Desktop Grid, Cluster) without changing the developed code. This is very special and important character for YML and little other middleware can do this.

5.4 Characters of different environments

The previous sections have testified that YML is a good choice for end users to make large scale computing and one of reasons is that it can supports different environments. As well known to us all, the different environments have different characters. In this section, we will talk about some issues which can help users to utilize the environments better. Those issues range from “task granularity”, “data transfer model” and “schedule mechanism”.

5.4.1 Task granularity

First, we have proved that fine-grain based parallel algorithm can achieve better performance in high speed network based environments (see detail in Chapter 4). But to the low speed network based environment, fine-grain based parallel algorithm can’t always achieve better performance. Let’s see the following experiments on Ploytech Lille platform.

Table 5.3 computing resources in PolyTech Lille platform

Number of PCs	CPU	Memory
32	AMD athlon 2.2, GHz	2G
16	AMD athlon 2.7 GHz	2G
16	Pentium 2.6 GHz	2G
16	Inter celon 2GHz	1G
16	AMD athlon 2.3 GHz	2G
4	Intel Pentium 2.4GHz	512M
8	Intel Celeron 1.4GHz	512M

Table 5.4 Block-size is 100*100 on PolyTech Lille platform

Algorithm	Block-count			
	2*2	3*3	4*4	5*5
Max-par	32.52	64.37	129.88	230.11
Par-par	30.53	67.91	133.29	238.24

Table 5.5 Block-size is 2000*2000 on PolyTech Lille platform

Algorithm	Block-count			
	2*2	3*3	4*4	5*5
Max-par	5234.8	8456.9	11897.4	15270.8
Par-par	5429.6	9897.5	13260.4	18938.38

The experiments are made on Polytech Lille platform. The software is YML and OmniRPC. The main difference between Polytech Lille platform and Grid5000 is bandwidth of network. Grid5000 is connected by special high speed network and Ploytech Lille platform is connected by 100M based Ethernet network. We also ensure there are enough computing resources in the experimental environment.

As we know, Max-par BbGJ algorithm can make more tasks executed concurrently. At the same time, there are more communications between each operation. When making scientific computing in such an environment, a key issue is to balance the communication time and computation time. In this experiment, when block-size is 100*100, computation time has the small proportion of the total time and communication time becomes a more important factor. To low speed network based environment, fine-grain based parallel algorithm can't achieve better performance. So the performance of Max-par BbGJ algorithm is little worse than that of Par-par BbGJ algorithm. When the block-size is 2000*2000, the computation time becomes more important in overall time. The fine-grain task based parallel algorithm shows its advantage. And the results in Table 5.5 testify our viewpoint.

To summary: A key character of Desktop Grid environment is its low speed network. So fine-grain task based parallel algorithm can't always achieve better performance. Only the proportion of computation time is larger than that of communication time, the better performance of fine-grain tasks based parallel algorithm can achieve better performance.

5.4.2 Data transfer model

From the last section, we know in the low speed network based environment, communication time is a key issue in making large scale computing. Then, is there any way to optimize communication when making large scale scientific computing? The answer is absolutely, yes. Because the data transfer model of many middlewares is based on Data server-Worker model. So improving the data transfer model becomes an important way to improve the performance of low speed network based environment. And much effort is made to improve it. In this section, we just try to simulate the data persistent and uncover the potential of improving the performance.

The simulation method is that: we will generate as little as communication during the process of program execution. If a data migration is needed in the program, we will not transfer the related data from the data server to target machine, but generate the same volume data on target machine. Thus, less communication is needed.

The experiment environment is based on Polytech Lille platform described in Table 5.1. And we will run Max-par BbGJ algorithm on low speed network based PCs. See Table 5.6

Another experiment environment is based on Grid5000 which is special high speed network based platform. We also make the experiment using Max-par BbGJ algoritm. See Table 5.7.

Table 5.6 Block-count is 5*5 on PolyTech Lille platform

Algorithm	**Block-size**				
	100*100	200*200	300*300	400*400	500*500
With DP	123.25	198.45	251.46	368.98	563.87
No DP	214.87	328.87	498.13	760.33	1073.60
Gain	42%	40%	49.5%	51.5%	47.5%

Table 5.7 Block-count is 5*5 on Grid5000 platform

Algorithm	**Block-size**				
	100*100	200*200	300*300	400*400	500*500
With DP	6.27	9.87	12.31	37.74	52.33
No DP	4.19	5.98	7.48	21.04	32.56
Gain	33%	39.4%	39.2%	43.2%	37.8%

From the simulation based experiments, we can conclude that with data persistence the performance can improve almost 50% in Desktop Grid environment and about 40% in Grid environment. Especially in Desktop Grid environment, data persistence technology can help to save a lot of time. So how to realize data anticipate migration is a key issue of improve the performance of platform in Desktop Grid environment. Further research will be made on this point in the next chapter.

5.4.3 Schedule mechanism

We have talked about the influences from "task granularity" and "data persistence". Now we will explore another factor, schedule mechanism which plays very important role in the process of making large scale scientific computing.

Schedule mechanism in Gird/Desktop Grid environment is complex for their characters of those environments. As described in chapter 2, heterogeneous is the key character of those environments. Heterogeneous includes a wide range from network, CPU, memory to core. In Desktop Grid environment, volatility is another important key issue which influences the performance of platform greatly. In this section, we just want to show the influence on schedule mechanism from character of heterogeneous.

The experiment environment is based on Polytech Lille platform described in Table 5.1. And we will run Max-par BbGJ algorithm on low speed network based PCs. In this experiment, we choose 5 nodes from different clusters in the first case and those machines are heterogeneous. The second case, we will select 5 nodes from the same cluster and those machines are homogenous. The detail result can see Table 5.8.

Table 5.8 Block-count is 5*5 on PolyTech Lille platform

Nodes	**Block-count**			
	100*100	200*200	300*300	400*400
Same cluster	229.07	335.72	568.35	920.75
Five clusters	248.93	376.70	587.6	1021.3

From the experiment, we can find the time using 5 nodes from 5 clusters is a little longer than that using 5 nodes from the same cluster. The case using 5 nodes from 5 different clusters represents the situation in which the heterogeneous isn't taken into consideration during scheduling tasks to computing nodes. In this experiment, we use the average time of ten independent experiments. Many times based experiments can make sure those tasks which need more time to be executed will not always be allocated to more powerful computers. This helps to makes the situation ("unreasonable schedule") happen. So facing the heterogeneous environment, the best way is to allocate tasks to appropriate computing resources according to task's requirement. Condor can deal with this kind of situation, but now it mainly suit for Grid environment and it doesn't support the Desktop Grid environment. In the next chapter, further research will be made on this point.

5.5 Conclusion

In this chapter, we have discussed some related issues about large scale computing on Grid and Desktop Grid environments. The aim of this discussion is to find an easy way to make large scale scientific computing on different computing environments. At the same time, costs on time and money for developing application program as another important factor, also should be taken into consideration. According to this goal, we made further research on different kinds of middleware and environments. Grids as the main stream architecture for high performance computing system should be given more attention. At the same time, volunteer computing based Desktop Grid environment arouse more attention from scientific researchers for its huge process power and very lower costs. In recent years, many tools to make large scale computing based on Grid and Desktop Grid environment have been developed. Facing so many choices, it is hard for end users to find appropriate tool to make large scale scientific computing on different environments. To deal with that, related researches ranging from programming model, data transfer model, task granularity to schedule mechanism, are made and some conclusions can be described as follows:

- From the section 5.2, we can know about YML is workflow based framework which provides end users with a very easy-to-use and high level programming interface. OmniRPC and XtremWeb both provide high level programming interface, but YML's interface is much higher level. Its interface is pseudo code based and platform/middleware/system independent.

- From the section 5.3, we can know YML is based on other middlewares such as XtremWeb and OmniRPC. Overhead is unavoidable. The section presents YML has an acceptable overhead. Besides, the code for YML can be migrated between Grid environment and Desktop Grid environment without any change. As far we know, this advantage is special for YML. Also, YML supports the separation of "control flow" and "implementation component" and this helps end users just to focus on application itself without considering the detail underlying execution environment. The separation also makes the developed code reused very easily.

- In the section 5.4, some factors which play important roles in making large scale scientific computing are discussed. Programming in low speed network based Desktop Grid environment should take the balance between communication time and computation time into consideration. We can balance that through changing task granularity. Data persistence is another key aspect for large scale scientific computing. The simulations in this part testify the performance of platform can be improved greatly through adopting appropriate data transfer model. Last but not least, schedule mechanism is very important aspects in high performance systems. In this part, we made researches on the influence from heterogeneous computing resources in Grid and Desktop Grid environment. Experiments show that it helps to improve the performance of platform for end users to take the properties of heterogeneous computing resources into consideration when scheduling tasks in Desktop Grid environment.

To summary, YML can be a good choice for end users to make large scale scientific computing for its many appealing features. It also can help users to reduce the time/money to solution for their applications.

Chapter 6

A Reference Architecture Based on Workflow for Building Scientific Private Clouds

6.1 Motivation

Cloud computing arouses great interests from scientific researchers and IT enterprises. A lot of people present their understanding on Cloud computing and many products have been launched by some famous IT enterprises such as Google, IBM, Amazon and Microsoft. But up to now, no agreement on what Cloud computing is. This chapter will try to find the essence of Cloud computing from different existing Cloud paradigms. Then try to explore the difference between Grid computing and Cloud computing in scientific computing area. Based on that, some common features of Cloud computing are summarized. According to these feathers, we present our viewpoint on scientific Cloud computing systems and some hottest research points. Then, according to our understanding on Clouds, lessons learned from gridification and our experiences of scientific computing on Grids, we will try to propose a solution to build Private Clouds for scientific computing.

6.2 Introduction

Cloud computing has arose more attentions from its born. Those attentions come not only from news (news reference volume) but also from scientific researchers (search volume index), see detail in Figure 6.1. Many famous IT enterprises such as Google, Microsoft, IBM, Sun, Amazon have launched their products/models of Cloud computing. At the same time, huge interests have also been arose among scientific computing domain. More and more famous scientists from Grid area [131] [132] [133] and web 2.0 [129] [130] also put forward their understandings on Cloud computing.

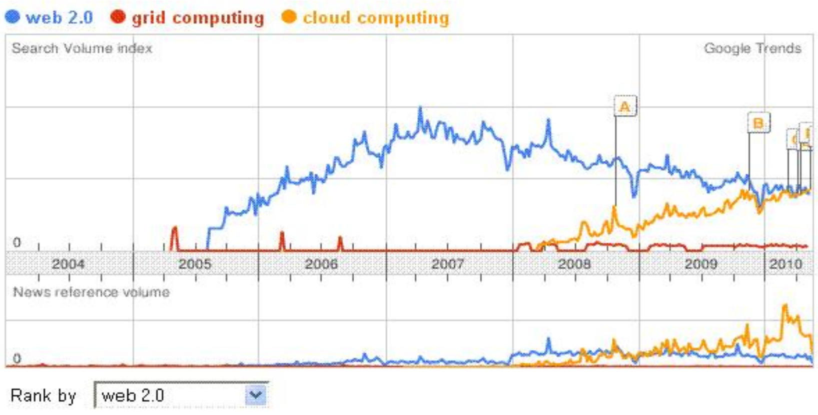


Figure 6.1. Cloud computing received more attention

About what Cloud computing is, there is still no clear definition which can be generally accepted. According to Gartner [128], Cloud computing is a style of computing where massively scalable IT-related capabilities are provided as a service across the Internet to multiple external customers. According to IBM[18], a Cloud is a pool of virtualized computer resources that hosts a variety of different workloads and allows them to be deployed and scaled-out through the rapid provisioning of virtual machines or physical machines. It also supports redundant, self-recovering, highly scalable programming models and resource usage monitoring in real time to enable rebalancing of allocations when needed. The viewpoint from some industries is to take Cloud systems as narrow Grids. To Cloud systems, its powerful services and applications are being integrated and packaged on the web [134]. Paper [135] thinks the emergence of Cloud computing as a new potential super structure or the third generation internet based structure for enterprise and academic computing. From the viewpoint of [136], Cloud computing is an emerging model of computing where machines in large data centers can be dynamically provisioned, configured, and reconfigured to deliver

[18] http://www.ibm.com/ibm/cloud/

services in a scalable manner, for needs ranging from scientific research to video sharing to email. [131] defines Cloud computing as a large scale distributed computing paradigm that is driven by economies of scale, in which a pool of abstracted, virtualized, dynamically scalable, managed computing power, storage, platforms, and services are delivered on demand to external customers over the internet. [132] proposes Cloud computing is a new and promising paradigm delivering IT services as computing utilities. Clouds are designed to provide services to external users and providers need to be compensated for sharing their resources and capabilities. [137] supposes Clouds characteristically expose a minimal set of system semantics required to support the Cloud's usage modes. Brown thinks Cloud computing is a data-processing infrastructure in which the application software and often the data itself are stored permanently not on your PC but rather a remote server that's connected to the Internet [138]. Cloud computing is about moving services, computation and/or data to an internal or external, location-transparent, centralized facility for cost and business advantage. By making data available in the Clouds, it can be more easily and ubiquitously accessed, often at much lower cost, increasing its value by enabling opportunities for enhanced collaboration, integration, and analysis on a shared common platform [139]. Bragg thinks the key concept behind the Clouds is web application. Many find it's now cheaper to migrate to the web Clouds than invest in their own server. So in near future, perhaps it is a desktop/Web browser for people without a computer under the help of matured Cloud technology [140]. Kaplan's viewpoint is that a broad array of web-based services aimed at allowing users to obtain a wide range of functional capabilities on a 'pay-as-you-go' basis that previously required tremendous hard- ware/software investments and professional skills to acquire. Cloud computing is the realization of the earlier ideals of utility computing without the technical complexities or complicated deployment worries... [141].

There are still other definitions from different domain scientists. No definition is totally agreed by all. Differences are more or less. Then what is exact definition of Cloud computing? In the following section, we will look through the existing Cloud computing platform presented by Google, Amazon, IBM and Microsoft and further research on famous Cloud middleware is also made. Through those explorations, we hope we can find some common features behind the different shapes of Clouds.

6.3 Different shapes of Clouds

6.3.1 Cloud computing from Google

Google is the most important and most watched Internet company today. Search engine rankings indicate Google's market share is nearly two-thirds of the total search market, versus approximately 20% for Yahoo and 10% or less for Microsoft. With its expertise running the world's most popular search engine and its vast, industry-leading infrastructure to support the world's most visited Internet site, expanding into Cloud computing services is a natural fit.

Google's Cloud computing is mainly to support its search engineer and provide search service for end users. And now it expands to other service such as Google earth, Google trend, Google desktop… According to its problem to be solved (data process in large scale unstable computing resources based platform), it takes three key technologies to support their services.

- Google File System [142] is a scalable distributed file system for large distributed data-intensive applications. It provides fault tolerance while running on inexpensive commodity hardware, and it delivers high aggregate performance to a large number of clients.

- Map Reduce [143] is a programming model and an associated implementation for processing and generating large data sets. Users specify a map function that processes a key/value pair to generate a set of intermediate key/value pairs, and a reduce function that merges all intermediate values associated with the same intermediate key.

- Big table [144] is a distributed storage system for managing structured data that is designed to scale to a very large size: petabytes of data across thousands of commodity servers. Many projects at Google store data in Bigtable, including web indexing, Google Earth, and Google Finance.

Its general architecture can be described through Figure 6.2.

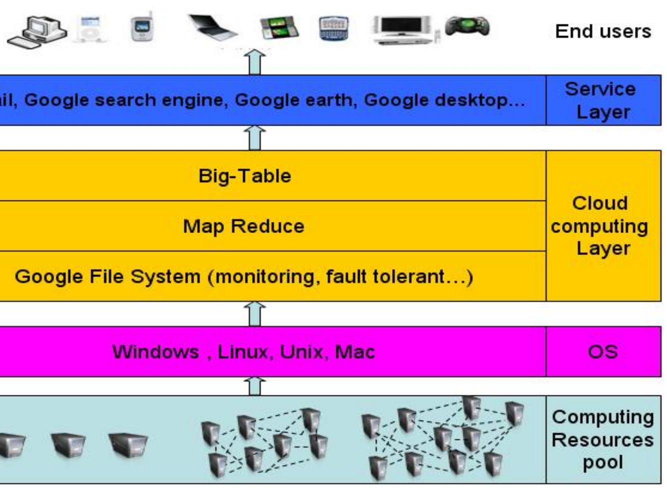

End users
Gmail, Google search engine, Google earth, Google desktop…
Service Layer
Big-Table
Map Reduce
Google File System (monitoring, fault tolerant…)
Cloud computing Layer
Windows , Linux, Unix, Mac
OS
Computing Resources pool

Figure 6.2. Cloud computing from Google

Google's App Engine is one of the most prominent examples a platform-as-a-service Cloud offering. App Engine provides a single, pre-built solution for constructing very large scale web-based applications hosted on Google's infrastructure. And the core technology of Google's app engine is based on the three key technologies described above.

6.3.2 Cloud computing from Amazon

Amazon.com[19] is a quintessential brand name in electronic commerce. In the decade since its founding, it has transformed once from an online book-seller to a general retail platform and again to the industry leading Cloud computing provider.

Amazon's Cloud offerings fall under a group of complementary products called "Amazon Web Services" and its infrastructure-level services:

- Elastic Compute Cloud (EC2): EC2 is Amazon's flagship Cloud offering. EC2 allows the metered, on-demand rental of virtual machine computing resources. EC2 is rented in units called instances, each of which represents a virtual server with particular hardware specifications. From a user's perspective, it is like renting physical servers by the hour in any quantity. There are five differentiated types of instances to rent with varying CPU power, memory, hard disk space and IO performance. An application needing a significant amount of RAM or CPU performance can rent more expensive but more powerful instances, while a network-bound application, like a web-server, can use cheaper and less powerful instances.

- Elastic Block Store (EBS): Elastic Block Store works in conjunction with EC2 to provide extra high performance, persistent storage to EC2 virtual machine instances. EC2 instances have local storage capacity, but such space is temporary and only available while an instance continues to run. EBS provides storage like a virtual disk (block storage) which can be attached to a given EC2 instance; the data will stay available independent of the EC2 instances currently running and can be moved from instance to instance without the need to explicitly build some sort of higher-level data transfer mechanism.

- Simple Storage Service (S3): S3 provides robust object storage metered per gigabyte per month. While EBS provides a virtual disk-like block storage abstraction to attach to EC2 virtual machine instances, S3 provides a storage facility which can be accessed independent of EC2 instances. One can use S3 by itself as a storage repository without using EC2; one can also have many EC2 instances accessing the same data from S3.

19 http://aws.amazon.com/ec2/

- SimpleDB: SimpleDB is a pseudo-relational data storage service. It stores data much like a relational database management system, providing a richer data query and manipulation interface than block or object storage. SimpleDB is also accessible independent of EC2 instances and presents higher-level database-like storage accessed using a SQL-like query language.

- CloudFront: CloudFront is a Content Delivery Network (CDN) which works with data stored in S3. A CDN is designed to enhance the delivery of data (content) to data consumers (customers / end users) by providing closer "edge locations" for distribution. By providing many different edge locations, a content provider can provide end users with lower delivery latency and better performance.

- Simple Queue Service (SQS): Amazon's Simple Queue Service provides reliable messaging between distributed software components. It is often used in conjunction with EC2 to coordinate the actions of different instances or distinct components of a bigger application running on EC2.

- AWS Premium Support: AWS Premium Support is not a technical product offering itself; it is paid support and consulting related to Amazon's Cloud services. Amazon will provide help with both operational support and technical issues related to software development using their Cloud services.

Amazon is the biggest retail seller on line and a huge number of deals have been made based on its Cloud computing platform (this is the motivation for Amazon to launch their Cloud platform). Further more, they present theirs talented products – virtual resources as services to meet all kinds of users with lower costs. The general idea can be described using Figure 6.3.

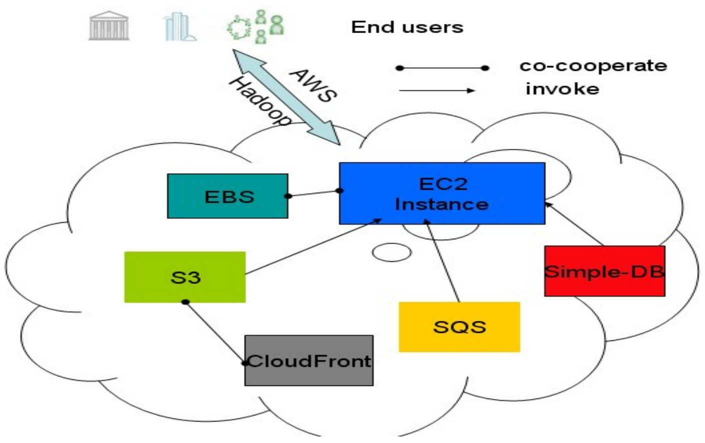


Figure 6.3. Cloud computing from Amazon

6.3.3 Cloud computing from IBM

IBM[20] announced its Blue Cloud initiative on November 11, 2007. In 2008, IBM unveiled that their "Blue Cloud", [145] is a series of Cloud computing offerings that will allow corporate data centers to operate more like the Internet by enabling computing across a distributed, globally accessible fabric of resources, rather than on local machines or remote server farms. Blue Cloud, built on IBM's expertise in leading massive-scale computing initiatives, will be based on open standards and open source software supported by IBM software, systems technology and services. IBM announced today that its Blue Cloud development is supported by more than 200 IBM Internet-scale researchers worldwide and targets clients who want to explore the extreme scale of Cloud computing infrastructures quickly and easily.

The IBM Clouds consists of data center holding multiple processors and local storage, some components developed by IBM and some open sources software. Provisioning, management, and monitoring will be handled by IBM Tivoli Monitoring and Tivoli Provisioning Manger, with access through Websphere application server, and DB2. On the platform, a virtualized infrastructure based on Linux and Xen virtualization supports individual applications. Linux with Xen will be managed by the Tivoli Monitoring agent to provide multiple virtual Linux machines. Parallel processing is enabled by Hadoop provisioning manager supporting the Eclipse programming infrastructure, with Google's Map/Reduce programming model used to create distributed processing loads. See Figure 6.4.

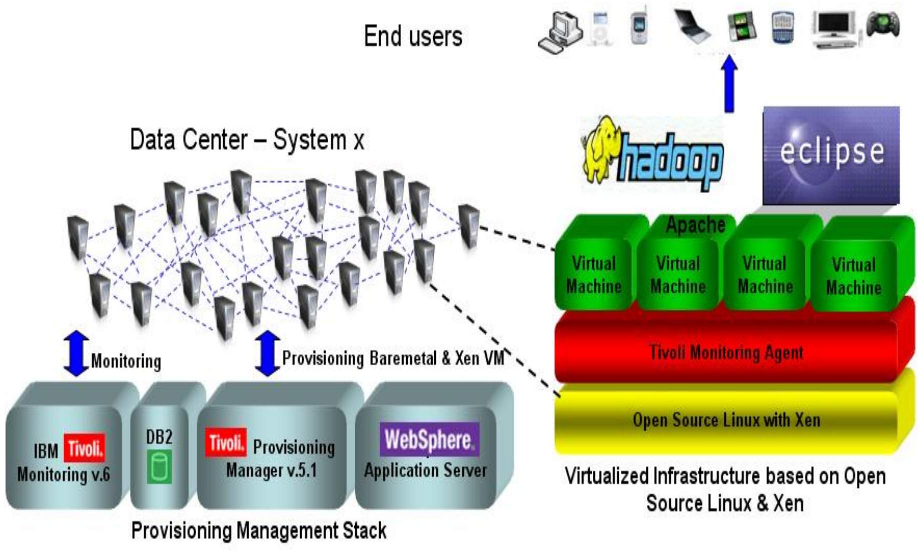

End users
Data Center – System x
hadoop
eclipse
Apache
Virtual Machine
Virtual Machine
Virtual Machine
Virtual Machine
Tivoli Monitoring Agent
Open Source Linux with Xen
Virtualized Infrastructure based on Open Source Linux & Xen
Monitoring
Provisioning Baremetal & Xen VM
IBM Tivoli Monitoring v.6
DB2
Tivoli Provisioning Manager v.5.1
WebSphere Application Server
Provisioning Management Stack

Figure 6.4. Cloud computing from IBM

[20] http://www.ibm.com/ibm/cloud/

IBM Tivoli software that manages servers to ensure optimal performance based on demand. This includes software that is capable of instantly provisioning resources across data center to provide users with a seamless experience that speeds performance and ensures reliability even under the most demanding situations. Tivoli monitoring checks the health of the provisioned servers and makes sure they meet service level agreements.

Hadoop is an open source software project under the Apache server project, designed to run jobs, distribute tasks and store data in a parallel and distributed fashion on a multiprocessor system. Most programs run on Hadoop are written in the Map/Reduce style, in which input is broken into small pieces that are processed independently according to a map. The results of these independent processes are then collated into groups and reduced to produce a result. IBM is providing Eclipse, a business development platform, on top of Hadoop, and has written its own Map/Reduce toolset for Eclipse in support of this.

Xen is high-performance open source virtualization software. It operates through the Xen hypervisor, which sits between server hardware and operating system permitting each physical server to run one or more virtual servers. Virtual server images can be run on any server at any time, and multiple virtual servers can simultaneously share a single server.

To summary, IBM Blue Cloud is a computing platform with software and hardware. It is the paradigm which expands enterprise architecture to the Internet based wide area and move data center to internet environment (this is the driver for IBM to launch their Cloud platform). In its paradigm, a lot of state-of-art technologies on large scale computing, service technology, software developed by IBM are adopted. The key characters of IBM blue Cloud is that Logic partition can be used through hardware virtual technology, i.e., users can use IBM enterprise workload manager to manage CPU resources of logic partition. Then it will distribute appropriate computing resources to corresponding logic partition. Also software based virtual technology can run another kind of operation system (Windows for example) on Linux based machine through Xen.

6.3.4 Cloud computing from Microsoft

The Cloud computing platform of Microsoft is based on Azure which is a Cloud operating system. Many services such as computing, storage, management are integrated into this operating system. Above the Azure, several kinds of services (live services, .NET services, SQL services, SharePoint services, Dynamic CRM services) facing to end users run on the Azure. See Figure 6.5. These services are the base of next generation Internet service (this is the driver for Microsoft to launch their Cloud platform). These services have a close connection with peoples' daily life and are popular for their easy of use. Live services can provide end users services like blog, picture, MSN. Dot NET's role is to provide a common use service such as workflow, service bus, access control and it can be reused very easily. SQL service is used for data management. Coordination between services is in charge by SharePoint services. Dynamics CRM services are used for application level just like Salesforce.com. Microsoft can provide different level services through different service interfaces. See Figure 6.5

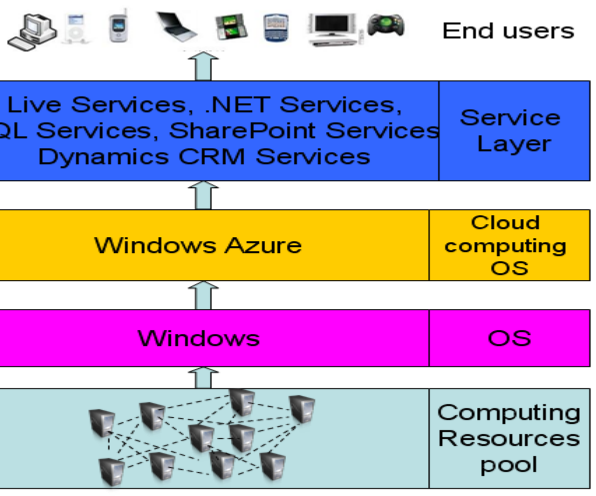


Figure 6.5. Cloud computing from Micosoft

6.3.5 Eucalyptus

Eucalyptus [146], a Cloud enabling infrastructure is the result of a research project from the University of California, Santa Barbara. Eucalyptus stands for "**E**lastic **U**tility **C**omputing **A**rchitecture for **L**inking **Y**our **P**rograms **T**o **U**seful **S**ystems". It aims to provide a simple way to set up Cloud solution for the research and development of Cloud driven applications. By combining common web-service, Linux tools and the Xen Virtual Machine Hypervisor, Eucalyptus successfully implemented partial functionality of the popular Amazon EC2.

The Eucalyptus infrastructure (Figure 6.6) consists of four main components, the Cloud Controller (CLC), the Cluster Controller (CC) the Node Controller (NC) and a storage service called Walrus. These components are implemented as stand-alone web services. They leverage WS-Security policy for secure communication via a well-defined WSDL API. They interact via SOAP and HTTP to dynamically provision Virtual Machines (VMs) or manipulate VM images. Eucalyptus currently supports only Xen VMs.

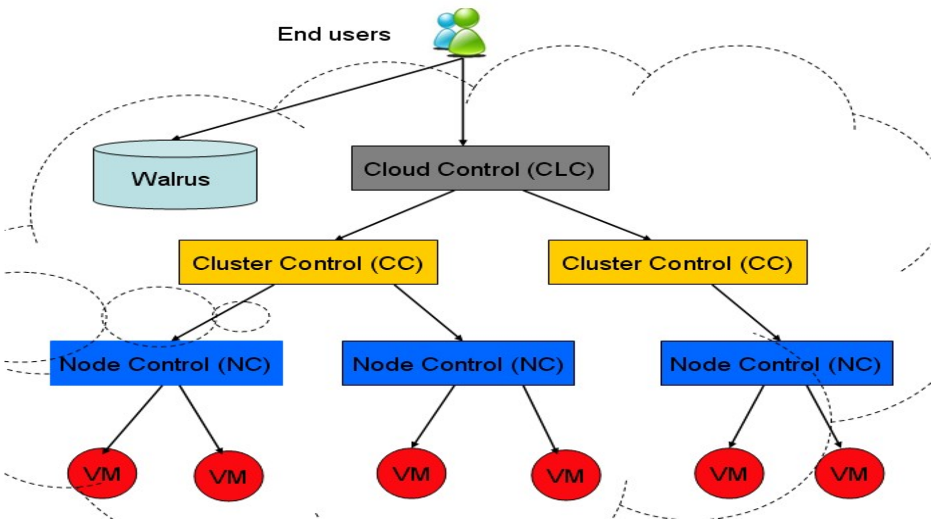


Figure 6.6. Architecture of Eucalyptus

Each physical machine in the cluster capable of hosting VMs is installed with a NC. The NC starts and stops VM instances and monitors the health of those instances on that particular machine. The CLC is the interface for external entities to communicate with the Eucalyptus environment. It receives requests to start or stop VM instances and forwards these requests to the CC. The CC maintains resource information about the entire Eucalyptus cluster by periodically polling the NCs. Based on the global resource availability, the CC then decides which NC in the cluster should be contacted to start or stop a VM.

Walrus is a storage service similar to that of Amazon S3. The primary use of Walrus is to store VM images called Eucalyptus Machine Images (EMIs). The EMI format is used by Eucalyptus to encode images which will in turn be used to start Xen VM instances. EMI is similar in concept to the Amazon Machine Image (AMI) format that is used to start EC2 instances. Walrus can be accessed using Curl, a popular command-line tool for interacting with HTTP services. Other than storing EMIs, Eucalyptus users can also use Walrus to store raw data the same way an EC2 user employs S3. Currently, Walrus must be installed on the same physical machine as the CLC.

6.3.6 Summary

We have introduced some famous and very important Cloud systems in the previous sections. Some points can be summarized as follows:

- Each enterprise has their special goal to present their Cloud platform. This is to say, they present the platform to solve some special problems and present some related key technology to deal with them. The Google's Clouds is to deal with large scale data retrieval (search engineer) based on web. They propose their three key technologies which are Google File System, Map Reduce and Big Table to realize their goals. They gradually propose some other services based on their Cloud platform. The same way, Amazon is a book seller on line. They build their large data center to make the on-line bookstore serve more customers. But they find those computing resources in data center can't be made full use of. So a talented idea which is to sell computing resources is proposed and according to this idea, they present their Cloud platform based on Virtual technology to maximize utilizing the computer resources. Microsoft presents Azure OS to support their large scale on-line services such as Live message, .NET services. Eucalyptus is research platform for making experiments on EC2 and it present related strategy to adapt to EC2 environment.

- The key technology for different Cloud systems is different. Cloud layer in those platforms plays an important role in separating user's interface and physical executable environment. In other words, end users need not to know how/when/where their applications are implemented.

- They all try to make end users get services through Internet at lower cost and in an easy-to-use way. The general way is to decrease costs through making full use of computing resources based on virtual technology and to provide a high level interface to make these services be used by end users in a very simple way.

- They all provide services on demand to users and those services are very stable. To make those services be based on SLA, some special technologies on monitor, fault tolerance, data migration and schedule strategy have to be adopted.

6.4 From Grids to observe Clouds

We have analyzed some important Cloud platforms in the previous section and have a general impression on Cloud computing. We also made a survey on Grid computing in chapter 2. Then a question is aroused which is what's the relation between Grids and Clouds. About the difference between Grid computing and Cloud computing, many scientists proposed their understanding on this question. Among that, Ian Foster [131], Giacomo [147], buyya [132] and Geoffrey [137] have made exhaustive analysis on difference between them. In this section, we want to compare them from the viewpoint of practice. Grids are presented in the mid 1990s and its aim is to allow consumers to obtain computing power on demand. The idea of creating computing Grid comes from utility of the electric power grid. After more than 10 years' effort, a lot of large scale Grid systems have been produced. Such as TeraGrid, EGEE, Open Science Grid, LHC

Computing Grid Seti@Home, Boinc, Grid5000. A lot of applications begin running on Grid platform. But the influence of Grid on most people engaged in scientific research is small. Many scientific applications are still running on clusters and servers (based on cluster computing and distributed computing). According to paper [148] and our experiences on Grid5000, the reasons of hard gridification can be summarized from the following two aspects.

6.4.1 Viewpoint from end users

6.4.1.1 Grid platform is hard to utilize for end users

General speaking, Grids expose all the interfaces to users, though a lot of interfaces is useless to a specific user. And users usually have to spend a lot of time on finding the interfaces they need to use. Then he needs to learn how to program using those interfaces. Generally speaking, it is not easy to learn for non-expert end-users.

Grid platform is a kind of very complex system. End users have to book computing resources they need and then deploy the environments they want to use. This is difficult process for non expert users, because users need to know the infrastructure of the Grid platform and the number of computing nodes they need and then a series of complexity issues span over programmatic, technology and management perspectives make users discourage using Grid platform. We will take using Grid5000 platform as example to show that. Grid5000 [113] is a national grid platform in France. It is a state of art Grid platform and has thousands of users. But it is not easy to use it. Users must know how many nodes their program need in advance. Once these computing nodes are booked, they can't be utilized by other users, even they are idle, i.e., the Grid platform is not based on demand provision. When a large scale computing resources in their program are needed, the common way is that users need write shell script to book computing resources and deploy environment they need. Normally, it is not easy for programmer to do that, not to mention to general end users. So many users would like to use theirs former platform such as cluster or supercomputer.

6.4.1.2 Grid middleware is hard for end-users to develop application

Grid Computing is evolving to mean the sharing of geographically distributed resources. A number of Grid middlewares such as Globus, Legion, Alchemi, Xtremweb, Gridsolve, Netsolve, Ninf and OmniRPC have been presented to adapt to Grid environment. But to successfully utilize such middlewares, end-users are required to know Grid platform's details ranging from resource configuration to service availability to hardware configuration. When creating Grid applications using middlewares, proper management strategy on underlying resources has to be selected. Other factors such as resource availability, fault tolerance, load balancing and scheduling mechanism also have to be taken into consideration. These requirements significantly increase the burden of Grid users, i.e., Grids middlewares aren't able to deliver on the promise of better applications and usage scenarios, which keeps many users from utilizing Grid middleware.

6.4.2 Viewpoint from Grid system

6.4.2.1 Problem from Grid Scheduler

Most schedulers in Grid middlewares are based on batch scheduler, Such as Gridsolve, OmniRPC and XtremWeb. This kind of scheduler doesn't support adaptive scheduling in Grid system, thus causing the scheduler can't dispatch tasks to appropriate computing resources. Normally, scheduler can't choose appropriate computing resources from computing resources pool according to tasks requirements. Users must know how many computing nodes they need to book for their application. Generally, it is the adequate book that may cause great waste of computing resources. Grid platform is a heterogeneous environment in both available time and capability of computing resources. So static and a single level scheduler can't meet the requirement of Grid platform. An example from TeraGrid can testify what we expressed above [147]. The heterogeneous and distributed nature of TeraGrid implies a "best effort" strategy when providing supercomputing services. This state of the Grid system is such that only very specialized researchers can benefit from the Grid computing power, due to the current design of scheduler is based on batch-queue systems. This makes interactive supercomputing become more difficult, as results have to wait for other large parallel jobs.

6.4.2.2 Problem from Grid data transfer model

Most data transfer models are almost based on Serve/Client. All the data are stored in data server. As Figure 6.7 shows, when "task 1" is to be executed on "machine 1", the machine need get the related data from data server and when the task finishes, the results will be sent back to data server. The same ways will be executed for "task 2" and "task 3". Most Grid middleware such as OmniRPC, Gridsolve, Netsolve, Xtremweb and BOINC, are based on this strategy. Data persistent and data replication technology can help to deal with that and through those technologies, a lot of time on unnecessary communication can be saved. Some efforts are being made in some Grid middleware, such as SmartGridSolve.

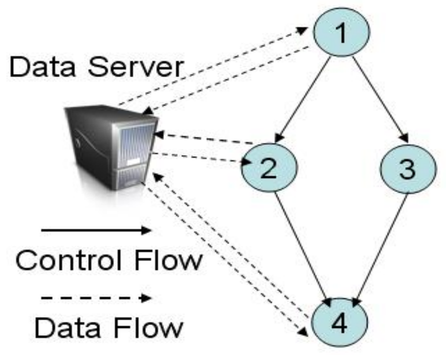


Figure 6.7. Data transfer model in most Grids middleware

6.5 Summary on Clouds

Some issues have been discussed from the limitation of existing Grid systems in the previous section. To Cloud system, it should overcome those disadvantages of Grid system. In this section, what kinds of common features should be possessed for Clouds is discussed. Then classification of Clouds is introduced and some challenges are listed. Finally, some research hottest point and our definition on scientific Clouds are presented.

6.5.1 Common features of Clouds

The evolution of Cloud computing has borrowed its basis from several other computing areas and system engineering concepts. Cluster and Grid Computing on one hand, and Web 2.0 on the other hand are perhaps the most obvious predecessor technologies that enabled the inception of Cloud computing. As well known to us all, Grid computing is proposed in the middle of 1990s. Grid technology is used to harness huge amount of computing resources from different administrator domains. As a result, Grids can provide huge computing power than previous architecture. At the same time, These rising popularity of user-driven online services, including MySpace, Wikipedia, and YouTube, has drawn attention to a group of technological developments known as Web 2.0. The rise of the Web 2.0 such as Web services, peer-to-peer networking, blogs, podcasts, and online social networks led to the emergence of new web applications or services which need to store and process large amount of data. In addition, with the exponential increase of users of Web 2.0, large scale IT platform with good features such as scalability, easy to use is needed. So when the web 2.0 (the requirements of large scale process power) meet the Grid computing (can provide huge process power), the Cloud computing is born. However, several other computing concepts have indirectly shaped today's Cloud computing technology, including peer-to-peer (P2P) computing, SOA and autonomic computing, SOC, Pervasive computing, Mobile computing. So we can conclude that Cloud computing should based on those technology which can harness a lot of computing resources and it also can provide an easy of use and high level interface for end users.

According to the section 6.3, we can take Cloud computing as an easy and fast solution to a specific problem. The key issue to propose the solution is to define clearly the problem to be solved exactly. Perhaps that's why people's opinions vary so much on what's Cloud computing is. As you know, the problem is totally different for different people. So the solution to problems is different. I think that's why defining Cloud computing can be so nebulous. It means the problems to be solved are totally different to each of us so the approach to build the Cloud platform is often unique. Though many differences exist in different Cloud computing, some features are still in common. They can be summarized as follows:

- Very large scale: the Cloud computing platform of Google has more than million PCs. Amazon, IBM, Microsoft also have more than 100 thousands PCs. Even Cloud platform in an enterprise has hundreds of thousands of PCs. In a word, the Cloud computing platform posses process power on demand.

- Virtualization: virtualization is an essential technological characteristic of Clouds and it can help to hide the technological complexity from the user and enable enhanced flexibility. End users can utilize services provided by Cloud platform without knowing implementation details, i.e., users need not to know where the services come from and how the services are executed. End users just need an interface to invoke the services they need.

- High reliability: reliability is considered one of the main features to exploit Cloud capabilities. Reliability denotes the capability to ensure constant operation of the system without disruption, i.e. no loss of data, no code reset during execution etc. Reliability is typically achieved through redundant resource utilization.

- High scalability: the Clouds can be scale dynamically to meet the dynamic requirement from increasing number of both tasks and users. Its capability can be rapidly and elastically provisioned to quickly scale up and rapidly released to quickly scale down. To end user, the capabilities available for rent often appear to be infinite and can be purchased in any quantity at any time.

- On demand service: the Cloud platform is a huge computing resource pool and users utilize computing resources they need. They need not book the resources in advance (Grids do), just use whenever, wherever they want to utilize without requiring users' interaction with each service's provider.

- Very low cost: the improvement of utilization rate of computing resources decreases the cost of Cloud platform. Automatic management pattern in Cloud computing also saves a lot of money than traditional management method.

- Location independent resource pool: the provider's computing resources are pooled to serve all consumers using a multi-tenant model, with different physical and virtual resources dynamically assigned and reassigned according to consumer's demand. The customer generally has no control or knowledge over the exact location of the provided resources. Examples of resources include storage, processing, memory, network bandwidth, and virtual machines.

- Pay by use: capabilities are charged using a metered, fee-for-service, or advertising based billing model to promote optimization of resource use. Examples are measuring the storage, bandwidth, and computing resources consumed and charging for the number of active user accounts per month.

6.5.2 Classification of Clouds

6.5.2.1 Deployment type

According to Wikipedia[21], the deployment models of Clouds can be classified as follows:

- Public Clouds: Public Clouds or external Clouds describes Cloud computing in the traditional mainstream sense, whereby resources are dynamically provisioned on a fine-grained, self-service basis over the Internet, via web applications/web services, from an off-site third-party provider who shares resources and bills on a fine-grained utility computing basis.

- Private Clouds: Private Clouds and internal Clouds are neologisms that some vendors have recently used to describe offerings that emulate Cloud computing on private networks. These products claim to "deliver some benefits of Cloud computing without the pitfalls", capitalising on data security, corporate governance, and reliability concerns. They have been criticized on the basis that users "still have to buy, build, and manage them" and as such do not benefit from lower up-front capital costs and less hands-on management, essentially "lacking the economic model that makes Cloud computing such an intriguing concept".

- Hybrid Clouds: A hybrid Cloud environment consisting of multiple internal and/or external providers "will be typical for most enterprises". By integrating multiple Cloud services users may be able to ease the transition to public

[21] http://en.wikipedia.org/wiki/Cloud_computing

Clouds services while avoiding issues such as PCI compliance. Another perspective on deploying a web application in the Clouds is using Hybrid Web Hosting, where the hosting infrastructure is a mix between Cloud Hosting for the web server, and Managed dedicated server for the database server.

- Community Clouds: A community Clouds may be established where several organizations have similar requirements and seek to share infrastructure so as to realize some of the benefits of Cloud computing. With the costs spread over fewer users than a public Clouds (but more than a single tenant) this option is more expensive but may offer a higher level of privacy, security and/or policy compliance.

6.5.2.2 Type of Cloud platform

The following list identifies the main types of Clouds (currently in use):

Infrastructure as a Service (IaaS):

The capability provided to the consumer is to rent processing, storage, networks, and other fundamental computing resources where the consumer is able to deploy and run arbitrary software, which can include operating systems and applications. The consumer does not manage or control the underlying Cloud infrastructure but has control over operating systems, storage, deployed applications, and possibly select networking components (e.g., firewalls, load balancers). IaaS can be called Resource Clouds and it also can divide into Data & Storage Clouds and Compute Clouds according to its main purpose.

Data & Storage Clouds deal with reliable access to data of potentially dynamic size, weighing resource usage with access requirements and/or quality definition. Examples: Amazon S3, SQL Azure.

Compute Clouds provide access to computational resources, i.e. CPUs. So far, such low-level resources cannot really be exploited on their own, so that they are typically exposed as part of a “virtualized environment”, i.e. hypervisors. Examples: Amazon EC2, Zimory, Elastichosts.

Platform as a Service (PaaS):

The capability provided to the consumer is to deploy onto the Cloud infrastructure consumer-created applications using programming languages and tools supported by the provider (e.g., java, python, .Net). The consumer does not manage or control the underlying Cloud infrastructure, network, servers, operating systems, or storage, but the consumer has control over the deployed applications and possibly application hosting environment configurations. Examples: Force.com, Google App Engine, Windows Azure (Platform).

The main difference between Compute Clouds and PaaS is that Compute Cloud Providers typically offer the capability to provide computing resources typically virtualized, in which to execute Cloudified services and applications. PaaS aims to offer full software stacks to develop and build applications.

Software as a Service (SaaS):

The capability provided to the consumer is to use the provider's applications running on a Cloud infrastructure and accessible from various client devices through a thin client interface such as a Web browser (e.g., email). The end user does not manage or control the underlying Cloud infrastructure, network, servers, operating systems, storage, or even individual application capabilities, with the possible exception of limited user-specific application configuration settings. Examples: Google Docs, Salesforce.com.

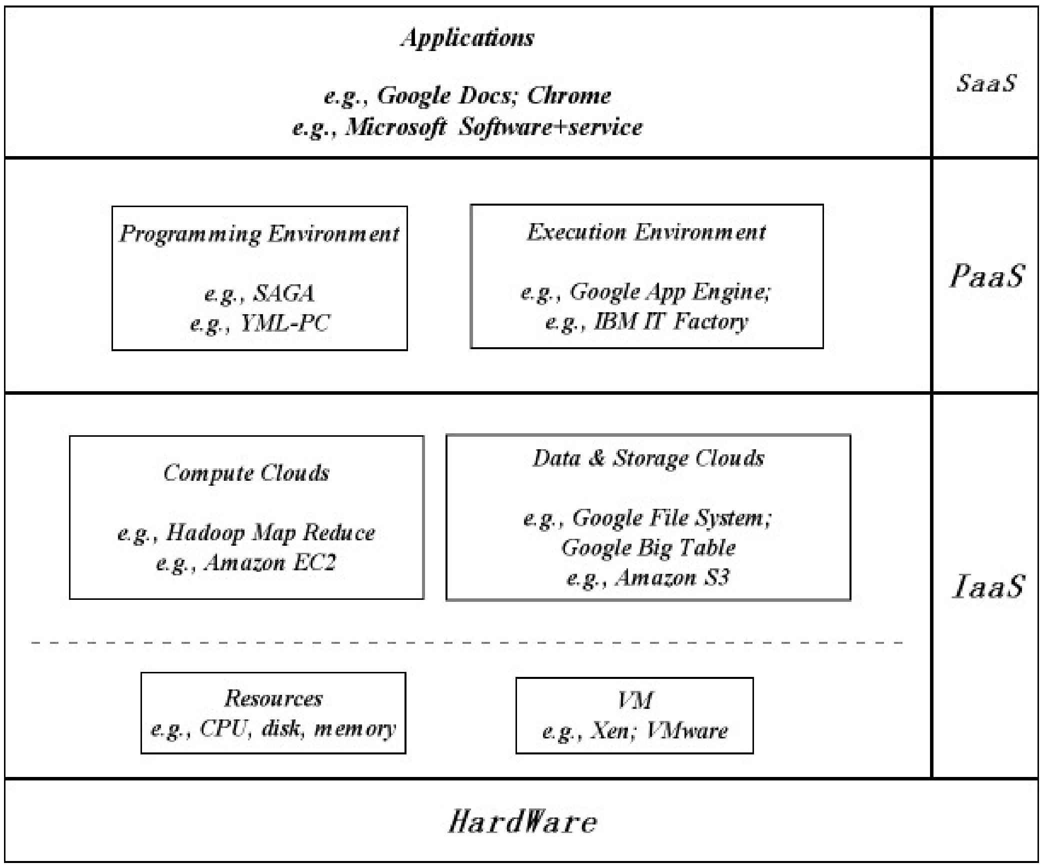


Figure 6.8. Type of Clouds and related examples

Overall, Cloud Computing is not restricted to Infrastructure/Platform/Software as a service, even though it provides enhanced capabilities which act as enablers to these systems. As such, I/P/SaaS can be considered specific “usage patterns” for Cloud systems which relate to models already approached by Grids, Web services etc. Some real example on different Clouds Type can be summarized using Figure 6.8.

6.5.3 Our understanding on Cloud computing

Summary can be made on Cloud computing from the previous sections as that Cloud computing platform is an automatic problem solving environments which collect huge computing power and users can get the services they need from Cloud platform without knowing about any technical details. It not only has the character of super process capability Grids had, but also include the good features Web 2.0 possessed such as low cost, easy of use, pay by use.

So based on our understanding, our definition for Cloud computing is presented as follows:

Cloud computing is a specific problem solving environment based on large scale resources pool (consist of cluster, Grids, Desktop Grids, super computer or hybrid platform); It encapsulates all technological details through virtual technology and can provide end users with on demand provision, non-trivial quality of service and pay by use, easy of use, high level program interface; End users can utilize all these services provided by Cloud platform in a very simple way without knowing where these services come from and on what kinds of platform/ system/ infrastructure these services run.

According to our understanding on Cloud computing, further research on Cloud computing in computational science will be made. Our general goal is that: Firstly, we will build our Private Cloud experimental platform and the Cloud platform will try to avoid the problems presented in Section 6.4. Secondly, try to develop some interfaces to make them inter-operate with Public Clouds. The detail description will be made in the next section.

6.6 A Reference architecture of scientific Private Clouds

Cloud computing platforms such as Amazon EC2 provide customers with flexible, on demand resources at low cost. However, while existing offerings are useful for providing basic computation and storage resources, they fail to consider these factors such as security, custom and policy. So many enterprises and research institutes would not like to utilize those Public Clouds. According to investigations on real requirements from scientific computing users made in China, the project “YML-PC” is started to build Private Clouds and hybrid Clouds environments for them. In this chapter, we will focus on the first step of “YML-PC” to present a reference architecture based on workflow framework YML for building scientific Private Clouds. Then some key technologies such as trust model, data persistence and schedule mechanism in “YML-PC” are discussed. Finally, some experiments are made to testify the solution presented in this paper is more efficient.

6.6.1 Introduction

Scientific computing requires an ever-increasing number of computing resources to deliver for growing sizes of problems in a reasonable time frame and Cloud computing paradigm holds good promise for the performance hungry scientific community [149]. Several evaluations have testified better performance can be achieved with lower cost by using Clouds and Cloud technology than that based on former technology. For example, papers [150] [151] make an evaluation of Cloud technology on public Clouds (e.g., EC2). Papers [152] [153] evaluate Cloud technology based on Private Clouds (e.g. cluster in internal research institute). And paper [154] shows the potential possibility to utilize volunteer computing resources to form Clouds. Papers [155][156][157] present some methods to improve the performance of Desktop Grid platform. Paper [158] analyzes the cost-benefit of Cloud computing versus Desktop Grids. Papers [159] [160] [161] introduce some Cloud solutions based on volunteer computing resources.

An investigation is made on requirements of building their scientific computing environments for non-large enterprises and research institutes in China. Those issues can be summarized as follows: First, most of enterprises and research institutes have their computing environment, but they suffer from shortage of computing resources. Second, they would not like to spend a lot of money to expand their computing resources. On the other hand, they hope they can make full use of wasted CPU cycle of individual PCs in the labs and offices. Third, they need a high level program interface to decrease their costs (time, money) on developing their applications adapting to computing environments. Last but not least, they would like to utilize their own computing environments for considering the importance and security of their data. After all, these data are bought from other corporate with high cost and they are also required to keep those data a secret. To meet these requirements mentioned above, a project has been started between university of science and technology of Lille, France and Hohai university, China. Its general goal is to build Private Cloud environment which can provide end users with a high level program interface, and users can utilize computing resources they need without considering where these computing resources comes from (i.e., the layer of program interface is independence of the layer of computing resources).

YML [117] [118] [119] is a large scale workflow programming framework, developed by PRiSM laboratories at university of Versailles and Laboratoire d'Informatique Fondamentale de Lille (LIFL, Grand Large Team, INRIA Futurs) at university of science and technology of Lille. The aim of YML is to provide users with an easy-of-use method to run parallel applications on different distributed computing platforms. The framework can be divided into three parts which are "end-users interface", "YML frontend" and "YML backend". "End-users interface" is used to provide an easy-of-use and intuitive way to submit their applications and applications can be developed using a workflow based language YvetteML. "YML frontend" is the main part of YML which includes "compiler", "scheduler", "data repository", "abstract component" and "implementation component". The role of this part is to parse parallel program into executable tasks and schedule these tasks to appropriate computing resources. "YML backend" is the layer to connect different Grid and Desktop Grid middleware through different special interfaces and users can develop these interfaces very easily. The YML is a component based framework in which components can interact with each other through well defined interfaces and researchers can add/modify one or several interfaces for other middlewares to YML very simply.

Paper [162] presents a method of resource management in Clouds through a Grid middleware. Here, we want to extend YML to build scientific Private Clouds for non-big enterprises and research institutes. We call this project as "YML-PC". Three steps are needed to make this project become reality. The first step is to integrate volunteer computing resources into dedicated computing resources through YML and make them work coordinately. Volunteer computing resources can be a supplement of dedicated computing resources and volunteer computing resources based platform have the ability to expand computing resource pool dynamically by nature. If dedicated computing resources are not enough for users, volunteer computing resources can be utilized to implement their tasks. Users don't know where their tasks are run on dedicated computing resources or volunteer computing resources, and they needn't to know. The key issue of this step is how to allocate tasks to different kinds of computing resources more reasonably and make those computing resources work coordinately with high efficiency. The second step is to develop an interface for Hadoop and make it to be integrated into YML. Then some evaluations will be made on cluster environment + Hadoop. The third step is to try to build Hybrid Clouds environment through combining step one with step two. The solution is that: step one can stand for a kind of Private Clouds and step two can be deployed on Public Clouds, then YML as a workflow based framework can harness Private Clouds and Public Clouds.

In this section, our works focus on the first step. Some research works on extending YML to YML-PC in this dissertation can be summarized as follows:

- Evaluate the hybrid computing environment (dedicated computing resources and volunteer PCs) with large scale scientific computing (Gauss Jordan algorithm as an example). As far as we know, no other evaluation is made using two different kinds of computing resources at the same time.

- Data flows. It will be added in the application file. Through adding this flow, data persistence and data anticipated migration can be realized in YML-PC. And it can help to improve the efficiency of platform greatly.

- Monitor and Trust model. They are introduced to monitor available status of non dedicated computing resources and predicate their future status. Also a method to evaluate expected execution time based on standard virtual machine is adopted. Through this method, heterogeneous computing resources can be changed into homogeneous computing resources and then can be evaluated. According to this evaluation and predication on some properties (avaiability, capability…) of computing resources, tasks can be allocated to appropriate computing resources.

The following sections will describe the design and implementation of the YML-based private Clouds in detail. Some basic evaluations also will be made. Finally some conclusions and future works will be further discussed.

6.6.2 Concept stack of Cloud platform

The part will show a detail design about how to build the environment of Cloud computing based on previous work from papers [163][131][164] [137]. As shown in Figure.6.9, generally speaking, Cloud computing can have four main layers. The base is the layer of "computing resources" and above this layer, "Operating system" and "Grid middleware" can be used to harness those different kinds of computing resources. Then "Cloud middleware" layer can help users compose applications without considering the underlying infrastructure, i.e., this layer hides different interfaces from different platforms/systems/middleware and provides a uniform, high level abstraction and easy-of-use interface for end-users. The top layer is "application layer" and Cloud platform will provide different interfaces according to different detail requirements. Business model helps to support "pay-by-use" model, and users can get the best services within their budget through "bidding mechanism".

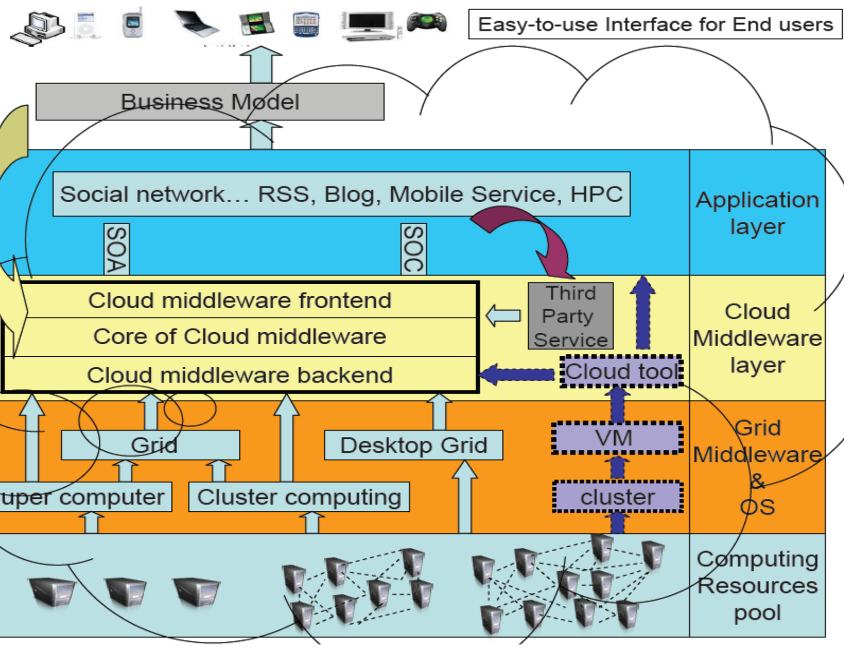


Figure 6.9. Concept stack of Cloud platform

Next, detail explanation will be made on those layers one by one:

- **Computing Resource pool** : this layer consists of different kinds of computing resources which can be clusters, supercomputer, large data center, volunteer computing resources and some devices. It aims at providing end-users with on-demand computing power.

- **Grid middleware and OS** : the role of this layer is to harness all kinds of computing resources in the computing resource pool. Some virtual machines can be generated through virtual technology (for example, Xen, VMware) based on cluster (perhaps and to be proved, also can be based on volunteer computing resources).

- **Cloud middleware layer**: in the Cloud platform, Cloud middleware can be divided into three parts according to their roles. Cloud middleware backend aims to monitor all kinds of computing resources and encapsulate those heterogeneous computing resources into homogeneous computing resources. Cloud middleware frontend is used to parse application program into executable sub-tasks. Cloud platform always provides end-users with higher level abstract interface. Through parsing the applications program, this layer can generate a file in which some necessary services (executable functions, computing resources, third-part service library) are listed. The core of Cloud middleware includes a "matchmaker factory" in which appropriate match can be made based on business model between tasks and computing resources according to their requirements and properties. Then scheduler allocates those "executable functions" and "third part services" to appropriate computing resources.

- **Application layer**: this layer is generated according to real requirements by end users based on SOA. And SOA can make sure all the interfaces from different service providers are common and easy to invoke.

- **Business model**: this model can support a pay-by-use model to end-users. It also can help end-users get the best services within their budget.

- **End user interface**: The interface must be high level abstraction and easy of use. It is very helpful for non expert computer users to utilize Cloud platform.

6.6.3 Design of YML-PC

The detail design of YML-PC is made based on concept stack of Cloud platform. See Figure.6.10. As mentioned above, the development on YML-PC can be divided into three steps. The components with *dashed border* will be developed in the second step. In this dissertation, we will focus on the first step. So the detail description will also focus on the design and implementation of first step of YML-PC. YML-PC is designed to build Private Clouds for scientific computing based on workflow. Some features of YML-PC can be summarized as:

- YML-PC can harness two kinds of computing resources at the same time and this can help to improve computing power greatly through integrating volunteer computing resources. At the same time, no extra cost is needed to do that and volunteer computing resources can also help YML-PC to have the ability of scalability in a dynamic way.

- YML-PC shields the heterogeneity of program interfaces of underlying middleware/system/ platform and provides a high level abstraction, unique interface for end-users.

- YML-PC can make full use of different kinds of computing resources according to their properties. To improve the efficiency of YML-PC, pre-scheduling and "data persistence" mechanisms are introduced into YML-PC.

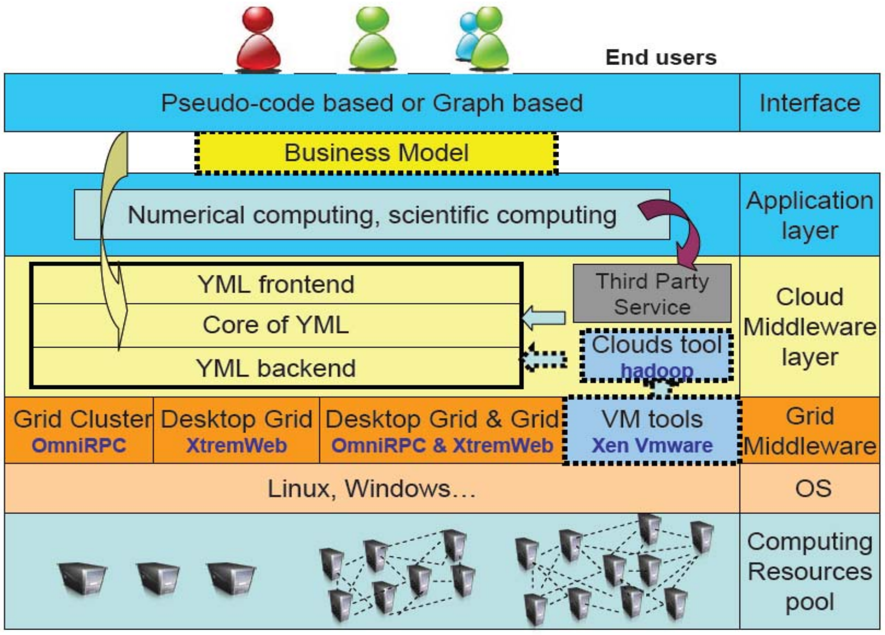


Figure 6.10. Reference Architecture of YML-PC

Now, we will explain YML-PC step by step according to its architecture layer (see Figure 6.10).

- Computing resource pool: The computing resource pool of YML-PC consists of two different kinds of computing resources: dedicated computing resources (servers or clusters) and non dedicated computing resources (volunteer PCs). As well known to us all, cluster is too expensive to scale up for a non big research institutes and enterprises. At the same time, there are a lot of PCs whose a lot of CPU cycles are wasted. So it is appealing (from the viewpoint of both economic and feasibility) to harness these two kinds of computing resources together. Computing resource pool with a lot of PCs has those features like low cost, scalability by nature and these features are key points for Clouds.

- OS: Operating system is the base of installing other software. Now YML-PC and OmniRPC (used to harness dedicated computing resources) only support Linux OS. YML-PC and XtremWeb used to collect volunteer computing resources can support both Windows OS and Linux OS.

- Grid middleware: The construction of computing resource pool in YML-PC is based on state-of-the-art technology: Grid and Desktop Grid technology. To YML-PC, we utilize Gird middleware OmniRPC to harness cluster based computing resources and Desktop Grid middleware XtremWeb to manage volunteer computing resources. As well, we can utilize these two middlewares at the same time to form computing resource pool consisting of two kinds of computing resources. As well known to us all, traditional scientific computing mostly run based on cluster or supercomputer and it is necessary to make full use of this kind of computing resources. At the same time, the power of volunteer computing is huge and it has been proved by existed volunteer platform such as Seti@home. It is very meaningful to make volunteer computing resources to be supplement/extension to the traditional computing resources.

- Application layer and Interface: The main design goal of YML-PC is for scientific computing and numerical computing. So to make scientific computing more simply, a pseudo-code based high level interface is provided to end-users.

- “YML frontend”, “YML backend” and the “core of YML” will be described in the next section.

6.6.4 Core design and implementation of YML-PC

The Figure.6.11 will show us the core design and implementation of YML-PC. We will explain these components in Figure.6.11 one by one.

YML frontend : This part is to provide end-users an easy-of-use interface and make them only focus on the design of algorithm itself. Users needn’t take low level software/hardware into consideration when they develop their application programs. About the program interface of YML-PC, we still adopt the interface of YML. For those “reusable services” (components described in section 3.2.3 of chapter 3), two ways can do that: the first method is that users can develop those “reusable services” by themselves or computer engineers; the second way is to invoke those functions from common library (e.g. LAPACK, BLAS, we also call this as “third party services”). Here what we want to emphasize is that both the pseudo-code based program and those functions developed are reusable and they both are platform-independent and system independent. System independence means that users need not to know what kinds of operating system/middleware are utilized. Platform independence stands for that their code can be run on any platform (cluster, Grids, Desktop Grids) without any change. This is to say, these codes developed by users can be reused without caring about the middleware, system and platform "YML backend" utilized. You can use OmniRPC on a grid/cluster platform or XtremWeb on a Desktop Grid platform or both, but users' code can be reused without any change.

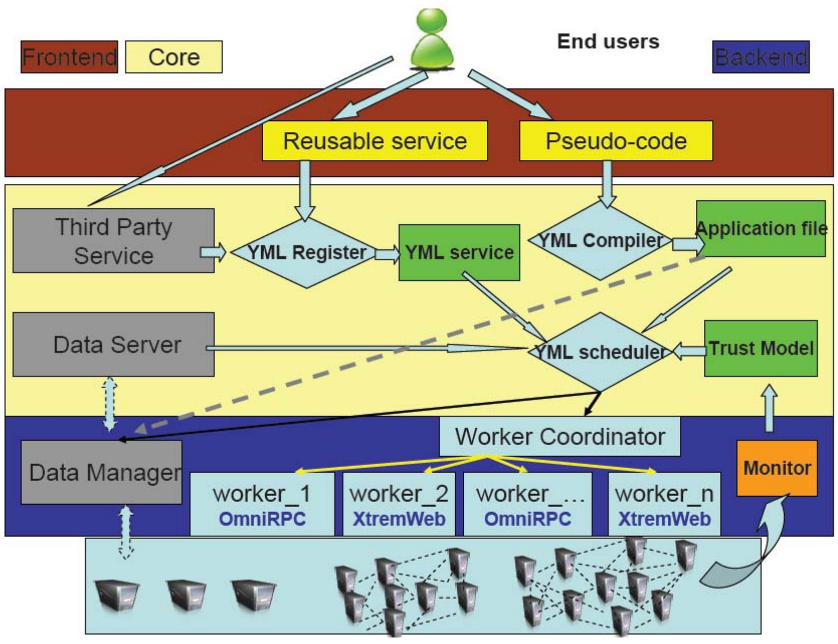
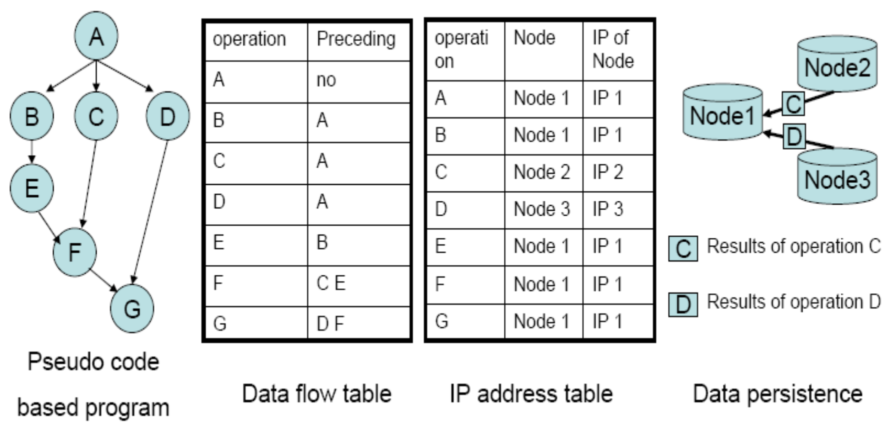


Figure 6.11. Core Part of YML-PC

The core of YML: Three components are included in this layer: *YML register, YML compiler and YML scheduler.*

YML register is used to register "reusable services" and "third party services". Once registered, these services can be invoked by "YML scheduler" automatically.

YML compiler is composed of a set of transformation stages which lead to the creation of an application file from "pseudo-code" based program. The application file consists of a series of events and operations. Events are in charge of sequence of operations. In other words, which operation can be executed in parallel/sequencial is decided by the events table. Operations refer to those services registered by "YML register". One important work made in this chapter is that "data flow table" is generated in application file. Through the "data flow table", data dependence between operations can be found (see "data flow table" in Figure.6.12). As well know to us all, these data dependences determine the execution way (in parallel/sequence) of different operations. According to these data dependence, pre-scheduling mechanism can be realized (see column "node" in "IP address table" of Figure.6.12) Then collaborating the "IP address table" (see it in Figure.6.12), data persistent and data anticipated migration can be realized. The general idea of this part of work can be described using Figure.6.12.



Data flow table

operation	Preceding
A	no
B	A
C	A
D	A
E	B
F	C E
G	D F

IP address table

operati on	Node	IP of Node
A	Node 1	IP 1
B	Node 1	IP 1
C	Node 2	IP 2
D	Node 3	IP 3
E	Node 1	IP 1
F	Node 1	IP 1
G	Node 1	IP 1

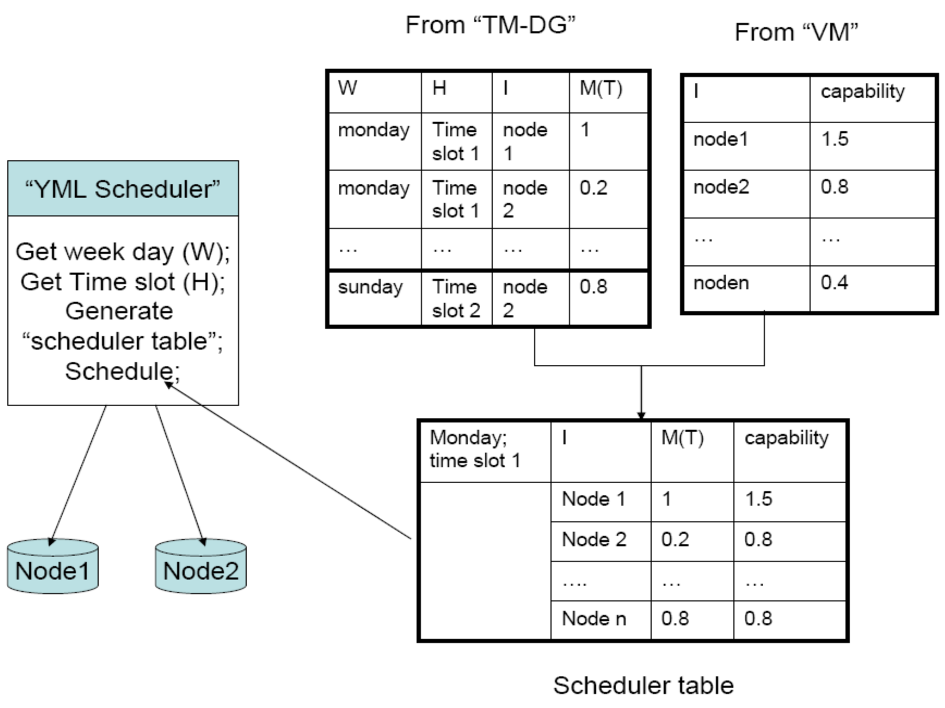
Figure 6.12. General Idea of "Data Persistence" in YML-PC

YML scheduler is a just-in-time scheduler. Its role is in charge of allocating the executable "YML services" to appropriate computing resources shielded by "YML back-end layer". "YML scheduler" is always executing two main operations sequentially. First, it checks for tasks ready for execution. This is done each time when a new event is introduced and leads to allocate tasks to the "YML back-end". The second operation is to monitor those tasks currently being executed. Once tasks have started to execute, the scheduler regularly checks whether these tasks have changed to the "finished state". The scheduler will push new task with its input data set and related "YML services" to underlying computing node when the node's state is completion or unexpected error.

To make the process presented above a reality, two parts of works are made in this chapter. The first is to introduce a monitor and predicate model for volunteer computing resources. It is well known that volatility is the key character of volunteer computing resources and if we don't know the regularity of dynamicity volunteer computing resources owned, the problem with data dependence between operations can't run on Desktop Grid platform. The reason is that frequent task migration will cause the program can't be finished for ever. We call this as "dead lock of tasks". To avoid this situation, we introduce a monitor and predict model "TM-DG" [165]. "TM-DG" is used to predict the available probability of computing nodes in the Desktop Grid during a certain time slot. The time slot depends on users' daily behaviours. For example, the availability of computing nodes in the lab has relation to students' school timetable. If students go to take classes, their computers in the lab can be utilized to make scientific computing. So the choice of time slot is related to time slot of classes. It is because two hours is needed for each class that the time slot in [165] is set as 2 hours and users can choose appropriate time slot according to its real situation. "TM-DG" collects two bodies of independence evidence, 1) percentage of completion of the allocated task, and 2) an active probe by a special test node, based on the time slot. Considering the "recommendation evidence" from other users, dempster-Shafer's theory [168] is used to combine these three bodies of evidence to get the degree of node trustworthy. The result of "TM-DG" can be expressed by a four-tuple <I, W, H, m(T)>, in which "I" represents the identity of computation node, "W" represents the day of the week, "H" represents a time interval in a day and "m" represents the probability of node availability. The four-tuple <node I, Monday, 1, 0.6> represents the time slot is from 0 to 2 a.m. on Monday, the probability of task successful execution on node "I" during this time slot is 0.6. So in this chapter, monitor component in "YML-backend" and schedule component in "Core of YML" are based on this time slot.

The second part of works: to make full use of computing resources, evaluation on capability of heterogeneous computing nodes has to be made. So a concept of standard virtual machine (VM) is proposed in this paper. The standard VM can be set in advance. For example, the VM is set through a series of parameters like that (Ps, Ms, Ns, Hs), in which "Ps" stands for CPU process power of VM (2.0 Hz CPU, for example), "Ms" represents memory of VM (1G Memory), "Ns" means network bandwidth of VM (1G) and "Hs" stands for hard space left (10 G). Users can add/minus the number of parameters according to real situation. A real computing node "Rm" can be described as (Pr, Mr, Nr, Hr). The capacity (Crm) of "Rm" can be presented as follows: "Crm"= $a1*Pr/Ps+a2*Mr/Ms+a3*Nr/Ns+a4*Hr/Hs$, in which $a1+a2+a3+a4=1$. The value of ax ($x=1, 2, 3, 4$) can be set according to different influence on final results from different parameters in real situation. We can set an appropriate value to ax ($x=1\ldots n$) based on history information. Through VM, expected execution time of tasks on a computing node can be estimated.

Scheduler can choose appropriate computing nodes according to predictions of availability of computing resources (from "TM-DG") and time needed to execute a task on this node (from "VM"). Scheduler will get the detail time and form the "scheduler table" and then schedule tasks to appropriate computing nodes. The "YML scheduler" mechanism can be described using Fig.6.13. When there is fault generated, the task will be re-scheduled. Future research about fault tolerance in YML-PC will focus on two ways: 1) allocate the same task to three or more volunteer computing nodes (Google always keep three copys for the same data block); 2) preemptive scheduling based multi-queue schedule mechanism will be examined.



W	H	I	M(T)
monday	Time slot 1	node 1	1
monday	Time slot 1	node 2	0.2
…	…	…	…
sunday	Time slot 2	node 2	0.8

I	capability
node1	1.5
node2	0.8
…	…
noden	0.4

Monday; time slot 1	I	M(T)	capability
	Node 1	1	1.5
	Node 2	0.2	0.8
	….	…	…
	Node n	0.8	0.8

Figure 6.13. Description of YML Scheduler

YML backend : YML backend encapsulates different underlying middlewares and provides consistent executable environment to tasks from the layer "core of YML". Concurrently it permits to utilize one or several middlewares through providing a specific interface for each middleware. The back-end layer consists in three parts named *Monitor*, *worker coordinator* and *data manager*. In general, YML backend send requests for executing a task on a computing node and if the task finishes, it also notifies to scheduler that the task is terminated successfully. *Data manager* is a component for managing all data exchanges between nodes and data server. This component provides two services: distributing appropriate data to workers and retrieving the final results. *Worker coordinator* maintains a list of active requests and a list of finished requests. The status can change dynamically according to the status of computing nodes. It will allocate those tasks from "YML scheduler" to appropriate computing nodes in computing resource pool. *Monitor* component is used to monitor the status of computing nodes. The monitoring mechanism is based on users' daily behavior which is adopted to predict available time of computing resources and make prediction for data migration.

6.6.5 Primary experiments on YML-PC

In this section, four primary experiments (emulations) are made to show: 1) the computing resource pool can be scaled very easily; 2) great improvements on platform efficiency can be achieved through emulating the data persistence; 3) great improvements on platform efficiency can be made through emulating appropriate task distribution between different virtual organizations.

Here Max-par BbGJ algorithm [166] [167] is used. According to the algorithm, q^2 is the number of block-counts of matrix. And the number of total tasks the algorithm will be generated is q^3. All these experiments are based on YML+OmniRPC, YML+XtremWeb, YML+OmniRPC and XtremWeb on Grid'5000 and Polytech Lille platform. In our experiments, the computational resources can be described as follows:

Table 6.1 computing resources in PolyTech Lille platform

Number of PCs	CPU	Memory
32	AMD athlon 2.2, GHz	2G
16	AMD athlon 2.7 GHz	2G
16	Pentium 2.6 GHz	2G
16	Inter celon 2GHz	1G
16	AMD athlon 2.3 GHz	2G
4	Intel Pentium 2.4GHz	512M
8	Intel Celeron 1.4GHz	512M

Table 6.2. Parts of resources in Grid'5000 platform.

Site	Nodes	CPU/Memory
Nancy	120	2 × Inter xeon , 1.6GHz/2GB
Nancy	47	2 × AMD opteron, 2GHz/2GB
Lyon	70	2 × AMD opteron, 2.4GHz/2GB
Bordeaux	93	2 × AMD opteron, 2.6GHz/2GB
Orsay	216	2 × AMD opteron, 2.0GHz/2GB
Rennes	99	2 × AMD opteron, 2.0GHz/2GB

6.6.5.1 YML-PC can get enough computing resources through collecting Volunteer computing resources

In this experiment, we will present the performance of YML-PC when it collects volunteer computing resources as supplement or extension of dedicated computing resources. To testify this character of YML-PC, we choose the Polytech Lille platform as our experiments environment. In this experiments, we will set the block-counts of sub-matrix is 8*8, i.e., at the least 8*(8-1) = 56 computing nodes can make sure all the tasks can be executed in parallel. We suppose we have 10 dedicated computing resources and clearly, it is not enough to make tasks totally executed in parallel. So we made two kinds of expansions by collecting volunteer computing resources. The first one is to use 20 volunteer computing nodes and the second case is to collect 50 volunteer computing nodes. We change the block-size as follows: 100*100, 250*250, 500*500 and 750*750. Of course, dedicated computing resources are harness through OmniRPC and volunteer computing resources are collected by XtremWeb. YML-PC can support those two kinds of middleware at the same time through Multi-backend model.

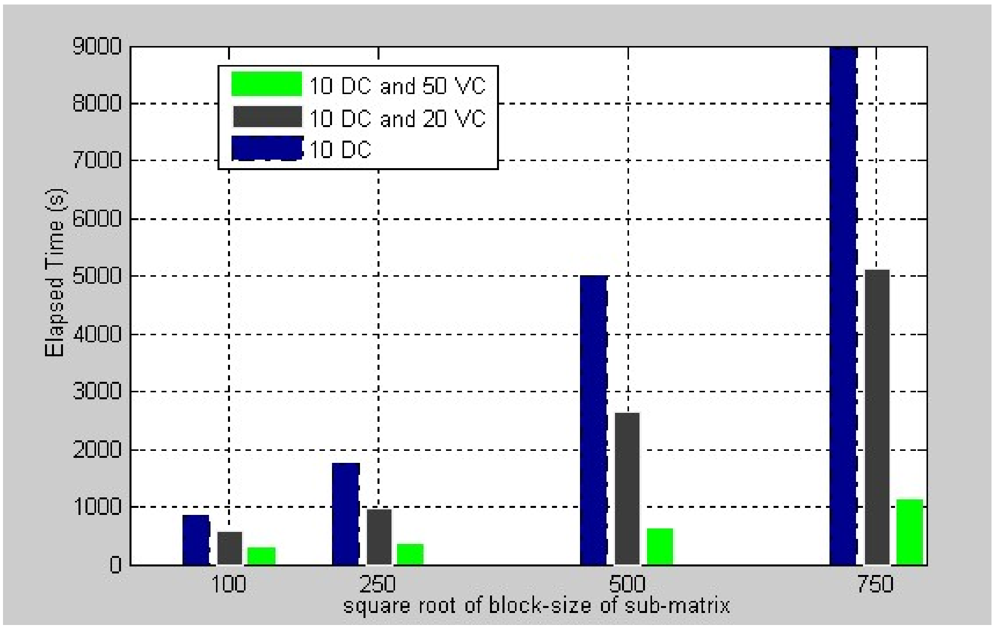


Figure 6.14. YML-PC can collect VC as the supplement of DC

To state conveniently, we call volunteer computing resources as “VC” and dedicated compouting resources as “DC”. From the Figure 6.14, we can know the performance of Max-par BbGJ algorithm using 10 DC and 20 VC is far better than that of just using 10 DC and adopting 10 DC and 50 VC can get better performance than that utilizing 10 DC and 20 VC. In summary, we can conclude that the performance of Max-par BbGJ algorithm can be improved greatly through collecting more computing power of VC. And this experiment also testifies YML-PC has the ability to harness two kinds of computing resources to enlarge the computing resources pool.

But here, we want to emphasize two points:

- The first is that, the situation for Max-par BbGJ algorithm in this experiment is that Max-par BbGJ algorithm can be improved. Just as we have explained that in the introduction of this experiment. 10 DC is not enough to make all the tasks executed in parallel. We just use this experiment to show that, when the DC is not enough for you to make large scale computing, you can choose YML-PC to collecting VC as the supplement of your DC without any extra costs. It is a very good solution for scientists to make large scale scientific computing.

- The second is that, we don't calculate the speed-up after collecting more VC into the platform. This experiment is just to show the computing resources of YML-PC can be expanded through collecting VC and we don't care about what the degree of improvement on Max-par BbGJ is. To the degree of improvement, if necessary, we will evaluate that on real platform in near future.

Summary: This experiment testifies the YML-PC support to collect VC to be supplement of DC and it is a good solution for scientific computing. The reason is that almost no costs are needed for using those VC. As will known to us all, VC based platform has the ability to support the dynamic scale up/down of computing resources. So the next experiment, we will present the computing resources of YML-PC can be scaled up though enlarge the number of VC.

6.6.5.2 YML-PC can be scaled up very easily

In this experiment, we want to express the YML-PC can be scaled up through increasing or decreasing the number of VC. We use the Grid5000 platform and we use YML and XtremWeb as our middleware. The reason is that XtremWeb can be easily scale up for its "pull model" based task allocation mechanism. In this experiment, we set block-size of sub matrix is 1500*1500.

To state more conviently, "R-B" in Figuer 6.15 represents the computing resource pool before scaling which has 10 computing nodes, while "R-A" stands for the computing resources pool scaled up which has 20 computing nodes. The progress of scale up is made during the process of program execution.

Figure.6.15 shows us that when the block-counts are less than 4*4, little influence on the elapsed time whether computing resource pool scales up or not. But when the block-count is more than 4*4, scalability of computing resource pool has an important influence on the elapsed time. The reason is from the algorithm itself. When the block-count is small, tasks generated are not much, 10 computing resources sometimes are enough for generated tasks. So the influence on the elapsed time is small. With the increasment of block-count of sub-matrix, the generated tasks increased greatly. More computing resources are needed. So the influence on elapsed time becomes more obvious. In a word, from the Figure.6.15, we can conclude that whether the block count is small or large, scalability of computing pool can help to improve the efficiency of platform. At the same time, this experiment testifies YML-PC has the ability of scalability. The scalability comes from the character of Desktop Grids platform. As well known to us all, in desktop Grid platform, when the computing resource is idle, it can ask for tasks to execute and when the computing node leaves from the platform for some reasons, the tasks executing in this node will be rescheduled. The action of "join" and "leave" will not influence the overall Desktop Grids platform. YML-PC has the ability of scalability just inheriting from Desktop Grids platform through provding a special interface to XtremWeb.

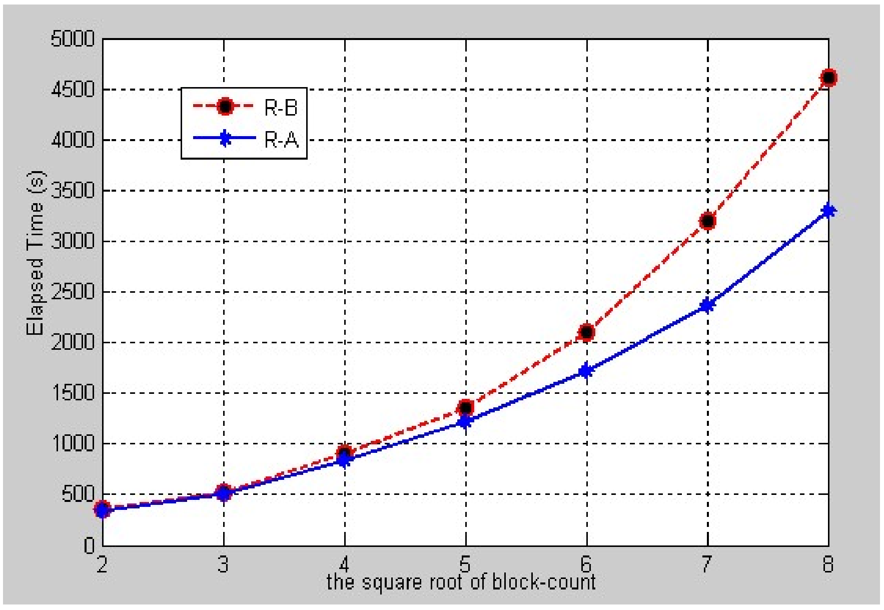


Figure 6.15. Feature of Scalability of YML-PC

6.6.5.3 Data Persistence in YML-PC

In this experiment, we want to show "data persistence" technology can help to improve the performance of YML-PC. "Data persistence" method has been introduced in section 6.6.4 of this chapter. To do this experiment, "data flow" table will be generated through parsing the application program by "YML Compiler". Tasks will be scheduled according to "data flow" to make "data persistence" become reality. The general process can be described uing Figure 6.16. Pseudo based program for YML (see appendix A) will be parsed through "YML Compiler" and then generate a "Data flow" table. The table consists of ID of computing node, name of operation, input and output. See detail in Figure 6.12.

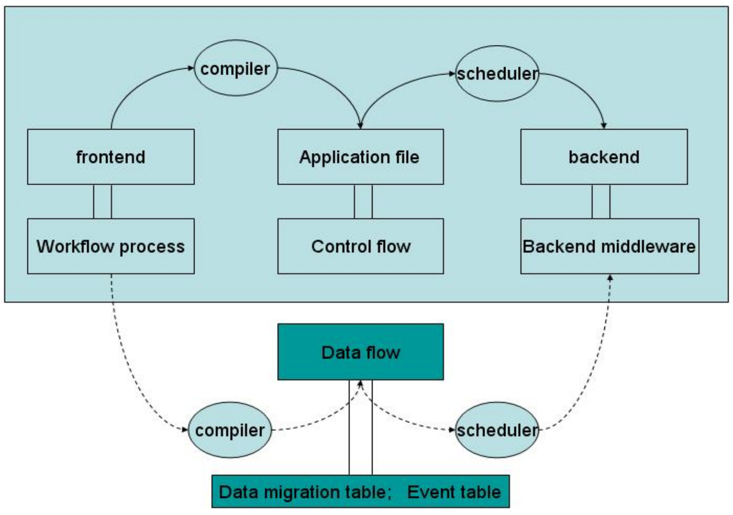


Figure 6.16. Data Flow Table of YML-PC

In this experiment, we set block-size of sub matrix is 1500*1500 and change the block-count as follows: 2*2, 3*3, 4*4, 5*5, 6*6, 7*7 and 8*8. The middleware is YML+OmniRPC. As we know, YML can generate the "data flow" table, but now, the backend OmniRPC doesn't support node-to-node based data transfer. So this experiment is simulation based. The method can be described as follows: when there is data migration (from data server to computing node) needed during the process of program running, we will generate a new 1500*1500 sub-matrix in romote computing nodes instead of transfer related data to them from data server. When a task is finished in a romote node and the result isn't the final result, we will do nothing. We just write the final result to the data server.

Through parsing YML-PC based Max-par BbGJ algorithm, we can get the data transfer model like Figuer.6.17. The Figure shows the siutaion of data transfer model in BbGJ algorithm which is divided into 5*5 blocks. From chapter 4, we know there are five iterative steps during the process of program running and the Figure shows the data transfer model between step one and step two. Line reprents the data transfer in the same iterative step and dot line stands for the data transfer between steps.

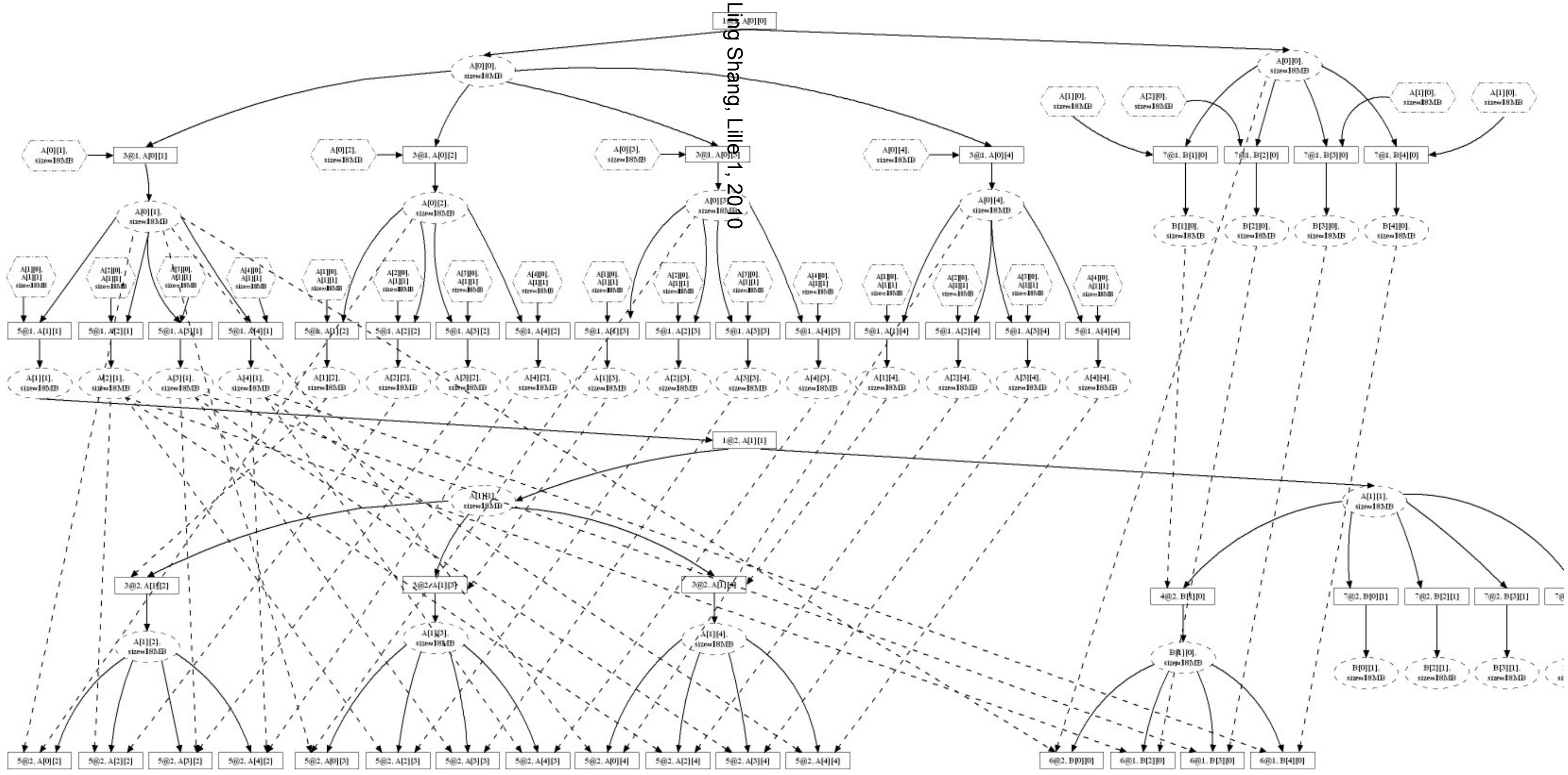


Figure 6.17. Data transfer model in Max-par BbGJ algorithm

The final simulation results can be described using Figure.6.18.

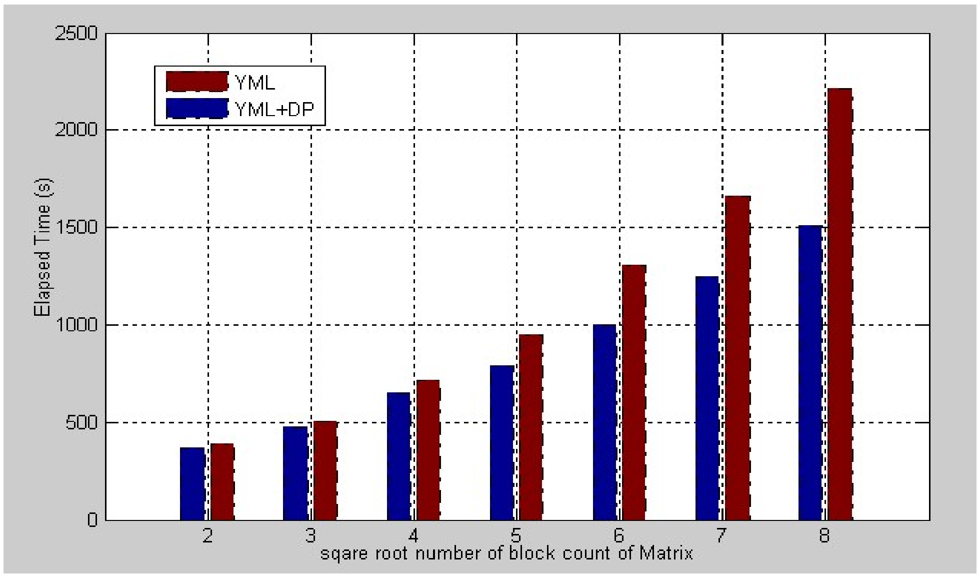


Figure 6.18. Data persistence in YML-PC

Figure.6.18 shows that data persistence is very important for scientific computing platform, especially to scientific computing with much data to deal with. It can save a lot of time and thus improve the efficiency of platform. With the increase of block count of sub-matrix, more tasks are generated and therefore a lot of data transfers between data server and workers are generated. The better performance can be achieved under the help of "data persistence" technology. So the future work will focus on develp an interface for the middleware which support node-to-node based data transfer (as far as we know, SmartGridSolve support node-to-node data transfer model, so perhaps we will develop a new component for YML to support the SmartGridSolve in near future).

6.6.5.4 Schedule mechanism in YML-PC

One of most important advantages of YML-PC is it support harnessing two kinds of computing resources at the same time. As we stated in chapter 5, to volunteer computing resources based platform, schedule mechanism is very important. So this experiment will show that appropriate selecting computing resources based on trust model in YML-PC is also important. In this experiments we set block-size is 1500*1500 and change block-count as follow: 2*2, 3*3, 4*4, 5*5, 6*6 and 7*7. The middleware is YML+Xtremweb. The experiment is based on Grid5000 platform.

Suppose, we can get the totally correct information about available time of volunteer computing resources through “TM-DG” model. In fact, the performance of “TM-DG” has been proved in paper [165] and [169] and it can achieve better performance in real Desktop Grid environment. We also create other two situations through disconnect computing nodes from the platform. Once case is 10% of total computing nodes will leave from the platform and the other is to let 20% of total number of computing nodes be removed from the platform.

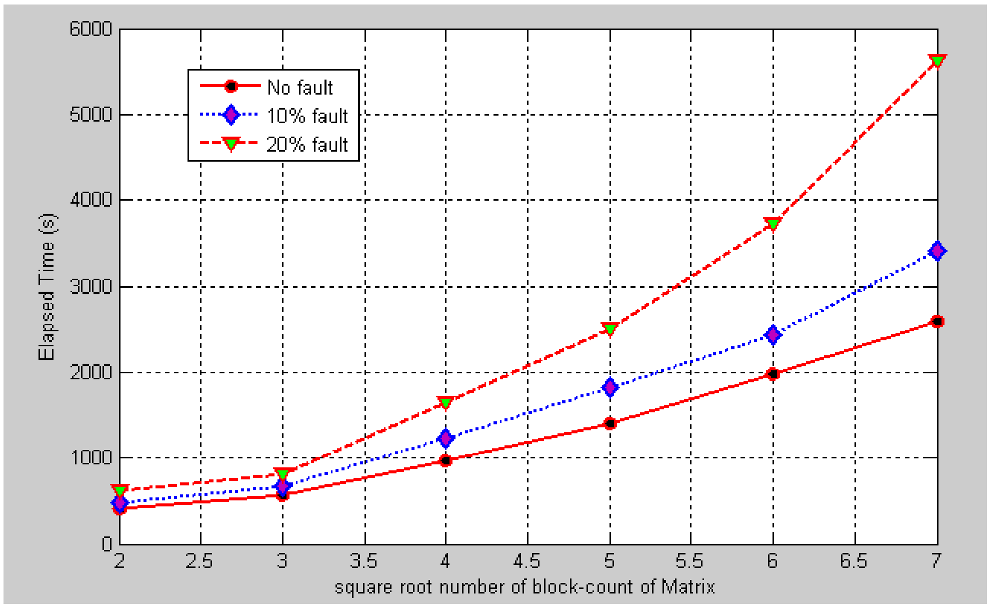


Figure 6.19. Schedule mechanism in YML-PC

‘No fault’ in Figure.6.19 represents the no faults happen during the process of program execution. In other words, the trust model is totally correct. ‘10% faults’ stands for 10% of computing nodes in YML-PC fail during the process of program execution. In other word, the accurate rate of trust model is 90%. ‘20% faults’ stands for 20% of computing nodes in YML-PC fail during the process of program execution. This is to say, the accurate rate of trust model is 80%.

Figure.6.19 testifies appropriate computing resources are chosen to execute tasks is very important. Improper match making between computing resources and tasks will decrease the efficiency greatly. So monitoring the volunteer computing resources in YML-PC is very important and we had better find the regularity of available computing nodes behind its appearance through monitoring. Trust model in paper [165] can be utilized in YML-PC which consists of volunteer computing resources and it can be improved by adopting better behavior model to describe users’ behavior regularity.

6.6.6 Conclusion and future work

Cloud computing has gained great success on search engine, social e-network, e-mail and e-commercial. Amazon can provide different level computing resources to users in the way of pay-by-use. Many research institutes, such as university of Berkeley, Delft University of Technology and so on, have made evaluations on Amazon Cloud platform. At the same time, Kondo et al try to evaluate the cost-benefits of public Clouds and Desktop Grid platform. Its conclusion shows that Desktop Grid platform is promising and it can provide on demand computing resources for Cloud platform with low costs. So based on the researches mentioned above and real situation of non-big enterprises and research institutes in China, this paper extends YML framework and presents YML-PC, which is a workflow based framework for building scientific private Clouds. The project YML-PC will be divided into three steps:

- Build private Clouds based on YML through harnessing dedicated computing resources and volunteer computing resources and make them work coordinately with high efficiency.

- Extend YML to support Hadoop and run Hadoop on cluster based virtual machines. Through Hadoop, YML's application can support Public Cloud platform (such as Amazon EC2).

- Combine step 1 and step 2, build a Hybrid Clouds based on YML to utilize computing resources both in Public Clouds and Private Clouds. This paper focuses on step 1. To improve the efficiency of YML-PC, "trust model" and "data persistence mechanism" are introduced in this paper. Simulations testify our idea is on the right way to build YML-PC.

Future works will focus on developing components to make YML-PC a reality. Then more users' behavior model will be researched to improve the accuracy of prediction on available "time slot" of volunteer computing nodes. Fault-tolerant based schedule mechanism is another key issue of our future work. A new idea, which is to deploy virtual tool (Xen, VMware for example) on volunteer computing resources and form several virtual machines on volunteer computing node, is to be evaluated.

6.7 Conclusion

The works of this chapter can be devided into two parts:

- The fisrt part is mainly to introduce state of the art technology on Cloud computing and summarize its common characters. According to the survey of existing Cloud platforms, viewpoints on Cloud computing from many famous scientists and our experiences of scientific computing on Grids, we present our understanding on Cloud computing.

- The second part is to present an reference architecture (YML-PC) for building Private Cloud platform according to our works in part one and real requirements from non-big enterprises and research institutes. YML-PC is an extension of YML project and the main works on extending YML in this dissertation focus on how to make dedicated computing resources and volunteer PCs work coordinately. The details can be described as follows: some evaluations based on two kinds of computing resources are made using Gauss Jordan algorithm. Then some technologies which help to improve the efficiency of platform (data persistence, general evaluation on capability of computing resources, appropriate schedule mechanism) are discussed and emulated on Grid and Desktop Grid environments.

Cloud computing arouses great interests from scientific researchers and IT enterprises since 2008. A lot of people present their understandings on Cloud computing and many products have been launched by some famous IT enterprises such as Google, IBM, Amazon and Microsoft. But up to now, no agreement is achieved on what Cloud computing is. This chapter analyzed some famous Cloud architectures provided by famous enterprises and research institutes and found the common features behind different appearances of different Clouds, which are easy-of-use, low cost, high performance and services on demand… Then a special view from Grids to observe Clouds is made and summarized a series of problems exposed in the process of using Grid systems (for example, our experience on Grid5000 and other experiences on world wide famous Grid projects, such as TeraGrid, EGEE). Those problems can be summarized as follows: Grids platform is hard to use; Grid middleware is hard for end users to program; schedule mechanism in many real Grid systems is the bottleneck of achieving better performance; data transfer model in many Grid middlewares is based on server-worker-server model. According to those insignificances presented above of existing Grid systems, our viewpoint on Cloud computing is proposed and a definition is made. Our works on Cloud computing is based on this definition. We also summarized some other viewpoints of Cloud computing on type and deployment type. Generally speaking, Cloud system can be divided into Private Clouds, Public Clouds and Hybrid Clouds according to its deployment type. It also can be classified into Software as a service, Platform as a service and Infrastructure as a service according to its type. According to our definition and real requirements from non-big enterprises and research institutes, a kind of Private Cloud platform which belongs to Platform as a service, will be proposed. In this Private Cloud platform, we will try to improve the insignificances presented above of Grid systems.

Based on our research on Cloud computing, YML-PC a reference architecture based on workflow for build Private Cloud platform, is presented to make scientific computing for end users in a simple way. The most advantage of YML-PC is that it can harness two kinds of computing resources which are dedicated computing resouces and volunteer computing resources, at the same time. Through collecting volunteer computing resources, the computing power can be improved greatly without any extra costs. Volunteer computing also can help YML-PC to have the ability of dynamic sacalability. To make sure the two kinds of computing resources can work coordinately, a trust mode "TM-DG" is introduced to predicate tha available time of volunteer computing resources. To improve the performance of YML-PC, data persistent technology is also introduced into YML-PC. Finally, some primary experiments on our Private Clouds tesify our viewpoints.

Chapter 7

Conclusion and Future Works

The goal of this dissertation is to find a solution to make large scale scientific easily and try to reduce time/costs to solution.

To achieve the target, this dissertation makes summary on evolution of large scale scientific computing platform. Nowadays, Grids and Desktop Grids are two main technologies for large scale scientific computing. So this dissertation makes a survery on Grid and Desktop Grid computing and introduces some famous Grid and Desktop Grid platforms and middlewares.

To evaluate the experimental environments and tools, research on a classical numerical algorithm Gauss Jordan algorithm which is used to invert large scale matrix has been made in this dissertation. Through analysis on data dependence between operations, a new parallel programming adapted version (Max-par BbGJ algorithm) for block based Gauss Jordan is presented. The advantage of Max-par BbGJ algorithm is that it considers both intra-iterative step based parallelism and inter-iteratives step based parallelism, while the tradtional one just takes intra-iterative step based parallelism into consideration. Max-par BbGJ algorithm makes sure each operation in algorithm can be executed in parallel when its executable conditions are met, i.e., all the operations can be executed totally in parallel. Experiments testify Max-par BbGJ can achieve better performance than traditonal one when the computing resoruces are enough. At the same time, BbGJ algorithm is used to evaluate different experimental environments and tools.

To find a way to make large scale scientific computing easily, research on programming model on different middlewares has been made. Experiments show that YML can provide a higher level programming interface which is very easy to use for end users. What we want to emphasize here is that OmniRPC and XtremWeb are also provide high level interface, but they are still difficult for end users to program. But the interface of YML is pseudo code based and user can develop their application without knowing any knowledge about programming. Also the code developed using YML can be run both on Grid systems and Desktop Grid systems. This is very important feature for YML and as far as we know, no other middleware can support both Grid enviroments and Desktop Grid environments. Then evaluation on overhead of YML is made and experiments show YML has acceptable overhead. More importantly, the component in YML can be reused in different algorithm which can decrease the burden of users' programming. Users just pay more attention on algorithm itself and develop pseudo code based program for YML. This will reduce time and costs to solution of their scientific computing greatly. To summary, YML is a good solution for making large scale scientific computing. To get better performance, influence from task granularity on different experimental environment is analyzed and tests show fine-grain based task parallel programs don't always achieve better performance than that of coarse-grain based in Desktop Grid environment. Experiments also tesify the data persistence technology can help to improve the performance in both Grid and Desktop Grid environments. Finally, schedule mechanism should take heterogenous (CPU, network, memory) into consideration. Only in this way, stable performance can be achieved in Desktop Grid environment.

Based on analysis on Grid environment and Desktop Grid environment made above and the characters of Cloud computing, a reference architecture (YML-PC) for building Private Clouds is proposed in this dissertation. YML-PC is an extension of YML. The computing resources pool of YML-PC consists of two kinds of computing resources which are dedicated computing resouces and volunteer computing resources. Volunteer computing resources can be the extension/supplement of dedicated computing resures. There are two adavantages to do that. First, processing power of volunteer computing resources is huge and it is easy to scale up/down the capability of platform. Through collecting volunteer computing resouces, YML-PC can provide computing power on demand. Second, the costs for utitlizing volunteer computing resouces are very low and it can decrease the costs of making large scale scientific computing. YML-PC also inherits a series of good features of YML in its easy-to-use interface and component reuse. This helps YML-PC to become a good solution for scientific computing. To make dedicated computing resources and volunteer computing resources work coordinately and efficiently, "TM-DG" and "monitor" model are introduced which help to predicate the regularity of avaialbe time of volunteer node in Desktop Grid environment. To get better performance of

YML-PC, “data transfer model” is also discussed and “data persistence” technology is introduced in YML-PC. Finally, experiments tesifiy the computing resouces of YML-PC can be scaled up very easily, thus the performance of platform can be improved greatly. Simulation of data perisitence on YML-PC show that “data persistence” can help to improve the performance of YML-PC. Experiments also tesify “TM-DG” and “Monitor” model can help to improve the performance of platform through allocating tasks to appropriate compouting resources (the judgement on whether the computing resrouce is appropriate or not is based on available time predicition from “TM-DG” and general evaluation based on “VM”).

Future works will focus on making YML-PC a reality.

First, develop an interface for a middleware which supports worker-to-worker data transfer model. This helps to realize “data persistence” in YML-PC.
Second, make some experiments on volunteer computing resources based on virtual technology (xen or VMare based). The aim of this work is to judge whether virtual machine can help to improve the performance of Desktop Grid environment or not.
Third, develop an interface for Hadoop and thus YML-PC can support Hadoop. Some experiments based on Public Clouds through Hadoop will be made.
Fourth, Hybrid Clouds based on XtremWeb, OmniRPC and Hadoop can be built and some evaluations will be made on them.

Appendix

Max-Par BbGJ for YML

```
<?xml version="1.0"?>
<application name="gauss_jordan">
            <description> produit matriciel pour deux matrice carree
            </description>
<graph>
     par
            seq(k:= 0; blockcount-1)
                 do
                         wait(flag[k-1][k][k]);
                        compute inversion(A[k][k],B[k][k],blocksize,blocksize);
                         compute EvalMat(B[k][k],A[k][k],blocksize);
                        notify(flag[k][k][k]);
                         notify(flag[k][k][k+blockcount]);
                 enddo

//
            par(k:=0; blockcount-2) (j:=k+1; blockcount-1)
                 do
                        wait(flag[k][k][k] and flag[k-1][k][j]);
                        compute prodMat(A[k][k],A[k][j],blocksize);
                        notify(flag[k][k][j]);
                 enddo
//
            par(k:=1; blockcount-1) (j:=0; blockcount-1)
               do
                     if (k gt j) then
                      wait(flag[k][k][k] and flag[k-1][k][j+blockcount]);
                           compute prodMat(A[k][k],B[k][j],blocksize);
                           notify(flag[k][k][j+blockcount]);
                      endif
               enddo
//
            par(k:=0; blockcount-1) (i:=0; blockcount-1)
               do
                     if (i neq k) then
                          wait(flag[k][k][k] and flag[k-1][i][k]);
                          compute mProdMat(A[i][k],A[k][k],B[i][k],blocksize);
                          notify(flag[k][i][k+blockcount]);
                     endif
                 enddo
//
            par(k:=0; blockcount-2) (i:=0; blockcount-1) (j:=k+1; blockcount-1)
                 do
                      if (i neq k)    then
                          wait(flag[k][k][j] and flag[k-1][i][k] and flag[k-1][i][j]);
                           compute prodDiff(A[i][k],A[k][j],A[i][j],blocksize);
                           notify(flag[k][i][j]);
                      endif
                   enddo
//
             par(k:=1; blockcount-1) (i:=0; blockcount-1) (j:=0; k-1)
```

```
                do
                   if (i neq k) then
                        wait(flag[k][k][j+blockcount] and flag[k-1][i][k] and flag[k-1][i][j+blockcount]);
                        compute prodDiff(A[i][k],B[k][j],B[i][j],blocksize);
                        notify(flag[k][i][j+blockcount]);
                   endif
                enddo
endpar
</graph>
</application>
```

Bibliography

[1] Scientific Computing, Editor-in-Chief Gene H. Golub, Springer (1997)

[2] McCormick, B. H. 1988. Visualization in scientific computing. SIGBIO News l. 10, 1 (Mar. 1988), 15-21.

[3] G. S. Almasi and A. Gottlieb. Highly parallel computing. 1989.

[4] D.E.Culler, J.P.Singh, and A.Gupta. Parallel Computer Architecture: A Hardware Software Approach. Morgan Kaufmann Publishers Inc., August 1998.

[5] M.J. Flynn. Some computer organizations and their effectiveness. IEEE Transactions on Computing, C-21:948_960, 1972.

[6] Hwang, K. 1992 Advanced Computer Architecture: Parallelism, Scalability, Programmability. 1st. Mc Graw-Hill Higher Education.

[7]A. Barak and O. La'adan. The mosix multicomputer operating system for high performance cluster computing. Journal of Future Generation Computer Systems, 13(4/5): 361_372, March 1998.

[9]S. Dwarkadas, P. Keleher, A.L. Cox, and W. Zwaenepoel. An evaluation of software distributed shared memory for next-generation processors and networks. In Proceedings of the Twentieth Symposium on Computer Architecture, pages 144_155, May 1993.

[10]P. Keleher, S. Dwarkadas, A.L. Cox, and W. Zwaenepoel. Treadmarks: Distributed shared memory on standard workstations and operating systems. In Proceedings of the Winter 94 Usenix Conference, pages 115_131, January 1994.

[11] Foster, I. and Kesselman, C. 1999. The Globus toolkit. In the Grid: Blueprint For A New Computing infrastructure, Morgan Kaufmann Publishers, San Francisco, CA, 259-278.

[12] Foster, I., Kesselman, C., and Tuecke, S. 2001. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. Int. J. High Perform. Comput. Appl. 15, 3 (Aug. 2001), 200-222.

[13] Foster, I: What is the Grid? A three point checklist. http://www-fp.mcs.anl.gov/~foster/Articles /What Is The Grid.pdf

[14]David P. Anderson, Gilles Fedak: The Computational and Storage Potential of Volunteer Computing. CCGRID 2006: 73-80

[15]Foster, I. and Kesselman, C. Globus: A Toolkit-Based Grid Architecture. The Grid: Blueprint for a New Computing Infrastructure, Morgan Kaufmann, 1999, 259-278.

[16]M. Sato, T. Boku, and D. Takahashi, “OmniRPC: a Grid RPC system for Parallel Programming in Cluster and Grid Environment.” in The 3rd IEEE International Symposium on Cluster Computing and the Grid, 2003, pp. 206–.

[17]M. Beck, J. Dongarra, and J. S. Plank, “Netsolve: A massively parallel grid execution system for scalable data intensive collaboration,” IPDPS, vol. 11, p. 223a, 2005

[18]Bernholdt, D., Bharathi, S., Brown, D., Chanchio, K., Chen, M., Chervenak, A., Cinquini, L., Drach, B., Foster, I., Fox, P., Garcia, J., Kesselman, C., Markel, R., Middleton, D., Nefedova, V., Pouchard, L., Shoshani, A., Sim, A., Strand, G. and Williams, D. The Earth System Grid: Supporting the Next Generation of Climate Modeling Research. Proceedings of the IEEE, 93 (3). 485-495. 2005.

[19] G. Fedak, C. Germain, V. Neri, and F. Cappello, "XtremWeb: A Generic Global Computing System," The 1st IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID 2001): pp. 582-587, May 2001.

[20] D.P. Anderson, BOINC: A system for public-resource computing and storage, The Fifth IEEE/ACM Intl. Workshop on Grid Computing, GRID'04, Nov. 2004, pp .4-10.

[21]M. Baker, R. Buyya and D. Laforenza The Grid: International Efforts in Global Computing. Proceedings of the International Conference on Advances in Infrastructure for Electronic Business, Science ,and Education on the Internet (SSGRR 2000), l`Aquila, Rome, Italy, July 31 - August 6. 2000.

[22]I. Foster, J. Geisler, W.Nickless, W. Smith, and S. Tuecke. Software infrastructure for the I-way high performance distributed computing experiment. In In Proceedings of the 5th IEEE Symposium on High Performance Distributed Computing, pages 562-571, 1997.

[23]J. Nabrzyski, J. M. Schopf, J. W. Eglarz Grid Resource Management: State of the Art and Future Trends. Kluwer Academic Publisher. 2003.

[24]M. D. Brown et al., The International Grid (iGrid): Empowering Global Research Community Networking Using High Performance International Internet Services, April 1999, http://www.globus.org/research/papers.html

[25]S. Smallen et al., Combining Workstations and Supercomputers to Support Grid Applications: The Parallel Tomography Experience, 9th Heterogenous Computing Workshop (HCW 2000, IPDPS), Mexico, 2000

[26]Menascá, D. A. and Casalicchio, E. 2004. QoS in Grid Computing. IEEE Internet Computing 8, 4 (Jul. 2004), 85-87.

[27]Toma, I., Foxvoug, D., Jaeger, M. C., Roman, D., Strang, T., Fensel, D.: Modeling QoS Characteristics in WSMO. In: the Middleware for Service Oriented Computing Workshop (MW4SOC 2006), Melbourne, Australia (2006)

[28]Zhang, C., Chang, R.N., Perng, C., So, E., Tang, C., Tao, T.: QoS-Aware Optimization of Composite-Service Fulfillment Policy. In: IEEE SCC. pp. 11-19. IEEE Press (2007)

[29]M. Baker, R. Buyya, and D. Laforenza, Grids and Grid Technologies for Wide-Area Distributed Computing, International Journal of Software: Practice and Experience (SPE), Volume 32, Issue 15, Pages: 1437-1466, Wiley Press, USA, December 2002.

[30]Yeo, C.S., Buyya, R., de Assuncao, M.D., Yu, J., Sulistio, A., Venugopal, S., Placek, M.: Utility computing on global Grids. In: The Handbook of Computer Networks. John Wiley & Sons, New York, USA (2007)

[31]Vaidy Sunderam: The PVM system: Status, trends, and directions. Parallel Virtual Machine — EuroPVM '96, springer pages 68-80

[32]V. S. Sunderam, "PVM: A Framework for Parallel Distributed Computing", Journal of Concurrency: Practice and Experience, 2(4), pp. 315–339, December 1990.

[33]V. S. Sunderam, G. A. Geist, J. J. Dongarra, and R. Manchek, "The PVM Concurrent Computing System: Evolution, Experiences, and Trends", Journal of Parallel Computing, 20(4), pp. 531–546, March 1994.

[34]A. Zadroga, A. Krantz, S. Chodrow, V. Sunderam, "An RPC Facility for PVM", Proceedings — High-Performance Computing and Networking '96, Brussels, Belgium, Springer-Verlag, pp. 798–805, April 1996.

[35] P. Gray and V. Sunderam. Metacomputing with the IceT system. International Journal of High Performance Computing Applications, 13(3):241–252, 1999.

[36] Gray,P.A. and Sunderam, V. S. 2002. Collaborative Metacomputing with IceT. J. Supercomput. 23, 2 (Sep. 2002), 139-166.

[37] Foster, I. and Karonis, N. T. 1998. A grid-enabled MPI: message passing in heterogeneous distributed computing systems. Proceeding of the 1998 ACM/IEEE Conference on Supercomputing. IEEE Computer Society, Washington, DC, 1-11.

[38]W. Gropp, E. Lusk, and A. Skjellum, Using MPI: Portable Parallel Programming with the Message-Passing Interface, MIT Press, 1999.

[39]W. Gropp, E. Lusk, and R. Thakur, Using MPI-2: Advanced Features of the Message-Passing Interface, MIT Press, 1999.

[40]Geist, A., Gropp, W., Huss-Lederman, S., Lumsdaine, A., Lusk, E. L., Saphir, W., Skjellum, T., and Snir, M. MPI-2: Extending the Message-Passing Interface. Euro-par 96. Lecture Notes In Computer Science, vol. 1123. Springer-Verlag, London, 128-135.

[41] Gropp, W., Lusk, E., Doss, N., and Skjellum, A. A high-performance, portable implementation of the MPI message passing interface standard. Parallel Computing. 22, 6 (Sep. 1996), 789-828.

[42]G. Burns, R. Daoud, and J. Vaigl. LAM: An Open Cluster Environment for MPI. In Proceedings of Supercomputing Symposium, pages 379–386, 1994.

[43]A. Lastovetsky and R. Reddy. HeteroMPI: Towards a Message-Passing Library for Heterogeneous Networks of Computers. Journal of Parallel and Distributed Computing, 66(2):197 – 220, 2006.

[44]G. Fagg, E. Gabriel, G. Bosilca, T. Angskun, Z. Chen, J. Pjesivac-Grbovic, K. London, and J. Dongarra. Extending the mpi specification for process fault tolerance on high performance computing systems. International Supercomputing Conference (ISC2004), 2004.

[45]N. Karonis, B. Tonnen, and I. Foster. MPICH-G2: a gridenabled implementation of the message passing interface. Journal of Parallel and Distributed Computing, 63(5):551–563, May 2003.

[46]Rabenseifner, R. and Schuch, A., "Comparison of DCE RPC, DFN-RPC ONC and PVM", in Proceedings of the Workshop on OSF DCE, LNCS, pp. 253-259, Karlsruhe, Germany, October 1993.

[47]R. Rabenseifner, The DFN Remote Procedure Call Tool for Parallel and Distributed Applications, in: In Kommunikation in Verteilten Systemen - KiVS 95. K. Franke, U. Huebner, W. Kalfa (Editors), Proceedings, Chemnitz- Zwickau, 1995, pp. 415-419.

[48]Johnson, D. B. and Zwaenepoel, W. 1993. The Peregrine high-performance RPC system. Softw. Pract. Exper. 23, 2 (Feb. 1993), 201-221.

[49]Chang, C., Czajkowski, G., and Von Eicken, T. 1999. MRPC: a high performance RPC system for MPMD parallel computing. Softw. Pract. Exper. 29, 1 (Jan. 1999), 43-66.

[50]Djilali, S., Herault, T., Lodygensky, O., Morlier, T., Fedak, G., and Cappello, F. 2004. RPC-V: Toward Fault-Tolerant RPC for Internet Connected Desktop Grids with Volatile Nodes. In Proceedings of the 2004 ACM/IEEE Conference on Supercomputing. IEEE Computer Society, Washington, DC, 39.

[51]Seymour, K., Nakada, H., Matsuoka, S., Dongarra, J., Lee, C., and Casanova, H. 2002. Overview of GridRPC: A Remote Procedure Call API for Grid Computing. In Proceedings of the Third international Workshop on Grid Computing M. Parashar, Ed. Lecture Notes In Computer Science, vol. 2536. Springer-Verlag, London, 274-278.

[52]Y. Huang: JISGA: A Jini-Based Service-Oriented Grid Architecture, International Journal of High Performance Computing Applications, 17(3), 2003, pp. 317-327.

[53]N. Furmento, J. Hau, W. Lee, S. Newhouse, and J. Darlington: Implementations of a Service-Oriented Architecture on top of Jini, JXTA and OGSI. In Second Across Grids Conference, Nicosia, Cyprus, 2004

[54]L. Gong. JXTA: A Network Programming Environment. IEEE Internet Computing,5(3), 2001.

[55]van Nieuwpoort, R. V., Maassen, J., Hofman, R., Kielmann, T., and Bal, H. E. 2002. Ibis: an efficient Java-based grid programming environment. In Proceedings of the 2002 Joint ACM-ISCOPE Conference on Java Grande. JGI '02. ACM, New York, NY, 18-27.

[56]Kielmann, T., Bal, H. E., Maassen, J., van Nieuwpoort, R., Eyraud, L., Hofman, R., and Verstoep, K. 2002. Programming environments for high-performance grid computing: the Albatross project. Future Gener. Comput. Syst. 18, 8 (Oct. 2002), 1113-1125.

[57]Jason Maassen, Thilo Kielmann, and Henri E. Bal. GMI: Flexible and efficient group method invocation for parallel programming. In Sixth Workshop on Languages, Compilers, and Run-time Systems for Scalable Computers, Washington DC, March 2002.

[58]I. Foster, C. Kesselman, J. Nick, and S. Tuecke. Grid Services for Distributed System Integration. IEEE Computer, pages 37–46, June 2002.

[59]Chris Johnson , Steve Parker and David Weinstein, "Component-Based Problem Solving Environments for Large-Scale Scientific Computing", Concurrency and Computation: Practice and Experience Vol. 14, Grid Computing environments Special Issue 13-14, 2002.

[60] L. Smith and M. Bull. Development of mixed mode mpi/openmp applications. In WOMPAT, 2000.

[61]B. Carpenter et al. MPJ: MPI-like Message-passing for Java. Concurrency: Practice and Experience, 12(11):1019–1038, 2000.

[62]D. Marinescu and C. Lee, editors. Process Coordination and Ubiquitous Computing. CRC Press, 2002.

[63]R. Tolksdorf. Models of coordination and web-based systems. In D. Marinescu and C. Lee, editors, Process Coordination and Ubiquitous Computing. CRC Press, 2002.

[64]N. Furmento, A. Mayer, S. McGough, S. Newhouse, T. Field, and J. Darlington. An Integrated Grid Environment for Component Applications. In Second InternationalWorkshop on Grid Computing (Grid 2001), pages 26–37, 2001. LNCS Vol. 2242.

[65]L. Fischer, editor. The Workflow Handbook 2002. Future Strategies, Inc., 2002.

[66]I. Foster, C. Kesselman. The Globus Project: A Status Report. Proc. IPPS/SPDP '98 Heterogeneous Computing Workshop, pp. 4-18, 1998.

[67]I. Foster, C. Kesselman. Globus: A Metacomputing Infrastructure Toolkit. Intl J. Supercomputer Applications, 11(2):115-128, 1997.

[68]I. Foster. Globus Toolkit Version 4: Software for Service-Oriented Systems. IFIP International Conference on Network and Parallel Computing, Springer-Verlag LNCS 3779, pp 2-13, 2006.

[69]Berman F, Wolski . The AppLeS project: A status report. Proceedings of the 8th NEC Research Symposium, May 1997

[70]H. Casanova, G. Obertelli, F. Berman, R. Wolski, The AppLeS parameter sweep template: user-level middleware for the Grid, in: Proceedings of SuperComputing 2000 SC'00, November 2000.

[71]Francine Berman , Richard Wolski , Henri Casanova , Walfredo Cirne , Holly Dail , Marcio Faerman , Silvia Figueira , Jim Hayes , Graziano Obertelli , Jennifer Schopf , Gary Shao , Shava Smallen , Neil Spring , Alan Su , Dmitrii Zagorodnov, Adaptive Computing on the Grid Using AppLeS, IEEE Transactions on Parallel and Distributed Systems, v.14 n.4, p.369-382, April 2003

[72]S. Wells, Legion 1.8 Basic User Manual. Charlottesville, VA: Dept. Comput. Sci., Univ. Virginia, 2003.

[73]Chapin, S. J., Katramatos, D., Karpovich, J. F., Grimshaw, A. S., Resource Management in Legion, University of Virginia Technical Report CS-98-09, February 1998.

[74]A. Natrajan, M. Humphrey, and A. S. Grimshaw, "The Legion support for advanced parameter-space studies on a grid," Future Gener. Comput. Syst., vol. 18, pp. 1033–1052, 2002.

[75]D. Arnold, S. Agrawal, S. Blackford, J. Dongarra, M. Miller, K. Sagi, Z. Shi, and S. Vadhiyar. Users' Guide to NetSolve V1.4. Computer Science Dept. Technical Report CS-01-467, University of Tennessee, Knoxville, TN, July 2001.

[76]H. Casanova and J. Dongarra, NetSolve: A Network Server for Solving Computational Science Problems, Inernational Journal of Supercomputing Applications and High Performance Computing, Vol. 11, No. 3, 1997.

[77]Asim YarKhan, Jack Dongarra, and Keith Seymour, NetSolve to GridSolve: The Evolution of a Network Enabled Solver IFIP WoCo9 conference "Grid-Based Problem Solving Environments: Implications for Development and Deployment of Numerical Software", Prescott, AZ, July 17-21, 2006.

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

[79]Caron E, Desprez F, Loureiro D. All-in-one Graphical Tool for the management of DIET a GridRPC Middleware, In CoreGRID Workshop on Grid Middleware (in conjunction with OGF'23), 2008.

[80]Y. Tanaka, H. Nakada, S. Sekiguchi, T. Suzumura, and S. Matsuoka, “Ninf-G: A reference implementation of RPC-based programming middleware for Grid computing,” Journal of Grid Computing, vol. 1, no. 1, pp. 41–51, 2003.

[81]Nakada H, Sato M, Sekiguchi S. Design and implementation of Ninf: Towards a global computing infrastructure. Future Generation Computing Systems (Metacomputing Special Issue) October 1999; 15(5–6):649–658.

[82]Tanaka, Y., Takemiya, H., Nakada, H., and Sekiguchi, S. 2004. Design, Implementation and Performance Evaluation of GridRPC Programming Middleware for a Large-Scale Computational Grid. In Proceedings of the 5th IEEE/ACM Grid 2004, 298-305.

[83]T. Brady, M. Guidolin, and A. Lastovetsky. Experiments with SmartGridSolve: Achieving Higher Performance by Improving the GridRPC Model. Grid 2008, Tsukuba, Japan, 29 September - 01 October 2008. IEEE Computer Society.

[84]M. Guidolin and A. Lastovetsky. ADL: An Algorithm Definition Language for SmartGridSolve. In Proceedings of the 9th IEEE/ACM International Conference on Grid Computing (Grid 2008), Tsukuba, Japan, 29 September - 01 October 2008. IEEE Computer Society.

[85]M. D. Beynon, R. Ferreira, T. Kurc, A. Sussman, and J. Saltz. DataCutter: Middleware for filtering very large scientific datasets on archival storage systems. In Proceedings of the Eighth Goddard Conference on Mass Storage Systems and Technologies/17th IEEE Symposium on Mass Storage Systems, pages 119{133. National Aeronautics and Space Administration, Mar. 2000.

[86]M. D. Beynon, T. Kurc, A. Sussman, and J. Saltz. Optimizing execution of component-based applications using group instances. In Proceedings of CCGrid2001: IEEE International Symposium on Cluster Computing and the Grid, pages 56{63. IEEE Computer Society Press,May 2001.

[87]Gilles Fedak, Haiwu He, Franck Cappello: BitDew: A data management and distribution service with multi-protocol file transfer and metadata abstraction. J. Network and Computer Applications 32(5): 961-975 (2009)

[88]Fedak, G., He, H., and Cappello, F. 2008. BitDew: a programmable environment for large-scale data management and distribution. In Proceedings of the 2008 ACM/IEEE Conference on Supercomputing (Austin, Texas, November 15 - 21, 2008). Conference on High Performance Networking and Computing. IEEE Press, Piscataway, NJ, 1-12.

[89]Thain, D., Tannenbaum, T., Livny, M.: Distributed computing in practice: the Condor experience. Concurrency and Computation: Practice and Experience 17, 323–356 (2005)

[90]Neubauer, F., Hoheisel, A., and Geiler, J. 2006. Workflow-based grid applications. Future Gener. Comput. Syst. 22, 1 (Jan. 2006), 6-15.

[91]Qin, J. and Fahringer, T. 2007. Advanced data flow support for scientific grid workflow applications. In Proceedings of the 2007 ACM/IEEE Conference on Supercomputing (Reno, Nevada, November 10 - 16, 2007). SC '07. ACM, New York, NY, 1-12.

[92] Ludäscher, B., Altintas, I., Berkley, C., Higgins, D., Jaeger, E., Jones, M., Lee, E. A., Tao, J., and Zhao, Y. 2006. Scientific workflow management and the Kepler system: Research Articles. Concurr. Comput. : Pract. Exper. 18, 10 (Aug. 2006), 1039-1065.

[93]I. Altintas, O. Barney, and E. Jaeger-Frank, “Provenance Collection Support in the Kepler Scientific Workflow System,” Proc. Int’l Provenance and Annotation Workshop (IPAW ’06), pp. 118-132, 2006.

[94]George Bosilca , Aurelien Bouteiller , Franck Cappello , Samir Djilali , Gilles Fedak , Cecile Germain , Thomas Herault , Pierre Lemarinier , Oleg Lodygensky , Frederic Magniette , Vincent Neri , Anton Selikhov, MPICH-V: toward a scalable fault tolerant MPI for volatile nodes, Proceedings of the 2002 ACM/IEEE conference on Supercomputing, p.1-18, November 16, 2002, Baltimore, Maryland

[95]Bouteiller, A., Cappello, F., Herault, T., Krawezik, G., Lemarinier, P., and Magniette, F. 2003. MPICH-V2: a Fault Tolerant MPI for Volatile Nodes based on Pessimistic Sender Based Message Logging. In Proceedings of the 2003 ACM/IEEE Conference on Supercomputing (November 15 - 21, 2003). Conference on High Performance Networking and Computing. IEEE Computer Society, Washington, DC, 25.

[96]Djilali, S., Herault, T., Lodygensky, O., Morlier, T., Fedak, G., and Cappello, F. 2004. RPC-V: Toward Fault-Tolerant RPC for Internet Connected Desktop Grids with Volatile Nodes. In Proceedings of the 2004 ACM/IEEE Conference on Supercomputing (November 06 - 12, 2004). Conference on High Performance Networking and Computing. IEEE Computer Society, Washington, DC, 39.

[97]Djilali, S. 2003. P2P-RPC: Programming Scientific Applications on Peer-to-Peer Systems with Remote Procedure Call. In Proceedings of the 3st international Symposium on Cluster Computing and the Grid (May 12 - 15, 2003). CCGRID. IEEE Computer Society, Washington, DC, 406.

[98]D. P. Anderson, "BOINC: A System for Public-Resource Computing and Storage," The Fifth IEEE/ACM International Workshop on Grid Computing (GRID'04), IEEE CS Press, pp. 4-10, Nov. 2004.

[99]M. Taufer, D. Anderson, P. Cicotti, and C. L. Brooks III, "Homogeneous Redundancy: a Technique to Ensure Integrity of Molecular Simulation Results Using Public Computing," The 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05), Heterogeneous Computing Workshop (HCW'05), pp. 119a, Apr. 2005.

[100]A. Chien, B. Calder, S. Elbert, and K. Bhatia, "Entropia: architecture and performance of an enterprise desktop grid system," Journal of Parallel and Distributed Computing, vol. 63, issue 5, pp. 597-610, May 2003.

[101]A. A. Chien, S. Marlin, and S. T. Elbert, "Resource management in the Entropia System," Chapter 26 in Grid Resource Management: Sate of the Art and Future Trends, Kluwer Academic, 2003.

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

[103]James Frey, Todd Tannenbaum, Ian Foster, Miron Livny, and Steven Tuecke, “Condor-G: A Computation Management Agent for Multi-Institutional Grids”, Journal of Cluster Computing volume 5, pages 237-246, 2002

[104]LITZKOW, M. L., MUTKA M. W., 1988, Condor - A Hunter of Idle Workstations, Proc. of the 8th International Conference of Distributed Computing Systems (ICDCS1988).

[105]B. O. Christiansen, P. Cappello, M. F. Ionescu, M. O. Neary, K. E. Schauser, and D. Wu. Javelin: Internet-Based Parallel Computing Using Java. Concurrency: Practice and Experience, 9(11):1139–1160, Nov. 1997.

[106]M. O. Neary, B. O. Christiansen, P. Cappello, and K. Schauser, "Javelin: Parallel computing on the internet," Future Generation Computer Systems, Special Issue on Metacomputing, vol. 15, issue 5-6, pp. 659-674, Oct. 1999.

[107]M. O. Neary, S. P. Brydon, P. Kmiec, S. Rollins, and P. Cappello, "Javelin++: Scalability Issues in Global Computing," Concurrency: Parctice and Experience, vol. 12, issue 8, pp. 727-753, Aug. 2000.

[108]M. Neary, A. Phipps, S. Richman, P. Cappello, Javelin 2.0: Java-Based Parallel Computing on the Internet, Proceedings of European Parallel Computing Conference (Euro-Par 2000), Germany, 2000.

[109] Morrison, J. P., Kennedy, J. J., and Power, D. A. 2001. WebCom: A Web Based Volunteer Computer. J. Supercomput. 18, 1 (Jan. 2001), 47-61.

[110]Morrison, J.P., Clayton, B, & Patil A. (2002). Comparison of WebCom in the context of Job Management Systems. International Symposium of Parallel and Distributed Computing (ISPDC 2002), Iasi, Romania, July 17-20, 2002.

[111]Morrison, J. P., Clayton, B., Power, D. A., and Patil, A. 2004. Webcom-G: grid enabled metacomputing. Neural, Parallel Sci. Comput. 12, 3 (Sep. 2004), 419-438.

[112]David A. Power, Adarsh Patil, Sunil John, and John P. Morrison. WebCom-G. Proceedings of the international conference on parallel and distributed processing techniques and applications (PDPTA 2003), Las Vegas, Nevada, June 23–26, 2003.

[113]Raphael Bolze, Franck Cappello, Eddy Caron, Michel J. Daydé, Frédéric Desprez, Emmanuel Jeannot, Yvon Jégou, Stéphane Lanteri, Julien Leduc, Nouredine Melab, Guillaume Mornet, Raymond Namyst, Pascale Primet, Benjamin Quétier, Olivier Richard, El-Ghazali Talbi, Iréa Touche: Grid'5000: A Large Scale And Highly Reconfigurable Experimental Grid Testbed. IJHPCA 20(4): 481-494 (2006)

[114] Franck Cappello, Eddy Caron, Michel J. Daydé, Frédéric Desprez, Yvon Jégou, Pascale Vicat-Blanc Primet, Emmanuel Jeannot, Stéphane Lanteri, Julien Leduc, Nouredine Melab, Guillaume Mornet, Raymond Namyst, Benjamin Quétier, Olivier Richard: Grid'5000: a large scale and highly reconfigurable grid experimental testbed. GRID 2005: 99-106

[115]Cappello, F., Djilali, S., Fedak, G., Herault, T., Magniette, F., Néri, V., and Lodygensky, O. 2005. Computing on large-scale distributed systems: Xtrem Web architecture, programming models, security, tests and convergence with grid. Future Gener. Comput. Syst. 21, 3 (Mar. 2005), 417-437.

[116]Gilles Fedak , Cecile Germain , Vincent Neri , Franck Cappello, XtremWeb: A Generic Global Computing System, Proceedings of the 1st International Symposium on Cluster Computing and the Grid, p.582, May 15-18, 2001

[117]O. Delannoy, N. Emad, and S. G. Petiton.Workflow Global Computing with YML. In The 7th IEEE/ACM International Conference on Grid Computing, pages 25-32, 2006.

[118]O. Delannoy and S. Petiton. A Peer to Peer Computing Framework: Design and Performance Evaluation of YML. In Third International Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Networks, pages 362-369. IEEE Computer Society Press, 2004.

[119]O. Delannoy, YML: A scientific Workflow for High Performance Computing, Ph.D. Thesis, Septembre 2008, Versailles

[120]N. Melab, E.-G. Talbi, and S. G. Petiton. A parallel adaptive gauss jordan algorithm. The Journal of Supercomputing, 17(2):167-185, 2000.

[121]N. Melab, E. Talbi, and S. Petiton, A parallel adaptive version of the block-based gauss-jordan algorithm, in IPPS/SPDP, 1999, pp. 350-354.

[122]S. Petiton, Parallelization on an MIMD computer with realtime Scheduler, Aspects of Computation on Asynchronous Parallel Processors, North Holland, 1989.

[123]L. M. Aouad and S. G. Petiton. Parallel basic matrix algebra on the grid5000 large scale distributed platform. In CLUSTER, 2006.

[124] Parallel Programming Models and paradigms: http://www.buyya.com/ cluster/v2chap1.pdf

[125] Designing Parallel Algorithms : http://wwwunix.mcs.anl.gov/dbpp/text/book.html

[126]Maxime Hugues and Serge G. Petiton: A Matrix Inversion Method with YML/OmniRPC on a Large Scale Platform. VECPAR’2008, Page: 95-108 Jun 24-27, 2008, Toulouse, France

[127]L. M. Aouad, S. Petiton, and M. Sato, Grid and Cluster Matrix Computation with Persistent Storage and Out-of-Core Programming, in The 2005 IEEE International Conference on Cluster Computing, September 2005. Boston, Massachusetts, 2005.

[128]Vaquero, L. M., Rodero-Merino, L., Caceres, J., and Lindner, M. 2008. A break in the clouds: towards a cloud definition. SIGCOMM Comput. Commun. Rev. 39, 1 (Dec. 2008), 50-55.

[129]T. Oreilly, "What is Web 2.0: Design patterns and business models for the next generation of software," O'Reilly Media, Tech. Rep., 2008. [Online]. Available: http://www.oreillynet.com/pub/a/oreilly/tim/new s/2005/ 09/ 30/ what-is-web-20.html

[130]S. Murugesan, UnderstandingWeb 2.0. IEEE IT Professional 9(4):34-41, Jul. 2007.

[131]I. Foster, Y. Zhao, I. Raicu, and S. Lu, "Cloud Computing and Grid Computing 360-degree compared," in Grid Computing Environments Workshop, 2008, pp. 1–10.

[132]Buyya, R., Yeo, C. S., and Venugopal, S. 2008. Market-Oriented Cloud Computing: Vision, Hype, and Reality for Delivering IT Services as Computing Utilities. In Proceedings of the 2008 10th IEEE international Conference on High Performance Computing and Communications (September 25 - 27, 2008). HPCC. IEEE Computer Society, Washington, DC, 5-13.

[133]Mc Evoy, G. V. and Schulze, B. 2008. Using clouds to address grid limitations. In Proceedings of the 6th international Workshop on Middleware For Grid Computing (Leuven, Belgium, December 01 - 05, 2008). MGC '08. ACM, New York, NY, 1-6.

[134]Weiss, A. 2007. Computing in the clouds. netWorker 11, 4 (Dec. 2007), 16-25.

[135]Kemal A. Delic , Martin Anthony Walker, Emergence of the academic computing clouds, Ubiquity, v.9 n.31, p.1-1, August 5-11, 2008

[136]P. T. Jaeger, J. Lin, and J. M. Grimes. Cloud computing and information policy: Computing in a policy cloud? Journal of Information Technology & Politics., 5(3):269{283, 2008.

[137]Jha, S., Merzky, A., and Fox, G. 2009. Using clouds to provide grids with higher levels of abstraction and explicit support for usage modes. Concurr. Comput. : Pract. Exper. 21, 8 (Jun. 2009), 1087-1108.

[138]Brown, Michael. White paper: Cloud computing. Maximum PC, January 12, 2009. Retrieved March 10, 2009, from maximumpc.com/print/5047.

[139] Cloud Computing: An Overview. Queue 7, 5 (Jun. 2009), 3-4.

[140]Roy Bragg. Cloud computing: When computers really rule. Tech News World, July 2008. Electronic Magazine, available at http://www.technewsworld.com/story/63954.html.

[141]Jeremy Geelan. Twenty one experts define cloud computing. Virtualization, August 2008. Electronic Magazine, article available at http://virtualization.sys-con.com/node/612375.

[142]Ghemawat, S., Gobioff, H., and Leung, S. 2003. The Google file system. In Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles (Bolton Landing, NY, USA, October 19 - 22, 2003). SOSP '03. ACM, New York, NY, 29-43.

[143]Dean, J. and Ghemawat, S. 2004. MapReduce: simplified data processing on large clusters. In Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation - Volume 6 (San Francisco, CA, December 06 - 08, 2004). Operating Systems Design and Implementation. USENIX Association, Berkeley, CA, 10-10.

[144]Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A., Burrows, M., Chandra, T., Fikes, A., and Gruber, R. E. 2006. Bigtable: a distributed storage system for structured data. In Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation - Volume 7 (Seattle, WA, November 06 - 08, 2006). Operating Systems Design and Implementation. USENIX Association, Berkeley, CA, 15-15.

[145]James Staten. IBM's Play In Cloud Computing? Listening Carefully, July 2008.
http://blogs.forrester.com/it_infrastructure/2008/07/ibms-play-in-cl.html

[146]Nurmi, D., Wolski, R., Grzegorczyk, C., Obertelli, G., Soman, S., Youseff, L., and Zagorodnov, D. 2009. The Eucalyptus Open-Source Cloud-Computing System. In Proceedings of the 2009 9th IEEE/ACM international Symposium on Cluster Computing and the Grid (May 18 - 21, 2009). CCGRID. IEEE Computer Society, Washington, DC, 124-131.

[147]Mc Evoy, G. V. and Schulze, B. 2008. Using clouds to address grid limitations. In Proceedings of the 6th international Workshop on Middleware For Grid Computing (Leuven, Belgium, December 01 - 05, 2008). MGC '08. ACM, New York, NY, 1-6.

[148]C. Mateos, A. Zunino, and M. Campo, "A survey on approaches to gridification," Software—Practice & Experience, 38(5), April 2008, pp. 523-556.

[149] Ostermann, S., Iosup, A., Yigitbasi, N., Prodan, R., Fahringer, T., Epema, D.: An early performance analysis of cloud computing services for scientific computing. Technical Report PDS-2008-006, Delft University of Technology (December 2008)

[150]M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A.Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica,M. Zaharia. Above the Clouds: A Berkeley View of Cloud computing. Technical Report, University of California at Berkley, USA, Feb. 10, 2009.

[151]S. L. Garfinkel. An Evaluation of Amazon's Grid Computing Services: EC2, S3 and SQS. Technical Report TR-08-07, Harvard University, August 2007.

[152]de Assuncao, M. D., di Costanzo, A., and Buyya, R. 2009. Evaluating the cost-benefit of using cloud computing to extend the capacity of clusters. HPDC '09. ACM, 141-150.

[153]Shadi Ibrahim, Hai Jin, Lu Lu, Li Qi, Song Wu, Xuanhua Shi: Evaluating MapReduce on Virtual Machines: The Hadoop Case. CloudCom 2009: 519-528

[154]David P. Anderson, Gilles Fedak: The Computational and Storage Potential of Volunteer Computing. CCGRID 2006: 73-80

[155]Eric Martin Heien, David P. Anderson: Computing Low Latency Batches with Unreliable Workers in Volunteer Computing Environments. J. Grid Comput. 7(4): 501-518 (2009)

[156]Bahman Javadi, Derrick Kondo, JeanMarc Vincent, David P. Anderson, "Mining for Statistical Models of Availability in Large Scale Distributed Systems: An Empirical Study of SETI@home", 17th IEEE/ACM MASCOTS 2009, London, UK, September, 2009

[157]X. Ma, S.S. Vazhkudai, Z. Zhang, "Improving Data Availability for Better Access Performance: A Study on Caching Scientific Data on Distributed Desktop Workstations", Journal of Grid Computing Vol. 7, No. 4, pp. 419-438, December 2009

[158]Derrick Kondo, Bahman Javadi, Paul Malecot, Franck Cappello, David P. Anderson, "Cost-benefit analysis of Cloud Computing versus desktop grids," ipdps, pp.1-12, 2009

[159]Artur Andrzejak, Derrick Kondo, David P. Anderson: Exploiting Non-Dedicated Resources for Cloud Computing in the 12th IEEE/IFIP (NOMS 2010), Osaka, Japan April 19-23, 2010.

[160]Domingues, P., Araujo, F., and Silva, L. Evaluating the performance and intrusiveness of virtual machines for desktop grid computing. (May 23 - 29, 2009). IPDPS 2009. pp1-8.

[161]Vincenzo D. Cunsolo, Salvatore Distefano, Antonio Puliafito, Marco Scarpa: Cloud@Home: Bridging the Gap between Volunteer and Cloud Computing. ICIC (1) 2009: 423-432

[162]Eddy Caron, Frederic Desprez, David Loureiro, Adrian Muresan, "Cloud Computing Resource Management through a Grid Middleware: A Case Study with DIET and Eucalyptus," cloud, pp.151-154, 2009

[163]Lizhe Wang, Jie Tao, Marcel Kunze, Alvaro Canales Castellanos, David Kramer, Wolfgang Karl, "Scientific Cloud Computing: Early Definition and Experience," hpcc, pp.825-830, 2008 10th IEEE International Conference on HPCC 2008

[164]C. Vecchiola, S. Pandey, and R. Buyya, High-Performance Cloud Computing: A View of Scientific Applications. 10th International Symposium on Pervasive Systems, Algorithms and Networks (I-SPAN 2009), Kaohsiung, Taiwan, Dec. 2009.

[165]L. Shang, Z. Wang, X. Zhou, X. Huang, and Y. Cheng, "Tm-dg: a trust model based on computer users' daily behavior for desktop grid platform," in CompFrame '07: Proceedings of the 2007 symposium on

Component and framework technology in high-performance and scientific computing. New York, NY, USA: ACM, 2007, pp. 59–66.

[166]Shang, L., Wang, Z., and Petiton, S. G. 2008. Solution of Large Scale Matrix Inversion on Cluster and Grid. In Proceedings of the 2008 Seventh international Conference on Grid and Cooperative Computing (October 24 - 26, 2008). GCC 2008. 33-40.

[167]Ling Shang, Serge Petiton, Maxime Hugues, "A New Parallel Paradigm for Block-Based Gauss-Jordan Algorithm," gcc, pp.193-200, 2009 Eighth International Conference on Grid and Cooperative Computing, 2009

[168]Philippe Smets, The transferable belief model and other interpretations of Dempster-Shafer's model, Proceedings of the Sixth Annual Conference on Uncertainty in Artificial Intelligence, p.375-384, July 27-29, 1990

[169] Karthick Ramachandran: Decentralized Resource Availability for a Desktop Grid. Master Thesis. May 2009. University of Western Ontario, Canana.

[170]M. Sato, M. Hirono, Y. Tanaka, and S. Sekiguchi. OmniRPC: A Grid RPC Facility for Cluster and Global Computing in OpenMP. In WOMPAT, LNCS 2104, pages 130-136. Springer, 200

Résumé

Les grilles de calcul et les grille de PC sur Internet offrent des alternatives intéressantes pour le calcul scientifique à grande échelle, qui demande des ressources de calcul importantes. Toutefois, l'adaptation des applications pour ces systèmes est difficile à cause des facteurs nombreux tels que l'interface complexe de programmation. L'objectif de cette thèse est de trouver une solution pour faciliter le calcul scientifique à grande échelle. Pour ce faire, j'ai travaillé sur l'algorithme de Gauss Jordan et une nouvelle version d'un schéma de parallélisme. Ce schéma peut exploiter le maximum de parallélisme entre des opérations. Comme un exemple excellent, l'algorithme de Gauss Jordan est également utilisé pour évaluer des environnements expérimentaux et des outils différents. Les expérimentations avec YML, OmniRPC et XtremWeb sur les grilles et les grilles de PC montrent que YML peut être une bonne solution pour que les utilisateurs fassent du calcul scientifique à grande échelle, à cause des bonnes caractéristiques comme « l'interface d'abstraction de haut niveau», « les composants réutilisables » et «le surcoût acceptable». Pour obtenir les meilleures performances de cette plate-forme, les questions concernées, telles que la granularité des tâches, la persistance des données et le mécanisme d'ordonnancement, sont également abordés dans cette thèse. Selon les analyses faites ci-dessus et les caractéristiques communes des nuages informatiques ciblés, YML-PC, une architecture de référence basée sur les workflows pour les constructions de nuages informatiques privés scientifique est proposée. YML-PC hérite les bonnes caractéristiques présentées ci-dessus et des autres technologies clefs telles que « la persistance des données », « La prévision du temps disponible » et « l'évaluation sur des nœuds de calcul hétérogènes » pour YML-PC, qui sont également abordés dans cette thèse. Les évaluations sur l'algorithme de Gauss Jordan sont réalisées sur les grilles, les grilles de PC et les nuages informatiques privés qui sont implantés sur la plate-forme Grid5000, la plateforme de calcul de Polytech Lille en France et la plateforme de calcul de Hohai, en Chine.

Mots clés: le calcul scientifique à grande échelle, l'algorithme de Gauss Jordan, grilles, grilles de PC, Nuages informatiques privé, YML, OmniRPC, XtremWeb

Abstract

Grid computing and Desktop Grid computing provide interesting alternatives for large scale scientific computing which needs very large scale computing resources. However gridification is hard to develop because of series of factors such as complex programming interface. The aim of this dissertation is to find a solution to make large scientific computing in an easy way. To do that, research on Gauss Jordan algorithm is made and a new parallel programming adapted version is presented. The adapted parallel version can achieve maximum degree parallelism between operations. Also the Gauss Jordan algorithm as an excellent example is used to evaluate different experimental environments and tools. Experiments with YML, OmniRPC and XtremWeb on Grid and Desktop Grid environments testify YML can be a good solution for end users to make large scale scientific computing for its series of good features such as higher level interface, component reuse and acceptable overhead. To get better performance of platform, related issues such as task granularity, data persistence and schedule mechanism are also discussed in this dissertation. According to analysis made above and the common features of Clouds possessed, YML-PC a reference architecture based on workflow for building scientific Private Clouds is proposed. YML-PC inherits those good features presented above and some other key technologies such as "data persistence", "available time prediction" and "evaluation on heterogeneous computing nodes" for YML-PC are also discussed in this dissertation. Evaluations are made based on Gauss Jordan algorithm on Grids, Desktop Grids and Private Clouds which build on Grid5000, Polytech Lille platform, France and Hohai platform, China.

Key words: Large scale scientific computing, Gauss Jordan algorithm, Grids, Desktop Grids, Private Clouds, YML, OmniRPC, XtremWeb