私立東海大學

資訊工程與科學研究所

碩士論文

指導教授：楊朝棟　博士

G-BLAST: 一個針對 mpiBLAST 軟體在網格計算上的網格化解決方案

G-BLAST: a Grid-Based Solution for mpiBLAST on Computational Grids

研 究 生：韓祖棻

中華民國九十六年六月

摘要

近幾年中，由於在生物資訊領域的研究與發展與日俱增，以至於藉由大量運算來求得更好的效能的需求亦不斷地持續成長，基因序列比對就是一個明顯的例子。基於叢集技術可以減少執行時間和增進基因序列比對的效率，所以此類的需求通常使用平行計算的技術來解決。例如，mpiBLAST 就是一個結合NCBI BLAST軟體和平行計算訊息交換介面標準所實作出來的平行版BLAST軟體。然而，大部分的實驗室都沒有足夠的能力來建造一個具有強大計算能力的叢集環境，因此他們往往都是用數十台甚至是上百台個人電腦來建構出一個看似計算能力不錯的叢集環境。而且叢集環境通常受限於本地端環境，所以此一限制將嚴重阻礙計算能力的擴充性，不過這些缺點和限制可以由格網架構的概念來解決。

在格網環境中，散佈各地之虛擬組織的資源可以透過格網的概念來調派和集中，因此可以滿足各種在生物資訊應用方面的計算需求。在這篇論文中，我們將開發一個名為G-BLAST的生物格網架構。目前，G-BLAST是針對在格網環境上執行序列比對的工作而設計，而其最終則是透過一台伺服器來連接應用在各叢集節點上的mpiBLAST來完成序列比對的工作。此外，G-BLAST 還具有選擇最適合的工作節點、根據不同節點的效能來動態切割基因資料庫以及根據以往所執行的歷史紀錄來動態調整每個所屬於G-BLAST底下之工作節點的效能值等能力。為了加強G-BLAST的使用性，我們開發一個格網服務的入口和一個網格服務的圖型使用者介面應用軟體；使用者可以透過此介面來遞交工作、觀察工作的狀態以及接收比對結果，而系統管理者則可以透過此介面來管理他們自己的工作節點和觀察工作節點的使用況。此外，我們的介面和G-BLAST是透過WSRF的標準來溝通。

關鍵字：格網計算, Globus Toolkit, WSRF, 生物格網, 叢集計算, MPICH-G2, mpiBLAST, BLAST

Abstract

The research and development of bioinformatics (e.g., genomic sequence alignment) has been growing with each passing day in the past few years so that continue demands on large computing powers are required to support better performance. This trend requires usually solved by parallel computing techniques, because Cluster technology can reduce the execution time and increase genomic sequence alignment efficiency. For example, mpiBLAST is a parallel version of NCBI BLAST that combines the NCBI BLAST with Message Passing Interface standards. However, most laboratories can not build up powerful computing environments. They usually connect dozens or even hundreds of personal computers to build weak computing environments. Besides, Cluster usually is limited by a local computing environment that hinders the computing extendibility significantly. The concepts of the Grid framework are designated to overcome the aforementioned problems. Grid environments coordinate the resources of distributed virtual organizations and satisfy various computational demands for bioinformatics applications. In this thesis, we have deployed a BioGrid framework named G-BLAST. Currently, G-BLAST is designed for genomic sequence alignment by using the Grid environment and accessible mpiBLAST application, which is designed for Cluster environment, from a server node. G-BLAST is endowed with selection the most adaptive work nodes, dynamic fragmenting genomic database, and self-adjust performance data abilities. To enhance the capability and usability of G-BLAST, we also deployed a Grid Service Portal and a Grid Service GUI desk application for general users to submit jobs and for host administrators to maintain their own work nodes.

Keywords: Grid computing, Globus Toolkit, WSRF, BioGrid, Cluster Computing, MPICH-G2, mpiBLAST, BLAST

Acknowledgements

I would like to express my gratitude to all the people who have made writing this thesis a more pleasant task. Particular thanks to Professor Chao-Tung Yang, principal advisor of mine, his encouragement, guidance, and support were invaluable in my work. No matter the capability in the open discussions, or the preciseness in thesis writing, Professor Yang gave me a deep influence and inspiration. I would also like to thank Professor Sian-Syong Zeng, Professor Fang-Rong Syu, Professor Siao-Si Wang and Professor Sian-Jheng Yu for their valuable comments and advice given while serving on my reading committee.

I am also grateful to many other classmates and members of my lab at THU. They gave me opportunities to gain more knowledge and shared their knowledge with me. My research would not have been completed without the help from them.

Last, I must be grateful beyond place to my parents, who at an early age instilled a love of learning and through the years gave me all of their support and encouragement to pursue it.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

Bioinformatics combines biology and information technology and includes computational tools and methods for managing, analyzing, and manipulating large biology datasets. Computing technologies are vital for bioinformatics applications [1], [2], [3]. For example, biology problems often require repeating a task millions of times, such as when searching for sequence similarities in existing databases or comparing groups of sequences to determine evolutionary relationships. In such cases, high-performance computers to process this information are indispensable. Biological information is stored on many computers around the world. The easiest way to process this data is to join these computers together through network. Such activities require high-performance computing infrastructures [4], [5] with access to huge information databases [6]. The major advances in computer technology and computer science over the past 30 years have dramatically changed much of our society.

Comparing a sequence against a database is one of the most common bioinformatics applications. A sequence alignment is needed before making comparative statements about nucleic acid or protein sequences. The concept of selecting an optimal sequence alignment is simple, but is not at all simple in practice. Choosing a good alignment artificially is possible, but it must do more than once or twice. An automatic method for finding the optimal alignment out of the thousands of alternatives is the right approach. Currently, many

bioinformatics applications can be used to conduct computing tasks on Linux PC cluster or Grid systems [7], [8], [9], [10], [11], [12].

NCBI BLAST [13] is one of the best tools available to bioinformatics for using heuristic similarity-search algorithms to find sequences in genome databases similar to given query sequences. mpiBLAST [14] is a parallelization of BLAST that executes BLAST jobs in parallel using Message Passing Interface (MPI) [15].

A Cluster is a form of parallel computer that uses more than one processor. The many kinds of parallel computer are distinguished by the processors they use and the way in which those processors exchange data. They take advantage of two commodity components: fast CPUs designed primarily for the personal computer market and techniques for connecting personal computers in so-called local-area networks or LANs. Beowulf clusters provide effective and low-cost means of delivering enormous computational powers to applications and are now used virtually everywhere. However, most laboratories can not build up powerful environments. They usually connect dozen of personal computers to build weak computing environments.

The concepts of Grid [16], [17] can solve these problems. Grid environments coordinate the resources of distributed virtual organizations and satisfy a great many computational demands. Grid enable virtual organizations to share geographically distributed resources in their pursuit of common goals, assuming the existence of a central location, central control, omniscience, and an existing trust relationship. It can coordinate the resources of distributed virtual organizations and satisfy a great quality of computational demands. Owing to a deluge of data and information, fields such as high-energy physics, bioinformatics, and digital archive, demand a great deal of computational and storage capacity.

The Globus Project [18] provides some software tools, named Globus Toolkit, that make it easier to build computational Grid systems and applications. These tools are collectively called the Globus Toolkit. The toolkit includes software for security, information infrastructure, resource management, data management, communication, fault detection, and portability.

The BioGrid [2], [19], [20], [21], [22] is a Grid environment aimed at improving the performance of bioinformatics applications. In the area of sequence alignment applications, the mpiBLAST is generally widely used because it has well computationally efficient. However, it has three disadvantages. First, you must perform the *mpiformatdb* program to reorder database if your computing environment had been changed, or if you want to perform mpiBLAST on different number of processors on your Cluster. Second, mpiBLAST may take longer execution time to complete job when using more processors, and these situations are shown in experimental results section. Third, the performance is not guaranteed if your experiment environmental crosses not on LAN but on Internet.

1.2 The Goal and Contributions

To solve this problem, we have designed and implemented a BioGrid Framework named G-BLAST Framework, a Grid-based solution for mpiBLAST that integrates mpiBLAST into a Grid environment. G-BLAST was designed for mpiBLAST to use Grid environment and access Cluster systems from a server node, and mpiBLAST application had installed on each Cluster system. We use Globus Toolkit 4.0 [18] to create Grid systems and build whole Grid environment. Because Grid environment is a heterogeneous environment, G-BLAST is endowed with selection the most adaptive work nodes, dynamic fragmenting database, and self-adjust performance abilities.

We also implemented a Grid Service Portal named G-BLAST Service and a Grid Service GUI desk application for normal users to submit jobs and host administrators to maintain their own work nodes. The interaction between this application and G-BLAST Service meets WSRF [23] standards. The purpose of this study is to get higher performance than traditional Cluster architectures and to have an easier user interface than previous mpiBLAST versions. This present approach focuses on how to integrate mpiBLAST into a Grid environment and build a hierarchical architecture that integrates a Grid environment and some Cluster environments. The GUI implemented is used to perform users' invocations. The functions of G-BLAST Framework are measuring system loading, checking disc using status, analysis genomic database and query file, selection the most adaptive Cluster systems for execution job, record job information, split job to several sub-jobs, monitoring sub-jobs statuses, and combination results of sub-jobs.

1.3 Thesis and Organization

The rest of this thesis is organized as follows. In chapter 2, we introduce the background and related works. Development and details of G-BLAST are presented in chapter 3. In chapter 4, we provide information about our experiment environment. The introduction of our PC Cluster and Grid test-bed and experimental results are presented and discussed in chapter 5. Conclusions are given in chapter 6, with emphasis on potential areas for future work.

Chapter 2

Background Review

2.1 Grid Computing

The main concept of Grid Computing [17], [24] is to extend the original ideas of the Internet to sharing widespread computing power, storage capacities, and other resources. Grid Computing can coordinate the resources of distributed virtual organizations and satisfy a great quality of computational demands. Besides integrating distributed resources, Grid Computing can also reduce idle time of servers via management of integrated computing resources. Owing to a deluge of data and information, fields such as high-energy physics, bioinformatics, and digital archive, demand a great deal of computational and storage capacity.

The development of Grid Computing makes on-demand allocation and management of integrated computing resources possible. Some features of Grid Computing are listed below.

- Flexible, secure, coordinated resource-sharing among dynamic collections of individuals, institutions, and resources
- Transparent, secure, and coordinated resource-sharing and collaboration across sites
- The ability to form virtual collaborative organizations that share applications and data in an open heterogeneous server environment in order to work on common problems
- The ability to aggregate large amounts of geographically dispersed computing resources to tackle large problems and workloads as if all the servers and resources are located at a single site

- A hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to computational resources
- The Web provides us information—the Grid enables us to process it.

2.2 Grid Middleware

The Globus Project provides software tools that make it easier to build computational Grid systems and applications. These tools are collectively called the Globus Toolkit. The toolkit includes software for security, information infrastructure, resource management, data management, communication, fault detection, and portability. We used it as the infrastructure of our BioGrid.

The composition of the Globus Toolkit can be pictured as three pillars: Resource Management, Information Services, and Data Management. Each pillar represents a primary component of the Globus Toolkit and makes use of a common foundation of security. The Globus Resource Allocation Manager (GRAM) implements a resource management protocol, the Metacomputing Directory Service (MDS) implements an information services protocol, and GridFTP implements a data transfer protocol. They all use the GSI security protocol at the connection layer.

GRAM provides an API for using remote system resources, by providing a uniform, local job scheduling system. The specifications are written in the Resource Specification Language (RSL), and processed by GRAM as part of each job request.

The Monitoring and Discovery Service (MDS) is the information services component of the Globus Toolkit and provides information about available resources on the Grid and their statuses. It gives current information that may include computer, network, and other machines

properties in your Grid, such as number of processors available, CPU loading, network status, file system information, bandwidth, storage devices, and memory.

The Globus alliance proposed a common data transfer and access protocol called GridFTP. It is a high-performance, secure, efficient data movement, and reliable data transfer protocol optimized for Grid environments. The GridFTP protocol extends the standard FTP protocol, the highly-popular Internet file transfer protocol, and provides a superset of the features offered by the various Grid storage systems currently in use.

2.3 WSRF

The Web Service Resource Framework (WSRF) construct was proposed to express the relationship between stateful resources and Web Service specifications that define a rendering of the WS-Resource approach in terms of specific message exchanges and related XML definitions. These specifications allow program-mers to declare and implement associations between Web services and stateful resources. They describe the means by which a view of the state of a resource is defined and associated with a Web Services description to form an overall type definition of a WS-Resource. They also describe how the state of a WS-Resource is made accessible through a Web Service interface, and define related mechanisms concerned with WS-Resource grouping and addressing. The WSRF has five separate specification documents that provide the normative definition of the framework: WS-ResourceProperties, WS-ResourceLifetime, WS-RenewableReferences, WS-ServiceGroup, and WS-BaseFaults. In this paper, the WSRF is standardized by OASIS [28] and implemented by Globus toolkit 4.

In addition, the Simple Object Access Protocol (SOAP) is a lightweight XML-based messaging protocol used to encode the information in Web Service requests and response

messages before sending them over a network. SOAP messages are independent of any operating system or protocol and may be transported using a variety of Internet protocols.

2.4 Cluster Computing

A Cluster is a form of parallel computer that uses more than one processor. The many kinds of parallel computer are distinguished by the processors they use and the way in which those processors exchange data. They take advantage of two commodity components: fast CPUs designed primarily for the personal computer market and techniques for connecting personal computers in so-called local-area networks or LANs. Beowulf clusters provide effective and low-cost means of delivering enormous computational powers to applications and are now used virtually everywhere.

To make use of multiple processes each executed on a separate processor, we need to apply parallelism computing algorithms. There are two common types of parallelism: MPI [15] and PVM [29].

- PVM: This is a master-worker approach and is the simplest and easiest to implement. It relies on being able to break computations into independent tasks. A master then coordinates completion of these independent tasks by worker processes.
- MPI: This is for use when computations cannot (or cannot easily) be broken into independent tasks. In this kind of parallelism, the computation is broken down into communicating, interdependent tasks. We used LAM/MPI for our cluster system. LAM/MPI [15] is a high-quality open-source implementation of the Message Passing Interface specification, including all of MPI-1.2 and much of MPI-2. Intended for production as well as research use, LAM/MPI includes a rich set of features for system

administrators, parallel programmers, application users, and parallel computing researchers.

- MPICH-G2 [25] is a grid-enabled implementation of the MPI v1.1 standard. In addition, MPICH-G2 allows coupling of multiple machines, with different architectures, to run MPI applications. MPICH-G2 automatically converts data in messages sent between machines of different architectures and supports multiprotocol communication by automatically selecting TCP for intermachine messaging and (where available) vendor-supplied MPI for intramachine messaging.

2.5 mpiBLAST

Comparing a sequence against a database is one of the most common bioinformatics applications. A sequence alignment is needed before making comparative statements about nucleic acid or protein sequences. The concept of selecting an optimal sequence alignment is simple, but is not at all simple in practice. Choosing a good alignment artificially is possible, but it must do more than once or twice. An automatic method for finding the optimal alignment out of the thousands of alternatives is the right approach.

A common application of sequence alignment is searching a database for sequences similar to a query sequence. Hence, there are many sequence alignment tools, such as BLAST (Basic Local Alignment Search Tool, [5], [26]), based on various algorithms. There are vast volumes of DNA sequence data, and we need to figure out which parts of that DNA control the various chemical processes of life and determine the functions of new proteins from the known functions and structures of some proteins. Availability of computer resources is the key factor limiting use of bioinformatics analyses as a result of the growing computational demands. Various databases of gene/protein sequences, gene expression, and related analysis

tools help scientists determine whether and how a particular molecule is directly involved in a disease process. That, in turn, aids in the discovery of new and better drug targets [17], [21].

The BLAST is a sequence database search tool that seeks similarities between two substrings in molecular biology by using score matrices to improve filtration efficiency and to introduce more accurate rules for locating potential matches. BLAST attempts to find all locally maximal segment pairs in query sequences and database sequences with scores above some set threshold. mpiBLAST is a freely available, open-source parallelization of NCBI BLAST. It contains a pair of programs that replace formatdb and blastall with versions that execute BLAST jobs in parallel on clusters of computers with MPI installed.

There are two primary advantages to using mpiBLAST rather than conventional BLAST. First, mpiBLAST segments a target database, and then dispatches the segments to nodes in clusters. Because the database segment in each node is small, it can usually reside in the buffer-cache, yielding a significant speedup due to the elimination of disk I/O. Second, it allows BLAST users to take advantage of efficient, low-cost Beowulf clusters because interprocessor communication demands are low. Table 2-1 shows Appropriate Query/Program Combinations of BLAST.

Table 2-1 Appropriate Query/Program Combinations for “Compare 2 Sequences”

First Sequence	Second Sequence	Program
Nucleotide	Nucleotide	*blastn*
Nucleotide	Protein	*blastx*
Protein	Nucleotide	*tblastn*
Protein	Protein	*blastp*

The main executable programs in the BLAST distribution are:

- [blastall] performs BLAST searches using one of five BLAST programs: blastn, blastp, blastx, tblastn, or tblastx. Table 1 summarizes the query, database sequence, and alignment types for the various BLAST commands.

- [blastpgp] performs searches in PSI-BLAST or PHI-BLAST mode. blastpgp performs gapped blastp searches and can be used for iterative searches in psi-blast and phi-blast mode.
- [bl2seq] performs a local alignment of two sequences. bl2seq allows the comparison of two known sequences using blastp or blastn. Most of the bl2seq command-line options are similar to those for blastall.
- [formatdb] is used to format protein or nucleotide source databases. It converts a FASTA-format flat file sequence database into a BLAST database.

The mpiBLAST algorithm consists of two primary steps:

- Databases are segmented and put on a shared storage device.
- mpiBLAST queries are run on each node.

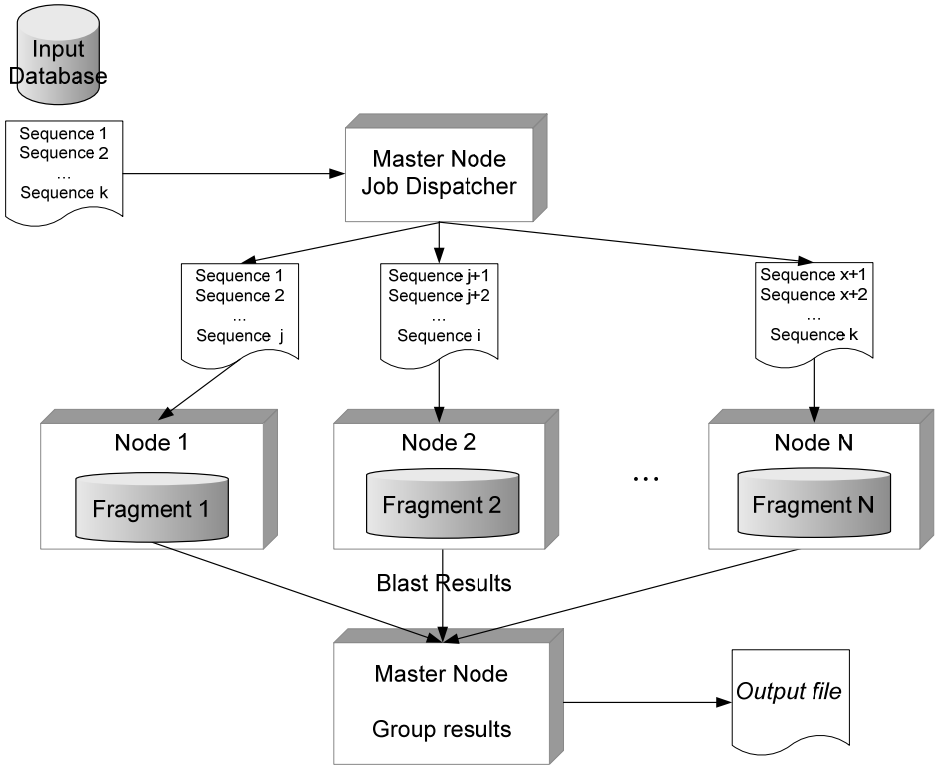


Figure 2-1 The mpiBLAST partitioning schema

The mpiBLAST Partitioning schema is shown in Figure 2-1. It uses multithreading to segment databases, assigning distinct portions of the database to each processor. It wraps the standard NCBI *formatdb* called mpiformatdb to format the Database. Command line arguments specify the number of fragments. *mpiformatdb* formulates command line arguments that force NCBI *formatdb* to format and divide the database into many fragments of approximately equal size.

When *mpiformatdb* execution is complete, the formatted fragments are placed in shared storage. Alignment of the database is accomplished by the local sequence alignment algorithm implemented in the NCBI [27] development library. If a node does not have fragments needed by a search, the fragments are copied from shared storage. Fragments are assigned to nodes using an algorithm that minimizes the number of fragments copied during each search.

Chapter 3

Design and Implementation of G-BLAST

We propose the G-BLAST Framework that its software architecture is shown in Figure 3-1 and its system architecture is shown in Figure 3-2. This framework can integrate many bioinformatics applications, such as mpiBLAST and FASTA, among others. Currently, this framework is just integration mpiBLAST 1.4.0. In G-BLAST Framework, it is in charge of dispatch an mpiBLAST job to Cluster Systems.

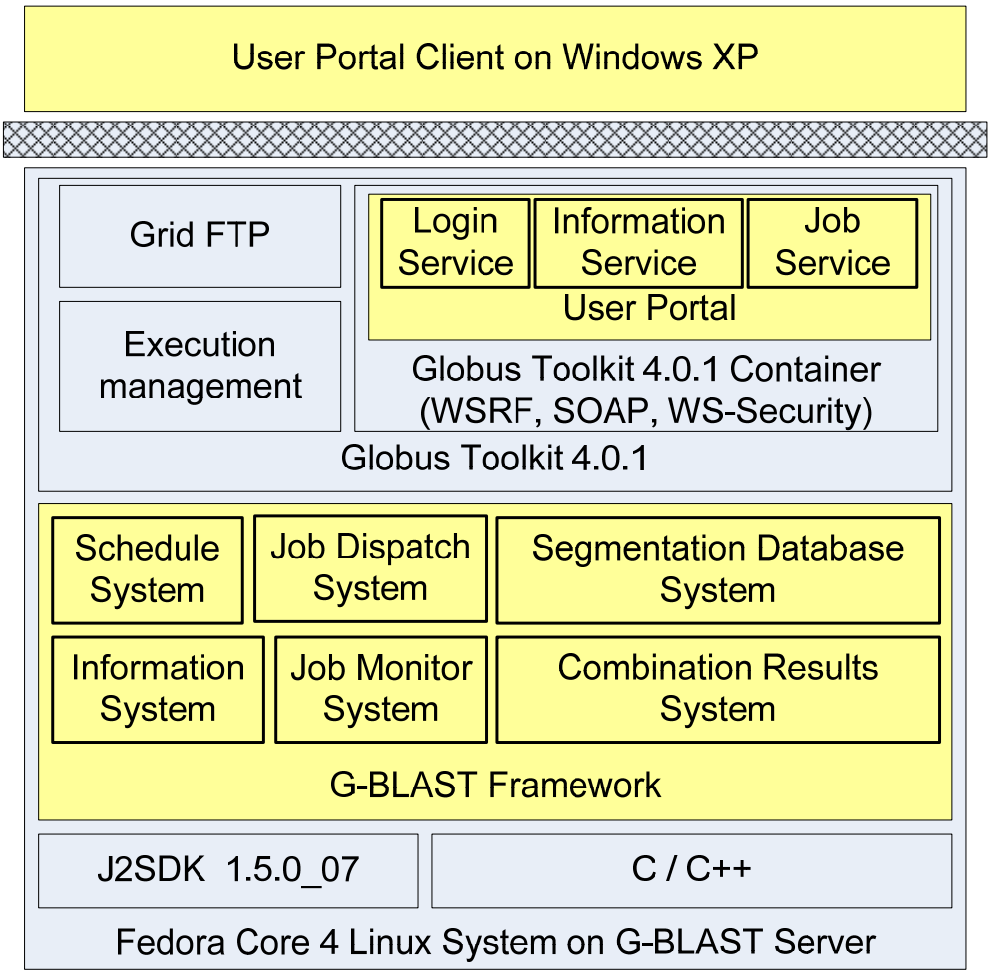


Figure 3-1 The G-BLAST Framework Software Architecture

Its works include measuring system loading, disc using status, analysis genomic database and query file, selection the most adaptive Cluster systems for execution job, record job information, split job to several sub-jobs, monitoring sub-jobs statuses, and combination results of sub-jobs. In addition, this framework is implemented by C/C++ language. Furthermore, the details are stored into Information Database while the job is performed. In this project, we use PostgreSQL Database System, version 8.1, to maintain and manage Information Database.

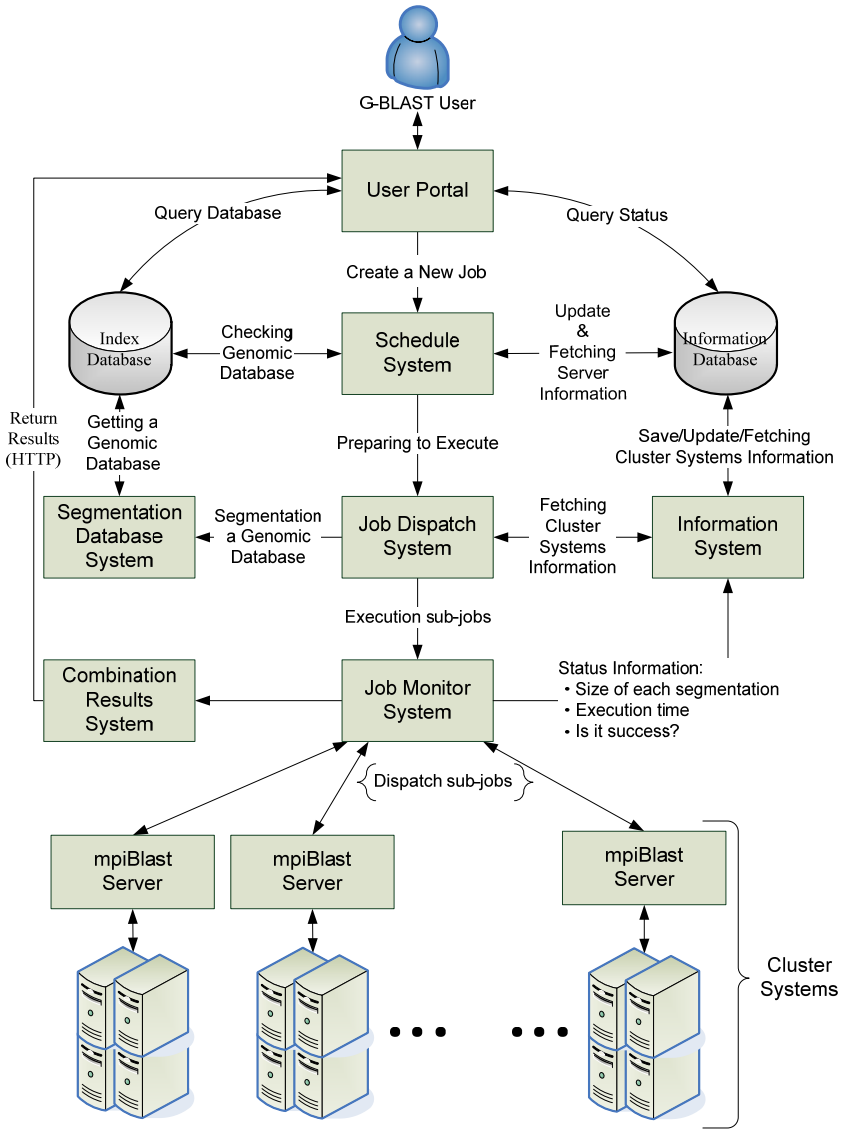


Figure 3-2 The G-BLAST System Architecture

Moreover, we design a command-line interface for system manager that functions are testing G-BLAST, managing Index Database, and managing Information Database. In management Index Database function, the genomic database is indexed in Index Database and generating a sorted genomic database when a genomic database registers in our environment through this function. If the index record had existed, the information of this genomic database will perform the Figure 3-4 procedure.

3.1 User Portal

The users have to use this framework via User Portal. The User Portal consisted Login Service, Information Service, Job Service, and a Client GUI application; these services and GUI application are implemented by JAVA language. In service portion, they are built on Globus toolkit 4 WSRF Container on G-BLAST server. In GUI application portion, the users use this application to register, login, submit job, check job status, and maintain their own information. Because this application is implemented by JAVA language, it is a cross-platform application. We selected WSRF standard to be G-BLAST framework's interface because WSRF can keep state for each job. In general, if an mpiBLAST user executes a job, she/he usually holds on the terminal screen until the job finish and connects server to check status and get result. But, in our interface, the Information Service can keep state. Therefore, users do not need to hold on the client application until the job finish after they send a new job to G-BLAST via Job Service; they can close the client application immediately. The state of job is changed into successful when the job finishes correctly, or changed into failed when the job executes incorrectly. If the state was successful, the G-BLAST returns the result, execution time, and more information to user. The user can login the client application via Login Service and see the details of each job via Information Service. Moreover, the users'

information and the hosts' information are maintained by Login Service. The User Portal's flowchart is shown in Figure 3-3.

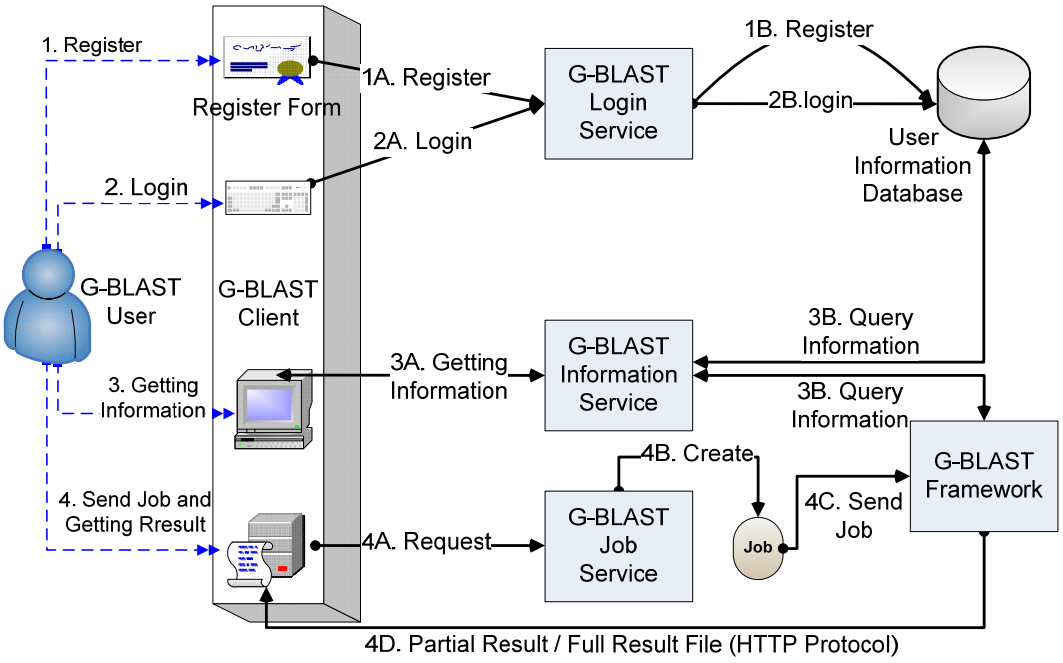


Figure 3-3 User Portal Flowchart

3.2 Schedule System

The Schedule System sorts and checks Genomic Databases in the server and monitors the job loading of the server. Since the sequence databases in this thesis are downloaded from NCBI and their sequences in each database are not stored in Size order, you must reorder the sequence every time when performing mpiformatdb command. "Size" refers to the letters or length of a sequence. Besides, the time complexity of reordering database is associated with number of sequence and length of each sequence. However, reordering database is wasting time and unnecessary. Therefore, this system sorts original database according to sequence length in descending order in advance and then divides them according to machine performance. Moreover, the process database can help Segmentation Database System to segment database more accurate. The databases will be provided for users after they are sorted.

The advantage of sorting database in advance is the database unnecessary reordering again when performing mpiformatdb. In addition, this system records the number of sequences, number of letters, size, and date of the genomic database into Information Database when the database is registered in G-BLAST, and records the same information after the database is sorted.

Database checking is necessary because the genomic database may be lost (e.g., hard discs was broken), even though the probability is extremely low. Hence, we must be sure this database existing, which is needed by the user. If the checking result is true, the information of physical file matches that of the Information Database, and then the user's invocation is accepted and the job will be delivered to the Job Dispatch System. If the result is false, this system will check for sure that the original database existing or not. If the database exists, it will be sorted again and then the job is delivered to the Job Dispatch System. If the database does not exist, this system will notify the user that the invocation was rejected. Figure 3-4 shows the algorithm for checking databases.

Server loading is monitored because there may are many user invocations. We need to get server's statuses in order to avoid the loading too heavy. This system records the date, time, and size of the genomic database into Information Database when G-BLAST is invoked, and records the date and time again after the invocation is accomplished. Since these records tell us the run time of each job and how many jobs are running, we can utilize them to estimate the run time of a new job in advance. Moreover, we can also use them to control server loading with this component by deciding which job runs immediately and which job must wait for a while. Therefore, we propose a Schedule strategy to do this work, and its algorithm is shown in Figure 3-5. The main technologies of implementation Schedule are using the Semaphore and Shared Memory. Because G-BLAST execution each job is

generating a new process via *fork()*System Call, each job has its own PID and memory area in Linux System. Hence, we have to solve share variable problem if we feel like using Semaphore to implement Schedule. So we utilize Shared Memory technology to solve this problem.

```
1   Checking_db(){
2       if (The sorted database does not exist){
3         if (The original database does not exist)
4           Notify the user that the sequence database does not exist ;
5         else
6           Sorting() ;
7       }
8       else{
9         if (The information of physical file does not match that of the Information
            Database){
10          if (The original database does not exist)
11            Notify the user that the sequence database does not exist ;
12          else
13            Sorting() ;
14        }
15      }
16  }
17
18  //Sorting database in sequence length order
19  Sorting(){
20      Save the information of original database into Information Database ;
21      while (Not finished reading sequences){
22        Read sequences ;
23        Compute the length of each sequence ;
24      }
25      Using Quick Sort Algorithm sorts sequences according to sequence length in
          descending order ;
26      Creating a new sorted database with the sorted sequences ;
27      Save the information of sorted database into Information Database ;
28  }
```

Figure 3-4 The Database Checking Algorithm

```
1   Schedule(job_id){
2     inQueue = False ;
3     if(G-BLAST System is busy){
4       inQueue = True ;
5       InQueue(job_id) ;
6       Hold(job_id) ;
7       Sleep(job_id) ;
8     }
9     while(This job is woke up by Monitor System and
            inQueue = True){
10      inQueue = False ;
11      DeQueue(job_id) ;
12      Free(job_id) ;
13    }
14    Deliver this job to the Job Dispatch System ;
15  }
```

Figure 3-5 The Schedule Algorithm

3.3 Information System

This system collects the information (e.g., execution time) of each site's own jobs from the Job Monitor System and computes the performance. Please note that a site means a Cluster unit or a stand-alone machine. The performance enables Job Dispatch System to select sites that they are assigned to perform the job together. Job Monitor System records the size of each fragment and job execution time and then notifies this system after the site's own job is accomplished.

We propose a method to compute the performance on site *j* called *Pj* as shown in equation 1. Because the time complexity of a local alignment search is close to O(*mn*) in mpiBLAST, our method utilizes this worse-case running time to compute a performance value. Furthermore, the performance of the site, which is in the Information Database, will be

adjusted by new performance value, and Job Dispatch System will reference this value to dispatch next job. The parameters *m* and *n* represent the length of query sequence and the length of the database. If there are *i* query sequences ($i > 1$), the time complexity will be $O(m_1n + m_2n + m_3n + \ldots + m_in)$. Therefore, the performance value named alignment rate is $(m_1n + m_2n + m_3n + \ldots + m_in)$ divide by execution time. Moreover, this method computes average alignment rate of the last five times in order to let Job Dispatch System can select sites more accurately. For example, if a site had been performed five jobs and their alignment rates are 15, 16, 17, 18, and 19. Now, there is a job which alignment rate is 20. This system sums of the last five times alignment rates which are 16, 17, 18, 19, and 20 and then divide by 5. So the latest performance of this site will become 18. The times of 5 was derived empirically from try-and-error using many different query size, database size, and number of sites. Related variables are defined below.

- *d*: The number of history records in the last four times if the records more than four times. ($0 \leqq d \leqq 4$)
- *k*: The number of sequences of query file.
- m_i: The length of the i^{th} query sequence.
- *n*: The length of the database.
- T_{exec}: The execution time of the job.

$$P_j = \sum_{h=0}^{d+1} \left(\frac{\sum_{i=1}^{k} m_i \cdot n}{T_{exec}} \right)_h \cdot \frac{1}{d+1} \quad \textbf{(1)}$$

3.4 Job Dispatch System

This system fetches and analyzes system information via Information System and then utilizes it to select sites, then informs Segmentation Database System how to segment the database and then deliver the job to Job Monitor System.

In this work, we propose a method for selection sites, and the Equation 2 is estimate the fragment size if this site is selected. Figure 3-6 shows the algorithm for selection the best sites, and Figure 3-7 shows the Job Dispatch System flowchart. And related variables are defined below.

- N: Number of total sites, where $N > 0$
- b: Number of sites which are selected, where $b > 0$
- *db_size*: The size of genomic database
- $C(b)$: Collection of all subsets of size b from all sites of N
- $n\_C(b)$: Number of elements of $C(b)$
- $S(i)$: ith subset of $C(b)$
- $P_{S(i)}$: The total performance of $S(i)$
- $P_{S(i),j}$: The performance of site j from $S(i)$
- $D_{S(i),j}$: The available disc space of site j from $S(i)$
- Tf_{ij}: The theoretical fragment size on site j

$$Tf_{ij} = \frac{P_{S(i),j} \cdot db\_size}{P_{S(i)}} \quad \textbf{(2)}$$

```
1   Selection(N, b){
2     if(N = 1){
3       if(D_{S(1), 1} ≦ db_size)
4         return 0 ;
5       return P_{S(1), 1} ;
6     }
7     else{
8       i = 1 ;
9       while(i ≦ n_C(b)){
10        j = 1 ;
11        while(j ≦ b){
12          if(D_{S(i), j} ≦ Tf_{ij}){
13            P_{S(i)} = 0 ;
14            break ;
15          }
16          j++ ;
17        }
18        i++ ;
19      }
20      return maximum({P_{S(i)}}) ;
21    }
22  }
23
24  maximum({P_{S(i)}}){
25    index = 1 ;
26    i = 2 ;
27    while(i ≦ n_C(b)){
28      if(P_{S(index)} < P_{S(i)})
29        index = i ;
30      i++ ;
31    }
32    Save the information of S(index) ;
33    return P_{S(index)} ;
34  }
```

Figure 3-6 Sites Selection Algorithm

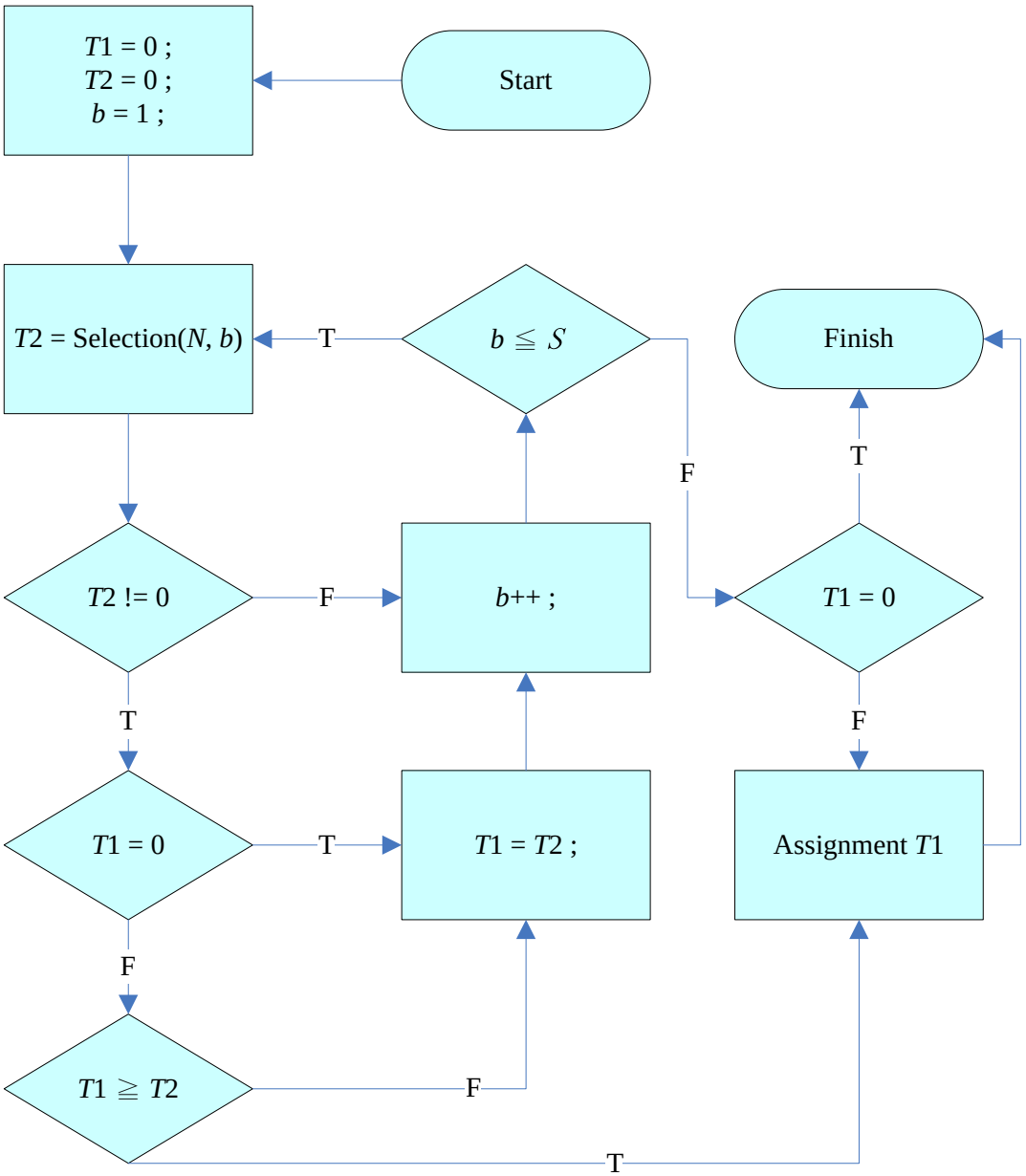


Figure 3-7 Job Dispatch System Flowchart

3.5 Segmentation Database System

This system segments genomic sequence databases. You perhaps have a question that is why you do not use *mpiformatdb* command provided by mpiBLAST.

The reason is that the *mpiformatdb* command uses simple division, putting the sequences in each fragment of the original sequence database in a round-robin strategy and divides

sequence databases into approximately equal size fragments. Since our system has a hierarchical architecture and heterogeneous environment, each site may have its own computational ability and network speed, dividing sequence databases into approximately equal size fragments is not a good method for us. Moreover, because mpiBLAST uses Local Alignment algorithm to compare a sequence against a database, the time complexity is associated with sequence length and query length. Consequently, it may lead to faster sites needing to wait for slower sites to complete their sub jobs, and strongly affects performance. Therefore, we divide sequence databases into different size fragments according to site speed and capacity. For example, if a job is performed by two sites, A and B, and site A has twice the total performance of site B. This component will divide the sequence database into a two-thirds fragment and a one-third fragment, and assign the two-thirds fragment to site A and the other to site B. This example is shown in Figure 3-8.

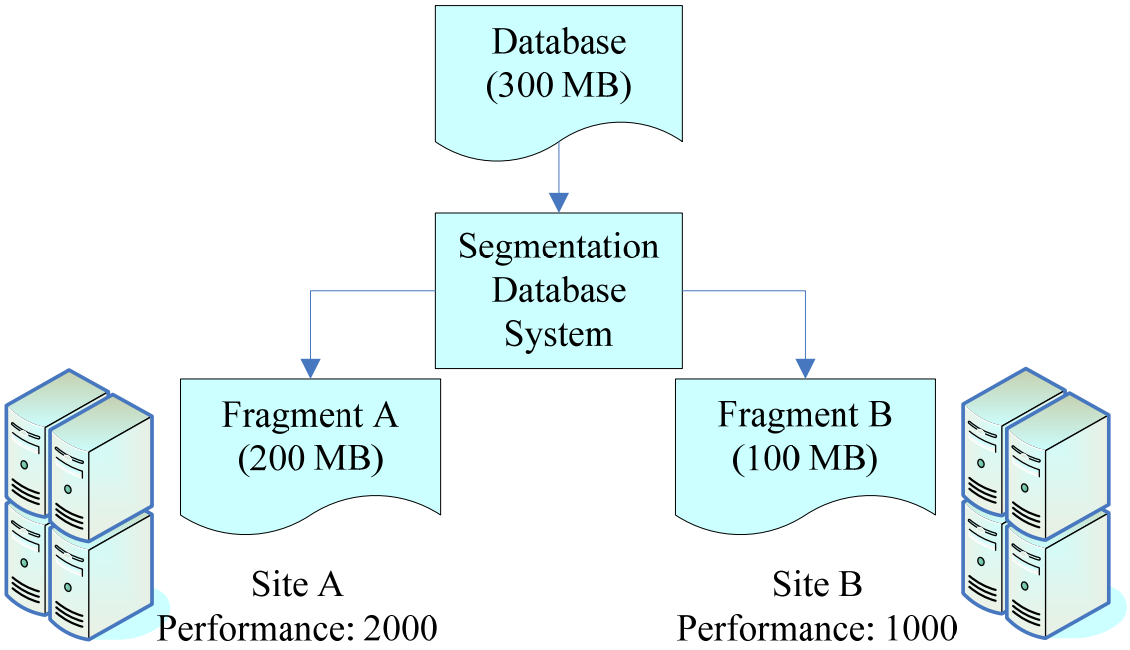


Figure 3-8 Example for Segmentation Database System

Figure 3-9 shows the Database Segmentation System flowchart. In this work, we propose a method for segmentation database, and both Equations 3 and 4 play an important role in deciding which file needs to be written. Related variables are defined below.

- N: Number of total sites
- S: Number of total sequences
- L_i: Length of sequence i
- P: The sum of performance of N sites
- P_j: The performance of site j
- C_j: The current total sequences length of file j
- Tl_{ij}: The theoretical total sequences length of file j, excluding sequence i
- $Tl_{ij}^{'}$: The theoretical total sequences length of file j, including sequence i
- D_j: The difference between C_j and (Tl_{ij} + L_i)

$$Tl_{ij} = \frac{P_j}{P} \cdot \sum_{l=1}^{i-1} L_l \quad \textbf{(3)}$$

$$Tl_{ij}^{'} = \frac{P_j}{P} \cdot \sum_{l=1}^{i} L_l \quad \textbf{(4)}$$

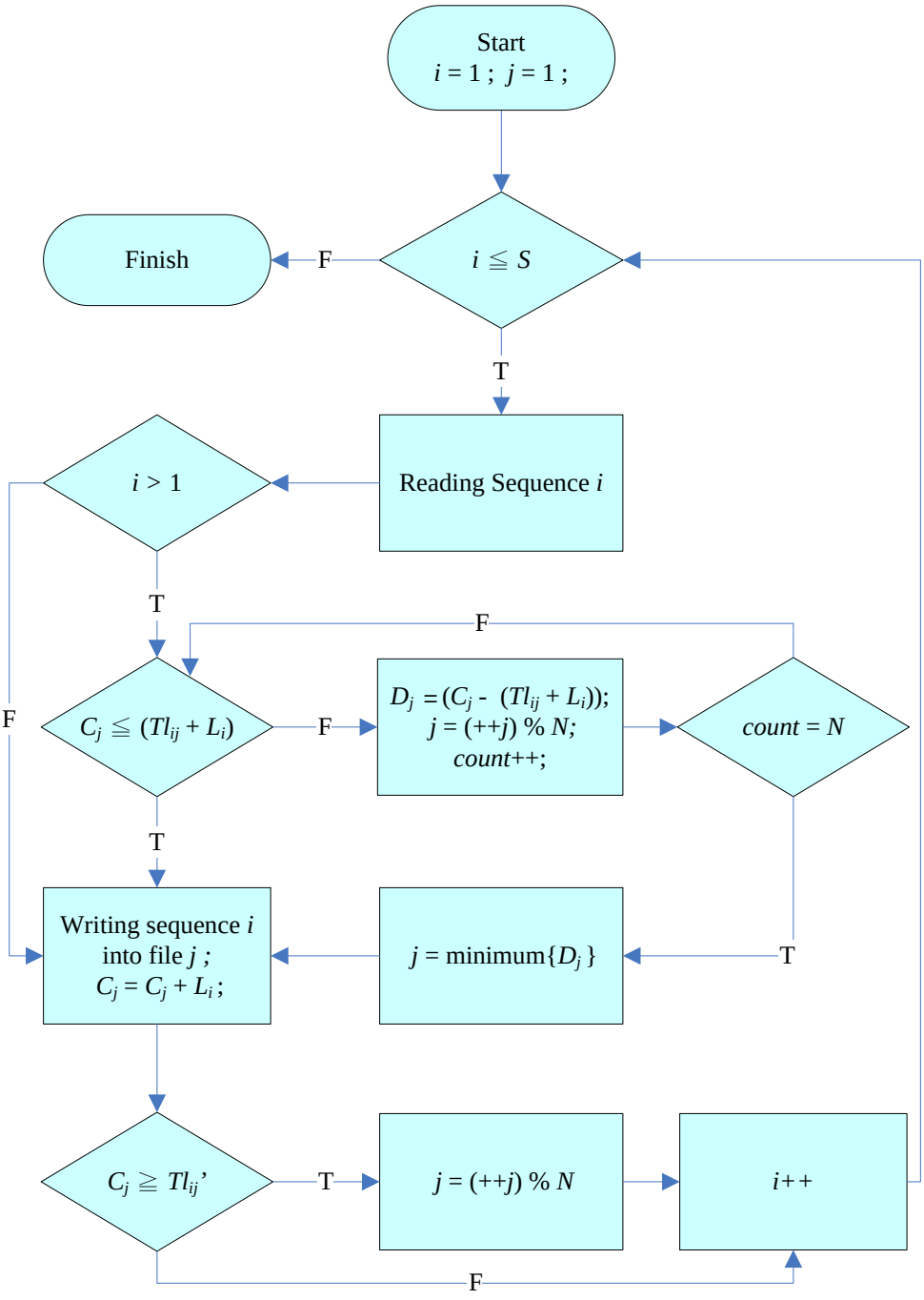


Figure 3-9 The Database Segmentation System Flowchart

3.6 Job Monitor System

It dispatches sub-jobs assigned by the Job Dispatch System and delivers the fragments, which are divided by the Database Segmentation System, to each site via GridFTP. It also monitors and records the statuses of each site, and notifies the Information System to digest this information after any site finishes its own job.

Moreover, it dynamically generates a C++ source code named run_mpiBLAST.cpp and a shell script file. The functions of this program are getting query sequence and sub-database via Grid FTP, execution grid-proxy-init command, execution mpiformatdb command, execution mpiblast command, and return sub-result to G-BLAST Server. The functions is compiling the C++ source code on slave node, then performing it and then return the result file to G-BLAST.

3.7 Combination Results System

Because sequence databases are segmented by the Segmentation Database System in advance, many unwanted result files are generated, rather than one complete result file. This component combines all result files belonging to a job into a complete result file and shows it on the user's display via the User Client Application.

3.8 mpiBLAST Cluster System

The mpiBLAST Cluster System software architecture is shown in Figure 3-10. This Cluster System consisted mpiBLAST Server and mpiBLAST Client.

3.8.1 mpiBLAST Server

This is a master node of a Cluster System. First, it receives a C++ source program named run_mpiBLAST.cpp from G-BLAST Server. The functions of this program are getting query sequence and sub-database via Grid FTP, execution *grid-proxy-init* command, execution *mpiformatdb* command, execution *mpiblast* command, and return sub-result to G-BLAST Server. Second, it compiles run_mpiBLAST.cpp. Third, it executes *run_mpiBLAST*. These steps are executed by G-BLAST Server via the Expect script.

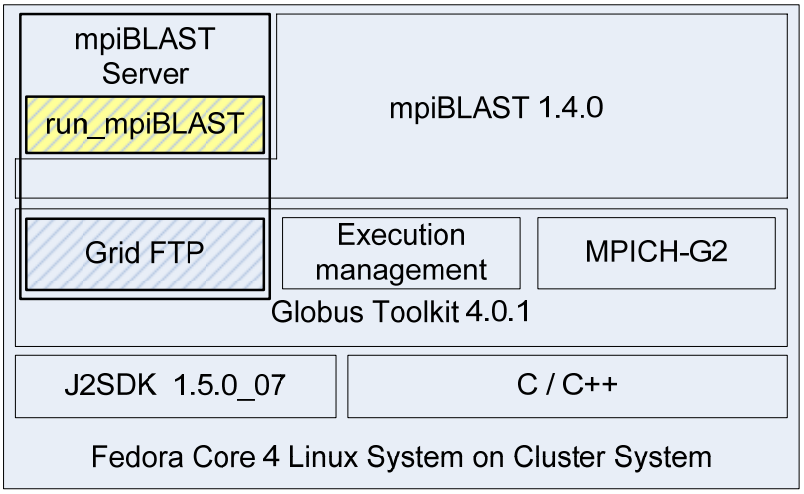


Figure 3-10 The mpiBLAST Cluster System Software Architecture

3.8.2 mpiBLAST Client

These nodes belong to a Cluster System. They comply with master node assignment to execute mpiBLAST program.

Chapter 4

Experimental Environments

4.1 mpiBLAST on PC Cluster Environment

Our PC Cluster is a low cost Beowulf-type class supercomputer that utilizes multi-computer architecture for parallel computations. It consists of ten PC-based symmetric multiprocessors (SMP) connected by one 16-port 1Gbps Ethernet switches with Fast Ethernet Interface at HPC laboratory in Tunghai University. Therefore, there are total 20 processors for our PC Cluster experiment. Moreover, this environment is made up one server node and nine computing nodes. The server node has an Intel Pentium D 2.8GHz processor and 1GBytes DDRII memory. Among these nine computing nodes, three of them are Intel Pentium D 2.8GHz nodes, while the others are six AMD Athlon 64 X2 Dual Core 3800+ nodes; each node has 1GBytes DDRII memory. The PC Cluster hardware configuration is shown in Table 4-1, the logical diagram of PC Cluster architecture is shown in Figure 4-1, and the PC Cluster software architecture is shown in Figure 4-2.

Table 4-1 PC Cluster Hardware Configuration

	Host Name	**CPU Type**	**Speed**	**Memory**	**Network**
Server	G-BLAST	Intel(R) Pentium(R) D CPU	2.8 GHz	256MB x 4	1 Gbps
Bio_AMD	bio01 ~ 06	AMD Athlon(tm) 64 X2 Dual Core	3800+	512MB x 2	1 Gbps
Bio_DELL	bio07 ~ 09	Intel(R) Pentium(R) D CPU	2.8 GHz	512MB x 2	1 Gbps

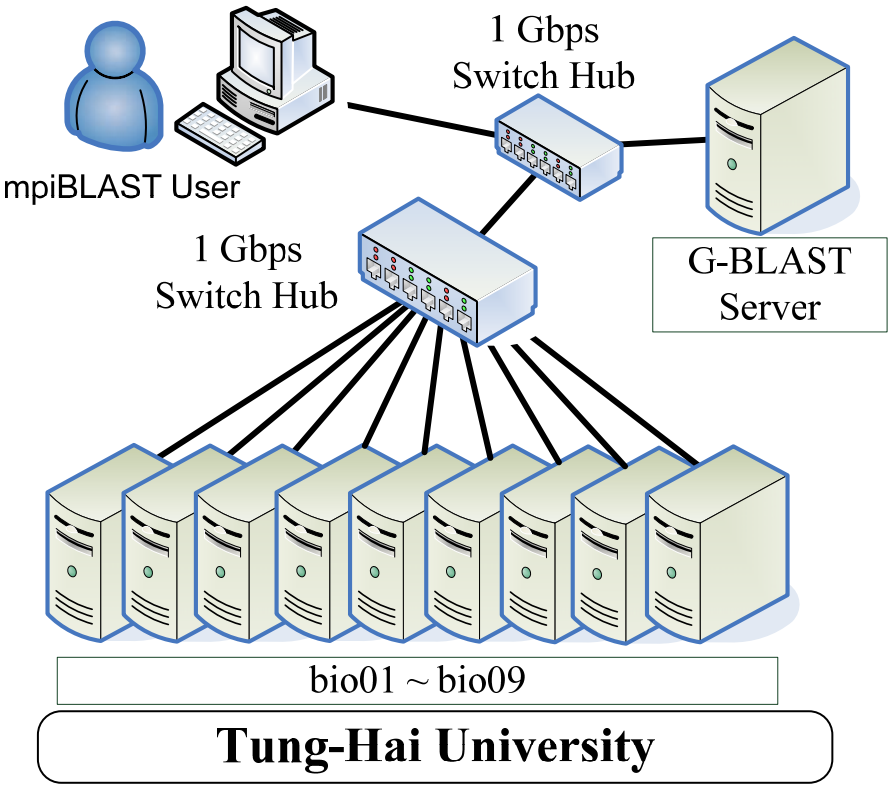


Figure 4-1 The Logical Diagram of PC Cluster Architecture

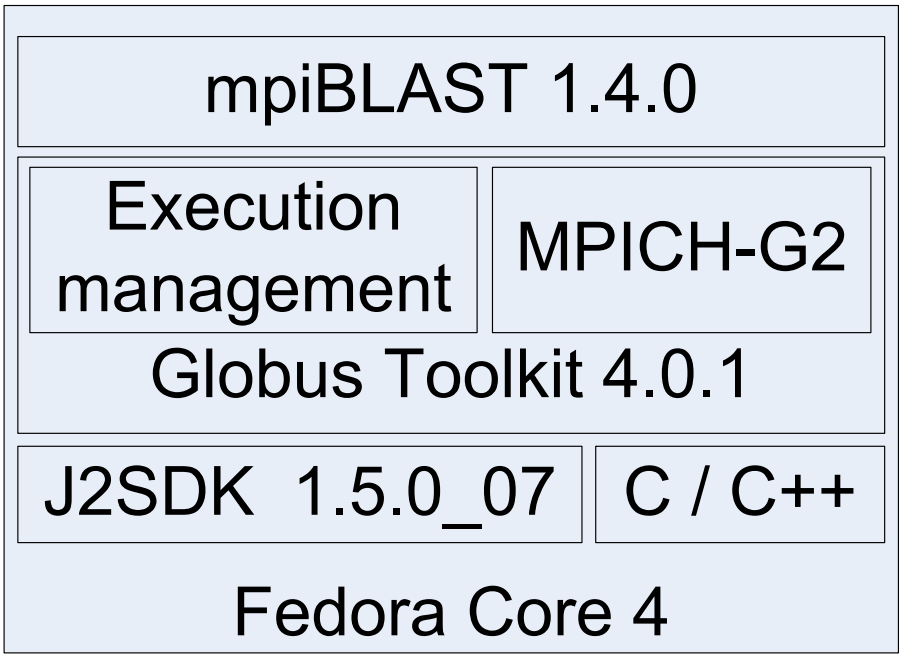


Figure 4-2 PC Cluster Software Architecture

4.2 G-BLAST on Grid Environment

The G-BLAST Framework experiments were performed on a Grid test-bed which contains three Cluster systems. One consists of one server node and five computing nodes in Tunghai University, another consists of one server node and two computing nodes in Tunghai University, and the other consists of one server node and three computing nodes in Hsiuping Institute of Technology. The G-BLAST hardware configuration is shown in Table 4-2, the logical diagram of Grid architecture is shown in Figure 4-3, and the software architecture is shown in Figure 3-10. In this environment, the G-BLAST server is not an mpiBLAST master node anymore. G-BLAST server is a Grid master node, and G-BLAST framework is built on it. In addition, the bio01, bio07, and gridhit0 are mpiBLAST server node in their own Cluster System.

Table 4-2 G-BLAST Hardware Configuration

	Host Name	**CPU Type**	**Speed**	**Memory**	**Network**
Server	G-BLAST	Intel(R) Pentium(R) D CPU	2.8 GHz	256MB x 4	1 Gbps
Bio_AMD	bio01 ~ 06	AMD Athlon(tm) 64 X2 Dual Core	3800+	512MB x 2	1 Gbps
Bio_DELL	bio07 ~ 09	Intel(R) Pentium(R) D CPU	2.8 GHz	512MB x 2	1 Gbps
HIT	gridhit0 ~ 3	Intel(R) Pentium(R) 4 x 2	2.8 GHz	512MB	100 Mbps

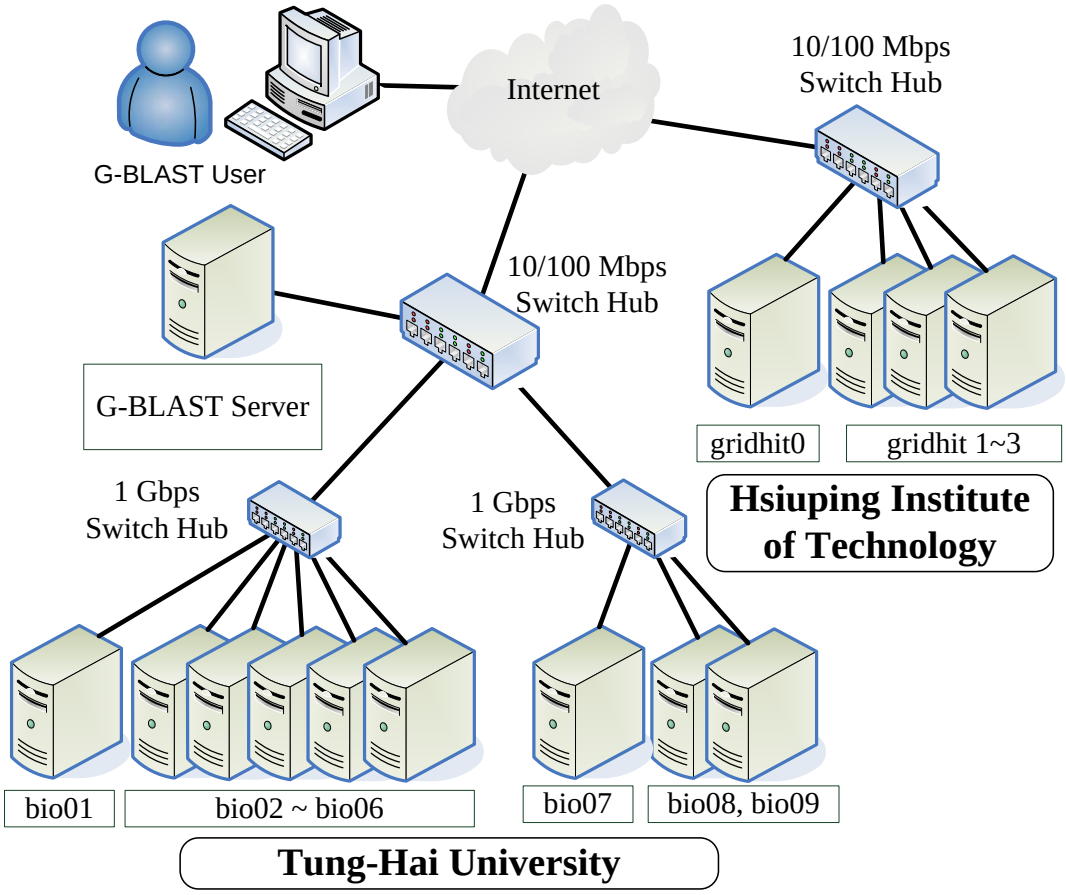


Figure 4-3 The Logical Diagram of Grid Architecture

Chapter 5

Experimental Results

Both PC Cluster and Grid experiment are using two query files nt.5706771 (containing 1 sequence, 195,299 letters) and nt.ests (containing 1,931 sequences, about 583,468 letters) against four genomic databases. The specifications of databases are shown in Table 5-1. These databases and query file are maintained by NCBI.

5.1 mpiBLAST on PC Cluster Environment (Query: nt.5706771)

Our experiments are execution mpiBLAST on 2, 4, 6, 8, 10, 12, 14, 16, 18, and 20 processors to compare the time. The specification of number of processors is shown in Table 5-2. In order to aspire get more accurate data, we execute 5 times per experiment and calculate the average time. Because we will use these experiment results compared to the experiments of G-BLAST, the time of these experiments are including *mpiformatdb* phase and *mpiblast* phase.

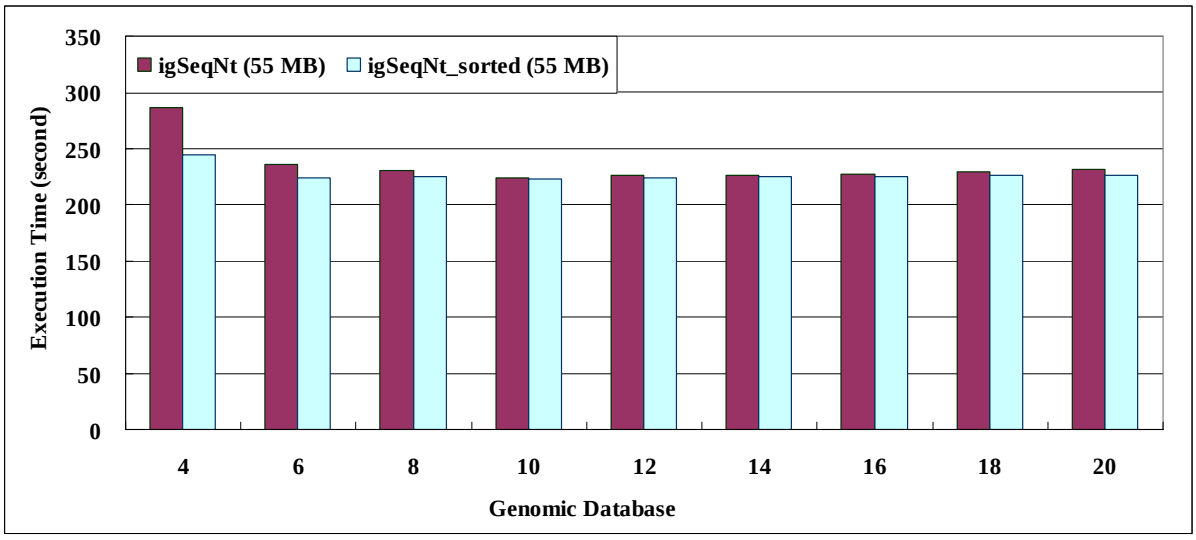
Table 5-1 Database Specification

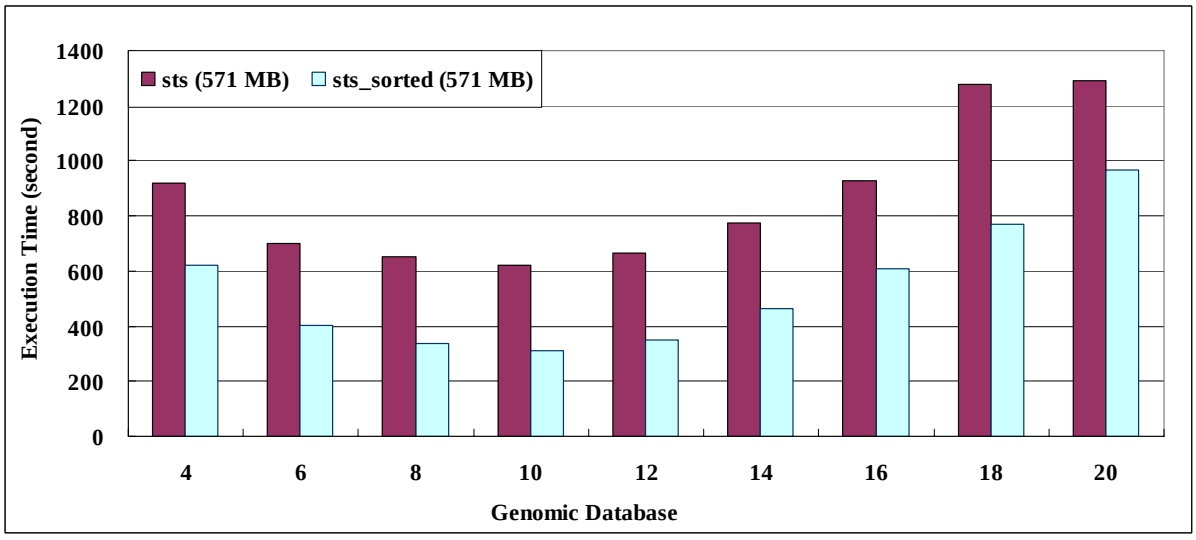
Database Name	Number of Sequences	Number of Letters	Size (MB)
igSeqNt	570,733	49,362,505	55
sts	882,506	496,121,312	571
patnt	3,256,906	1,981,990,296	2,142
est_mouse	4,719,477	2,193,300,391	2,773

Table 5-2 Number of Processors Specification

Number of Processors	Machines
4 ~ 6	G-BLAST+bio07 ~ 09
10 ~ 20	G-BLAST + bio07 ~ 09 + bio01 ~ 06

Figure 5-1 is the mpiBLAST software experimental results that using nt.5706771 against four genomic databases. These results compare the execution time between original database and sorted database. Notice that these sorted databases (e.g., igSeqNt_sorted) are sorted by G-BLAST in advance so they are formatted by *mpiformatdb* with *--skip-reorder* parameter.





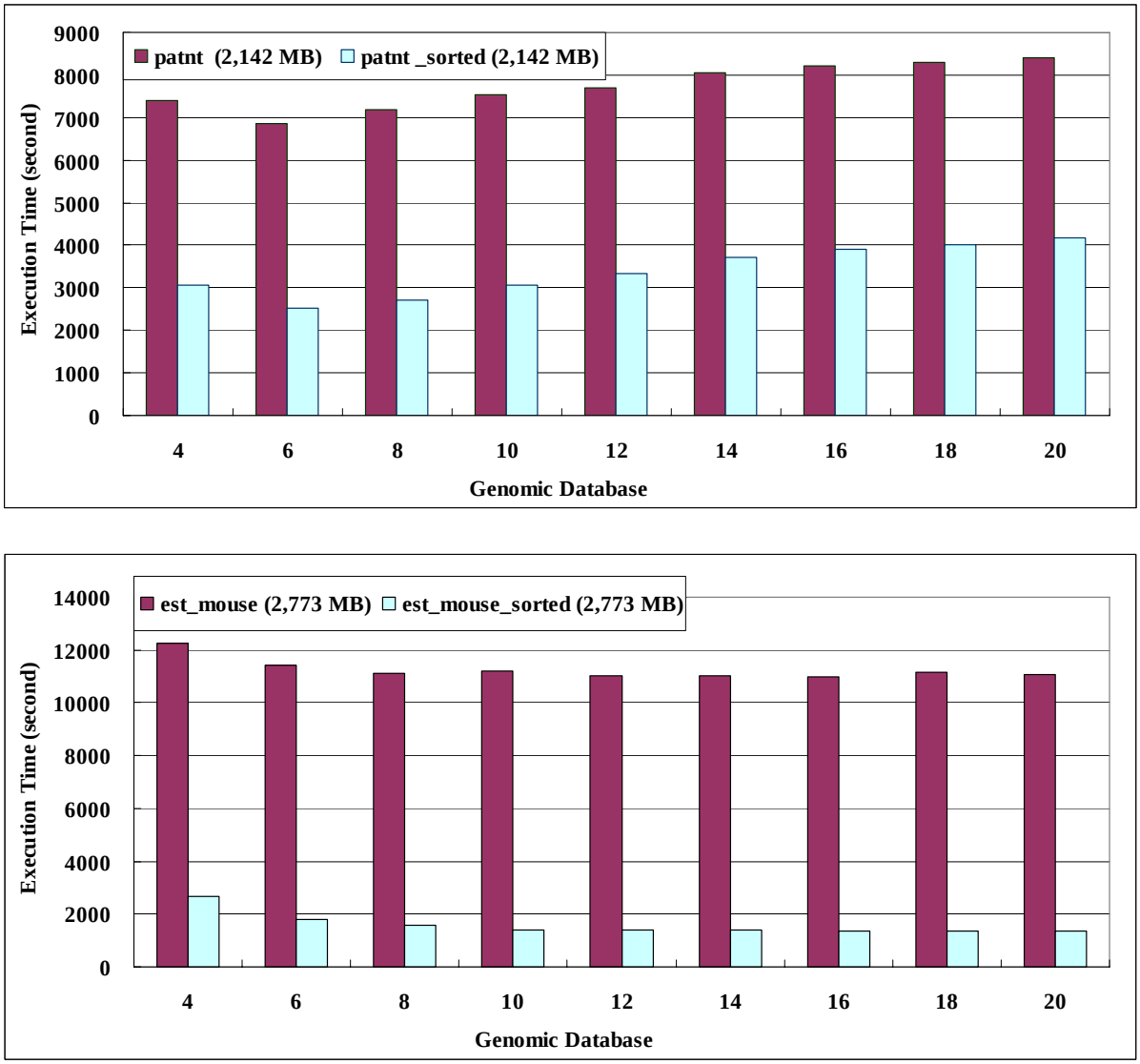


Figure 5-1 Execution Time of mpiBLAST in PC Cluster (Query: nt.5706771)

From the results, we can easily to find out the Cluster System can reduce more time to perform the sequence alignment. However, we also can observe inverted scaling for mpiBLAST (i.e., longer execution time when using more processors) when running on more than 10 processors in database (igSeqNt, 55MB), running on more than 10 processors in database (sts, 571MB), running on more than 6 processors in database (patnt, 2,142MB), and running on more than 16 processors in database (est_mouse, 2,773MB). Therefore, we observed that the size of database is weakly pertinent to the execution time of scaling of

mpiBLAST. Furthermore, the user usually assignment the maximum number of processors (e.g., the maximum number of processors is 20 in our PC Cluster environment) for an mpiBLAST job. In addition, we also can observation that the reordering database is wasting time, especially in large database, so this phase is unnecessarily.

5.2 G-BLAST on Grid Environment (Query: nt.5706771)

Figure 5-2 shows the total turnaround time of using G-BLAST Framework compared to execution time of mpiBLAST on PC Cluster that using nt.5706771 against four genomic databases. Notice that the best case means the minimum execution time in the nine experiment results on PC Cluster experiment, and the worst case means the maximum execution time. In the original database portion, we can observation that G-BLAST is significantly faster than the best time of mpiBLAST.

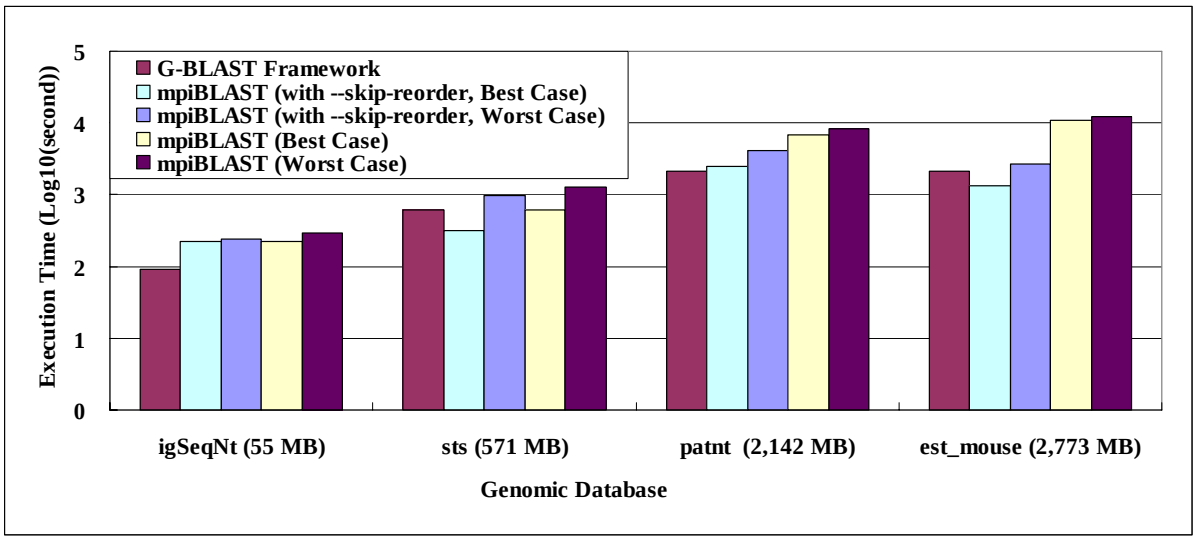


Figure 5-2 G-BLAST Framework vs. mpiBLAST (Query: nt.5706771)

In here, the most improvable execution time is that it skips reordering the genomic database. Therefore, G-BLAST has advantage over mpiBLAST in that G-BLAST is sorting

database in advance. In the sorted database portion, because G-BLAST can choose the most adaptive sites via Job Dispatch System, the performance of alignment should scale well in each case. However, the preprocessors (e.g., Segmentation Database System, Job Monitor System, Information System, and Combination System) are adding some overheads on G-BLAST but mpiBLAST has not these overloads. Therefore, the performance seems weakly good in the "*--skip-reorder*" cases. Figure 5-3 illustrates a comparison of speedup between G-BLAST Framework and mpiBLAST with and without "*--skip-reorder*" case for a query file (nt.5706771) against four databases.

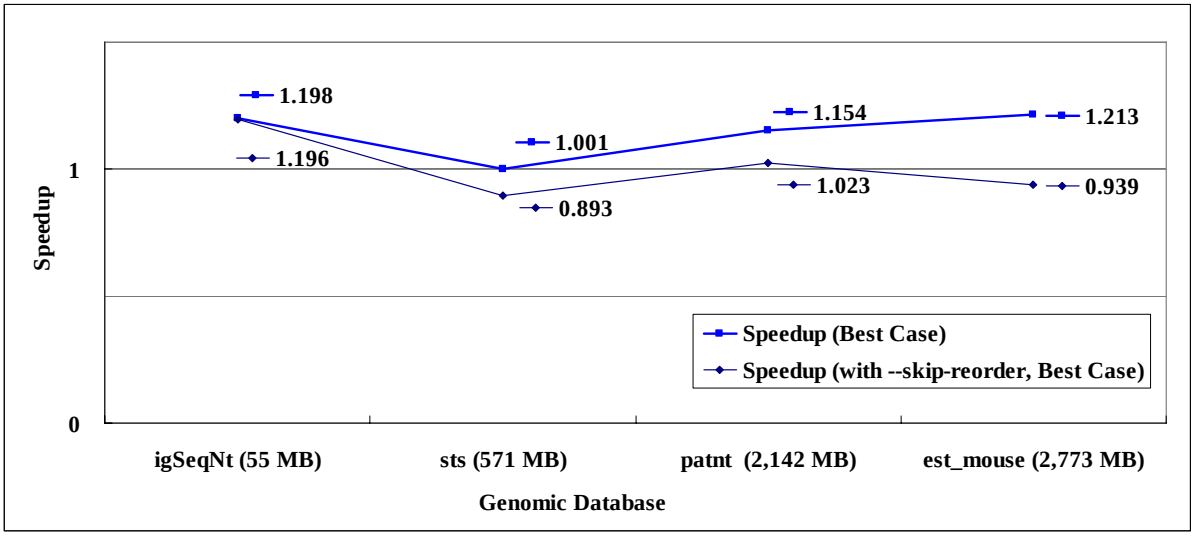
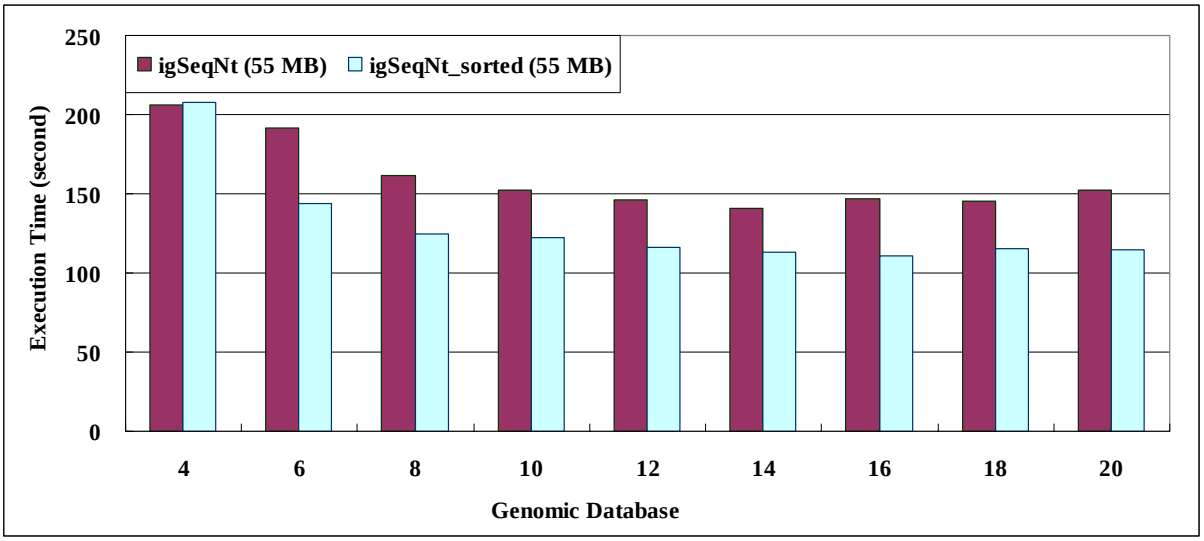
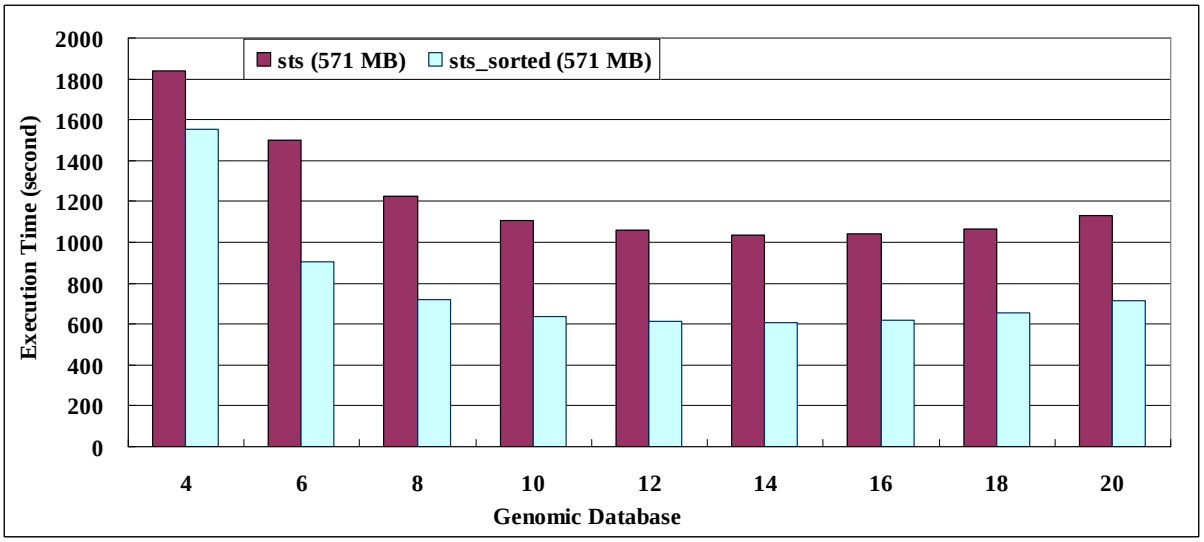


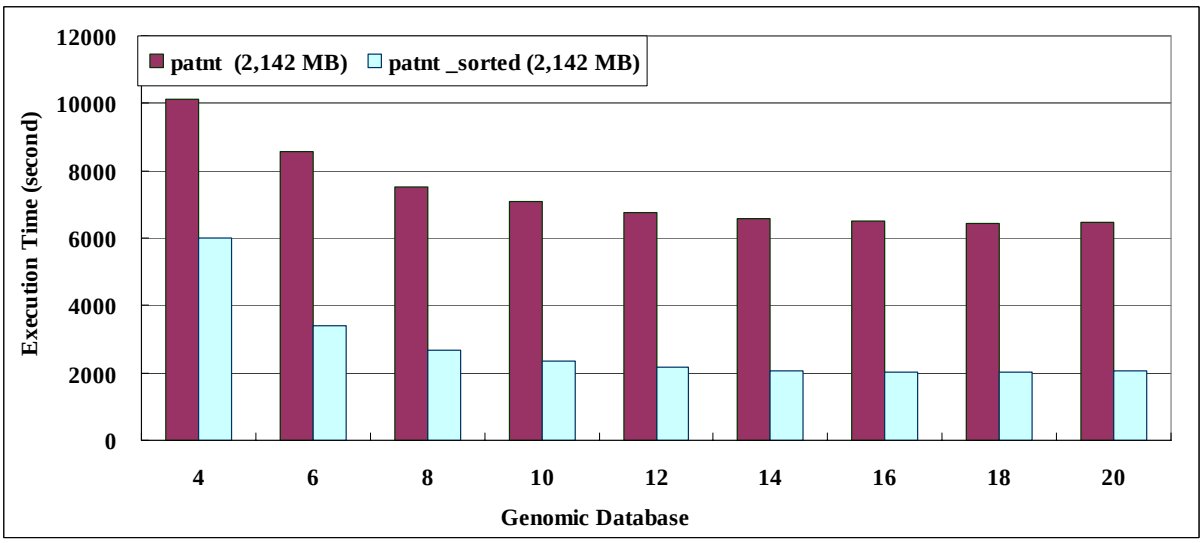
Figure 5-3 The Speedup of G-BLAST Framework (Query: nt.5706771)

5.3 mpiBLAST on PC Cluster Environment (Query: nt.ests)

Figure 5-4 is the mpiBLAST software experimental results that using nt.ests against four genomic databases. From the results, we can easily to find out the Cluster System can reduce more time to perform the sequence alignment.


igSeqNt (55 MB)
igSeqNt_sorted (55 MB)
Execution Time (second)
250
200
150
100
50
0
4
6
8
10
12
14
16
18
20
Genomic Database


sts (571 MB)
sts_sorted (571 MB)
Execution Time (second)
2000
1800
1600
1400
1200
1000
800
600
400
200
0
4
6
8
10
12
14
16
18
20
Genomic Database


patnt (2,142 MB)
patnt _sorted (2,142 MB)
Execution Time (second)
12000
10000
8000
6000
4000
2000
0
4
6
8
10
12
14
16
18
20
Genomic Database

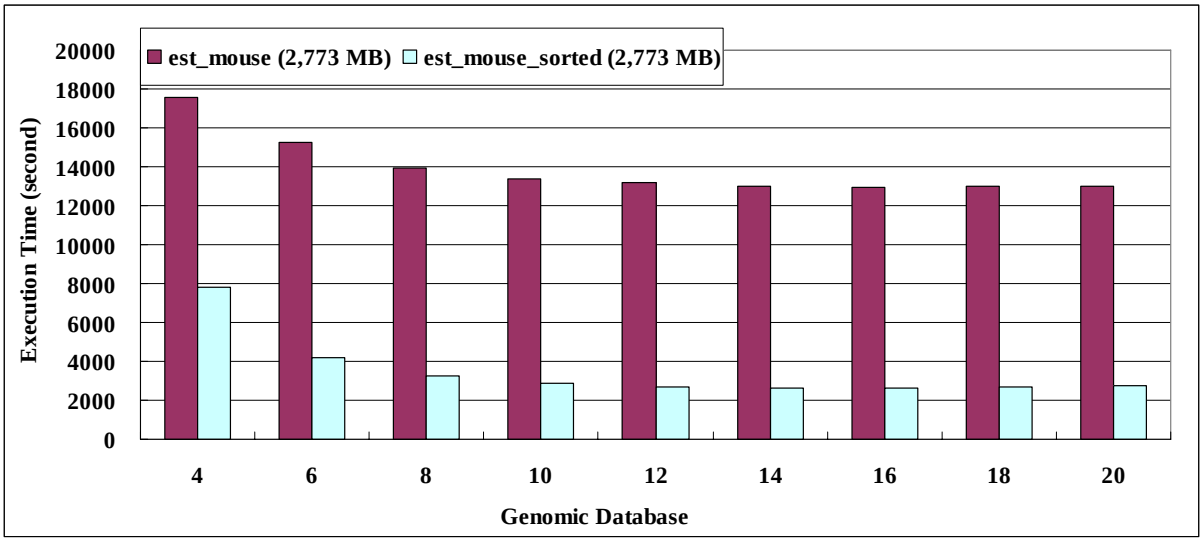


Figure 5-4 Execution Time of mpiBLAST in PC Cluster (Query: nt.ests)

However, we also can observe inverted scaling for mpiBLAST (i.e., longer execution time when using more processors) when running on more than 14 processors in database (igSeqNt, 55MB), running on more than 14 processors in database (sts, 571MB), running on more than 18 processors in database (patnt, 2,142MB), and running on more than 16 processors in database (est_mouse, 2,773MB). Therefore, we observed that the size of database is weakly pertinent to the execution time of scaling of mpiBLAST. Furthermore, the user usually assignment the maximum number of processors (e.g., the maximum number of processors is 20 in our PC Cluster environment) for an mpiBLAST job. In addition, we also can observation that the reordering database is wasting time, especially in large database, so this phase is unnecessarily.

5.4 G-BLAST on Grid Environment (Query: nt.ests)

Figure 5-5 shows the total turnaround time of using G-BLAST Framework compared to execution time of mpiBLAST on PC Cluster that using nt.ests against four genomic

databases. Notice that the best case means the minimum execution time in the nine experiment results on PC Cluster experiment, and the worst case means the maximum execution time. In the original database portion, we can observation that G-BLAST is significantly faster than the best time of mpiBLAST.

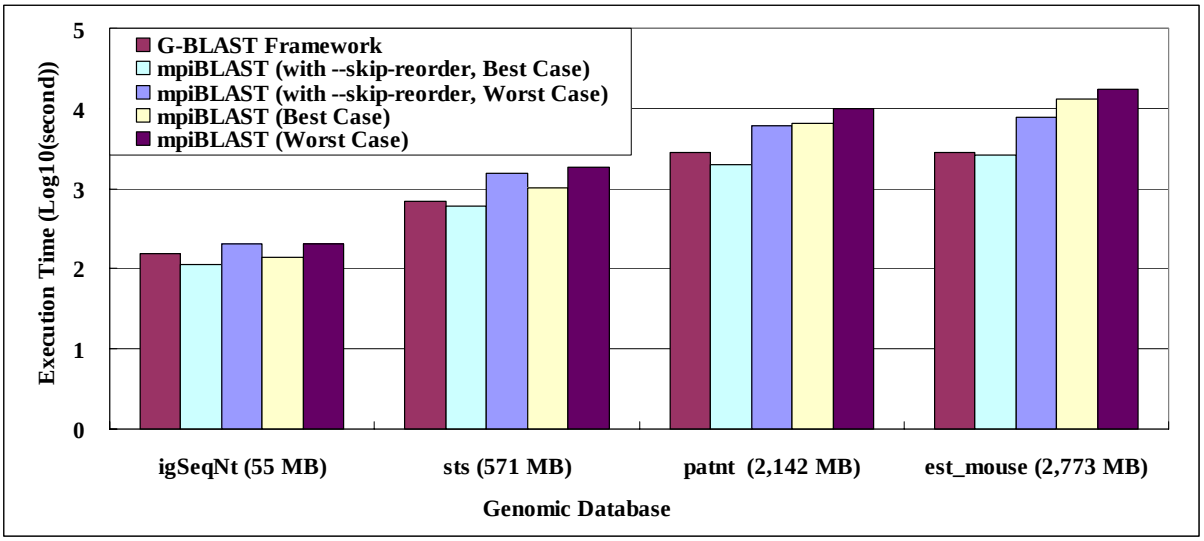


Figure 5-5 G-BLAST Framework vs. mpiBLAST (Query: nt.ests)

In here, the most improvable execution time is that it skips reordering the genomic database. Therefore, G-BLAST has advantage over mpiBLAST in that G-BLAST is sorting database in advance. In the sorted database portion, because G-BLAST can choose the most adaptive sites via Job Dispatch System, the performance of alignment should scale well in each case. However, the preprocessors (e.g., Segmentation Database System, Job Monitor System, Information System, and Combination System) are adding some overheads on G-BLAST but mpiBLAST has not these overloads. Therefore, the performance seems weakly in the "*--skip-reorder*" cases. Figure 5-6 illustrates a comparison of speedup between G-BLAST Framework and mpiBLAST with and without "*--skip-reorder*" case for a query file (nt.est) against four databases.

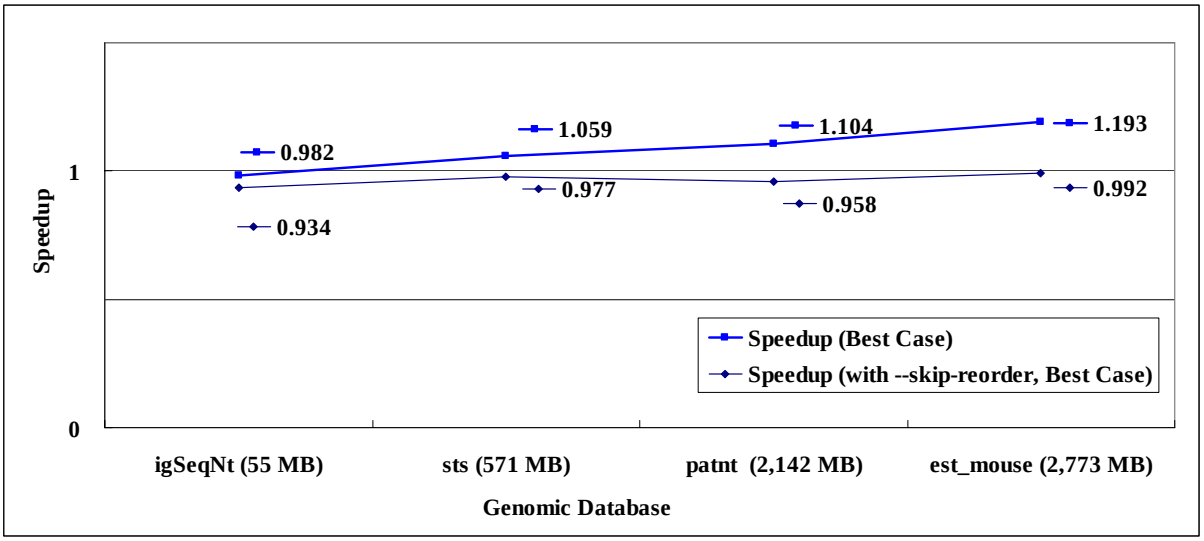
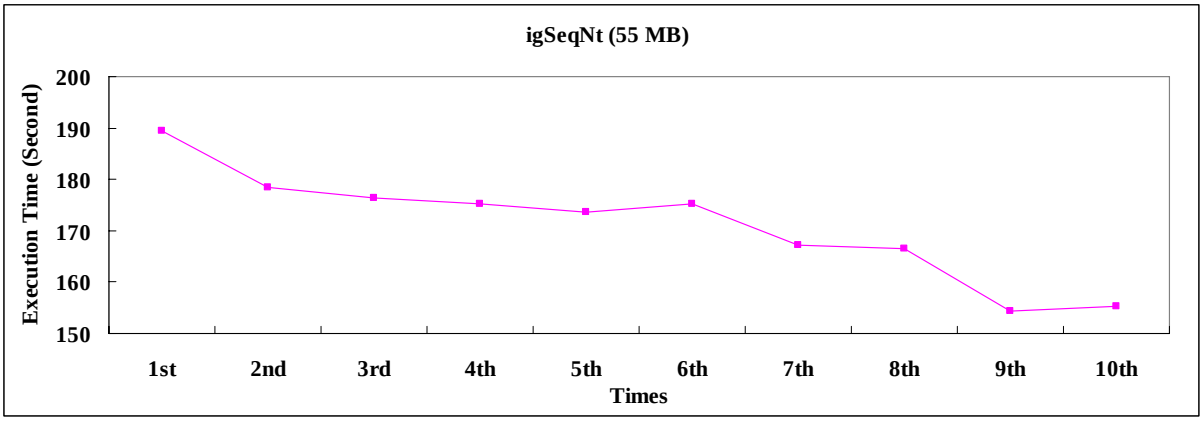
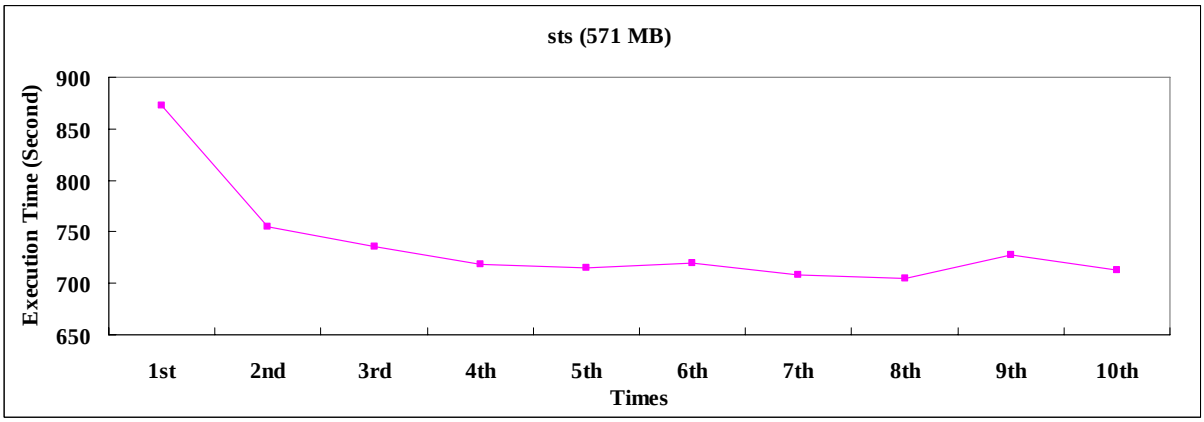


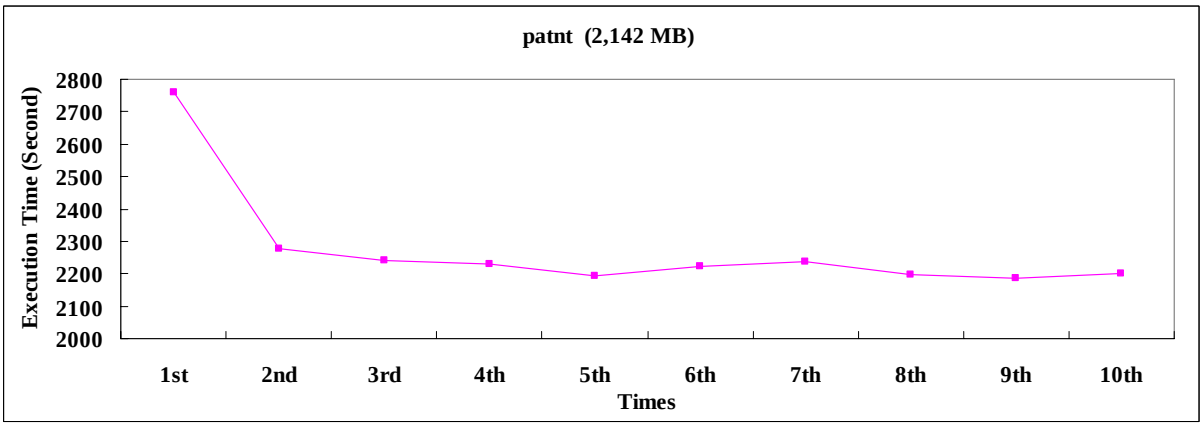
Figure 5-6 The Speedup of G-BLAST Framework (Query: nt.ests)

5.5 Learning Curve

Figure 5-7 shows the learning curve of G-BLAST Framework. These experimental results use nt.ests against four genomic databases respectively. We set the same performance value to PC Clusters (Bio_AMD, Bio_DELL, and HIT) at the start of each experiment in order to highlight the learning ability of G-BLAST. The performance value will be modified in accordance with execution performance after each experiment, and this performance value is computed by Information System. Because the fragment size is associated with performance value, we can observe the variation of performance value from Figure 5-8. From these results, we can easily to find out the G-BLAST has good learning ability.


igSeqNt (55 MB)
Execution Time (Second)
200
190
180
170
160
150
1st
2nd
3rd
4th
5th
6th
7th
8th
9th
10th
Times


sts (571 MB)
Execution Time (Second)
900
850
800
750
700
650
1st
2nd
3rd
4th
5th
6th
7th
8th
9th
10th
Times


patnt (2,142 MB)
Execution Time (Second)
2800
2700
2600
2500
2400
2300
2200
2100
2000
1st
2nd
3rd
4th
5th
6th
7th
8th
9th
10th
Times

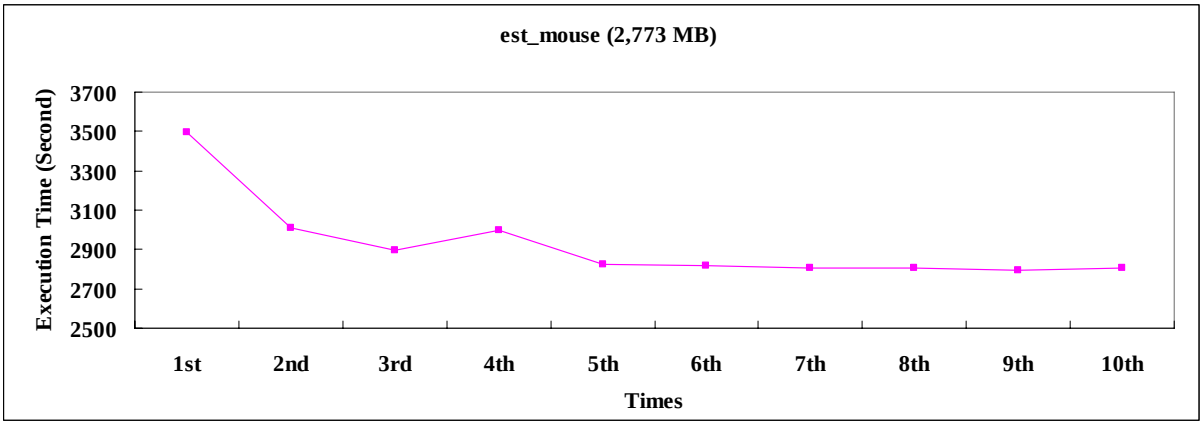
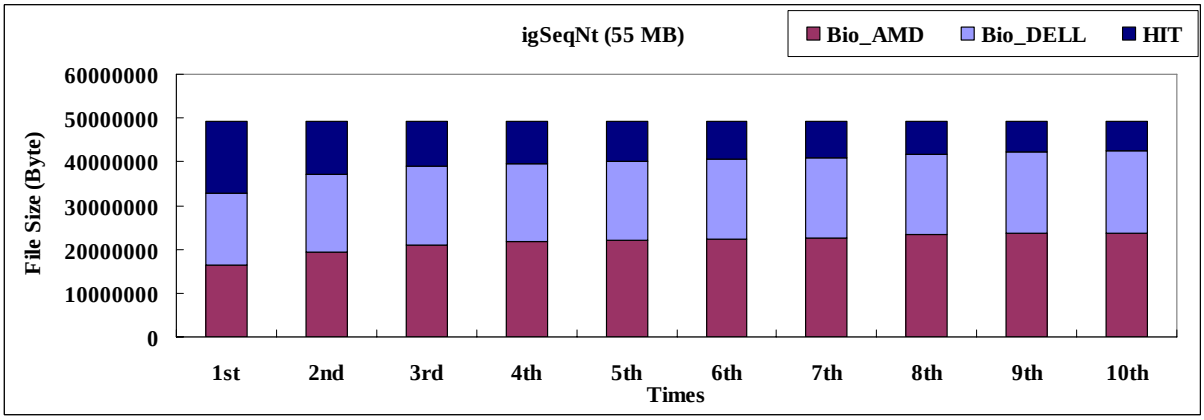
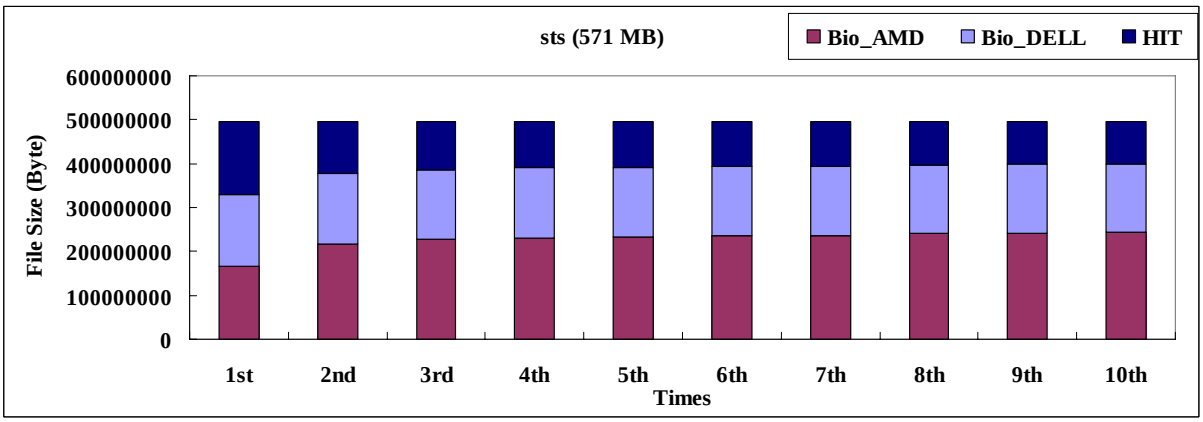


Figure 5-7 The Learning Curve of G-BLAST Framework





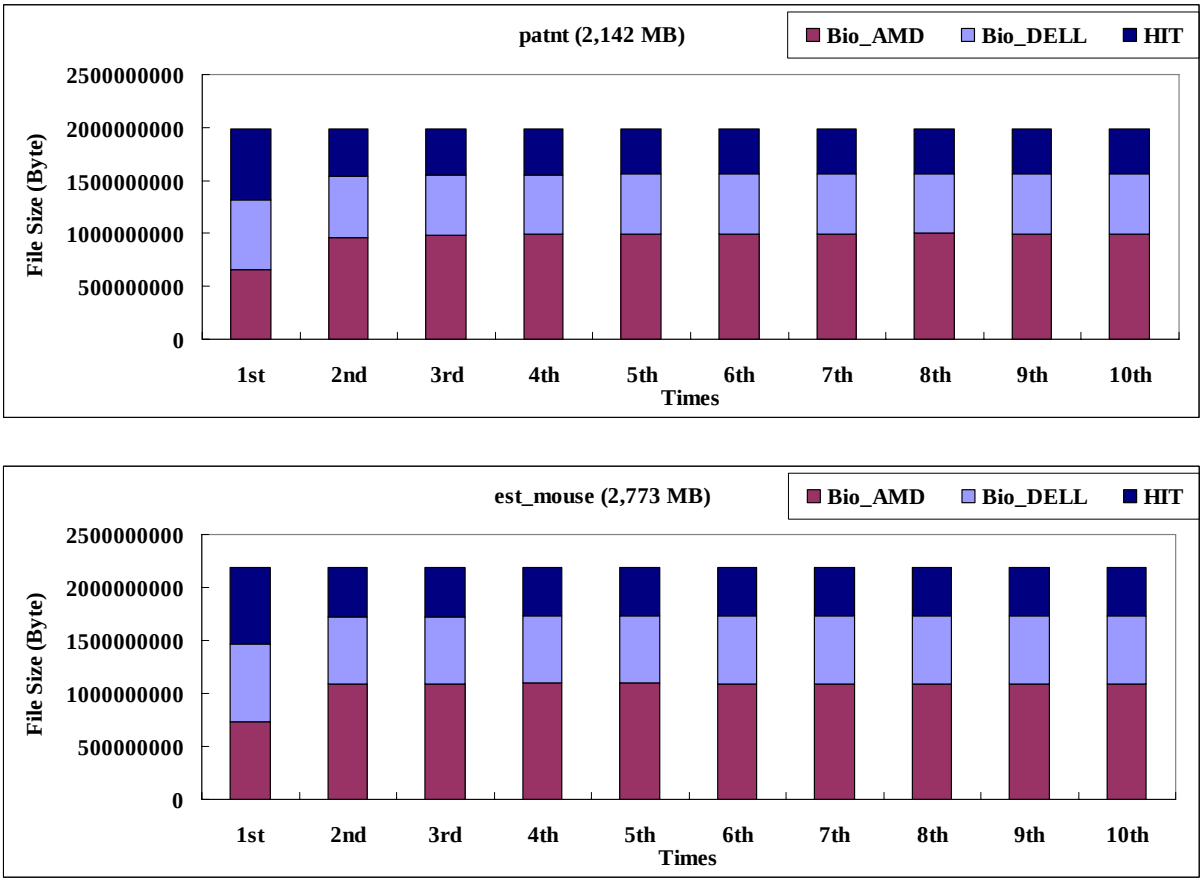


Figure 5-8 The Fragment Size of Each Experiment

5.6 Discussions

Four findings from G-BLAST Service are worth summarizing.

First, G-BLAST segments sequence databases in advance in accordance with the Job Dispatch System and then decides which sites need to run the jobs. Conventional mpiBLAST divides sequence databases into approximately equal fragments using the mpiformatdb command and then assigns the fragments to nodes. However, the sites may have different computation capacities and network speeds, especially in the Grid environment. It perhaps leads to faster sites wait for slower sites to complete their own job. Overall performance is strongly affected by this situation.

Second, G-BLAST does not use the mpiformatdb command. It just segments the sequence databases, which is faster than using mpiformatdb. G-BLAST lets mpiBlast Server executes mpiformatdb and formats its own fragments. This is just like performing mpiformatdb in parallel, and is faster than conventional mpiBLAST. In addition, execution mpiformatdb by mpiBlast server can avoid G-BLAST Server overloading with segmenting sequence databases.

Third, G-BLAST is flexible compared to conventional mpiBLAST. Because each Cluster System is independent, each can easily adjust its environment or performance. Conventional mpiBLAST needs more steps to make such adjustments.

Fourth, G-BLAST mpiBLAST is GUI-operated, rather than command-line-operated, which makes it easier to use. Results are also visible on users' screens allowing one–touch storage.

Chapter 6

Conclusions and Future Work

This paper proposes the Grid-based solution named G-BLAST Framework. It integrates mpiBLAST into a Grid environment. We also report on implementing a Grid Service Portal that enables users to use G-BLAST Framework.

Currently, there are three advantages of this framework. First, the G-BLAST is friendlier than mpiBLAST because we provider a GUI application that enable users to use G-BLAST via this interface. Second, the G-BLAST is more flexible than mpiBLAST because the G-BLAST selects sites dynamically and segments genomic database according to sites performance and history records. Third, the G-BLAST is faster than mpiBLAST because the G-BLAST re-orders the genomic database in advance and performs *mpiformatdb* with "*--skip-reorder*" in parallel.

All the above, the G-BLAST has more efficient than mpiBLAST. Moreover, we will keep improving the Schedule System, Information System and Job Dispatch System strategies to enable better G-BLAST performance in the future.

Bibliography

[1] A Sturn, B Mlecnik, R Pieler, J Rainer, T Truskaller, Z. Trajanoski, "Client-Server Environment for High-Performance Gene Expression Data Analysis," *Bioinformatics*, 2003, 19(6):772-773.

[2] K. Fumikazu, Y. Tomoyuki, F. Akinobu, D. Xavier, S. Kenji, and K. Akihiko, "OBIGrid: A New Computing Platform for Bioinformatics," *Genome Informatics*, 13:484-485, 2002.

[3] S. Gernot, R. Dietmar, and T. Zlatko, "ClusterControl: A Web Interface for Distributing and Monitoring Bioinformatics Applications on a Linux Cluster," *Bioinformatics*, 20(5):805-807, 2004.

[4] R. Buyya, "High Performance cluster Computing: System and Architectures," Vol. 1, *Prentice Hall PTR*, NJ, 1999.

[5] R. Prodan and T. Fahringer, "ZENTURIO: An Experiment Management System for cluster and Grid Computing," *Proceedings of IEEE International Conference on cluster Computing* (*CLUSTER'02*), pp. 9-18, Chicago, Illinois, USA, 2002.

[6] F. Achard, G. Vaysseis, and E. Barillot, "XML, bioinformatics and data integration," *Bioinformatics*, vol. 17 no.2 2001 Pages 115-125.

[7] G. Kandaswamy, L. Fang, Y. Huang, S. Shirasuna, S. Marru, and D. Gannon, “Building web service for scientific grid applications,” *International Business Machines Corporation*, 2006, VOL. 50

[8] J. Andrade, L. Berglund, M. Uhlen, and J. Odebrg, “The use of the Grid technology to solve computationally and data intensive bioinformatics tasks,” *Workshop on state-of-the-art in scientific and parallel computing*, 2006

[9] A. Krishnan, “GridBLAST: a Globus-based high-throughput implementation of BLAST in a Grid computing framework,” *Concurrency and Computation: Practice and Experience*, 2005, 17(13), pp. 1607-1623

[10] C. Oehmen and J. Nieplochs, “ScalaBLAST: A Scalable Implementation of BLAST for High-Performance Data-Intensive Bioinformatics Analysis,” *IEEE Transactions on Parallel and Distributed Systems*, 2006, vol. 17, no. 8, pp. 740-749.

[11] Y. Sun, S. Zhao, H. Yu, G. Gao, and J. Luo, “ABCGrid: Application for Bioinformatics Computing Grid,” *Bioinformatics*, 2007, 23(5): 1175 – 1177

[12] J. Wang, and Q. Mu, “Soap-HT-BLAST: high throughput BLAST based on Web services,” *Bioinformatics*, 2003, 19(14): 1863-1864.

[13] NCBI BLAST, http://130.14.29.110/BLAST/.

[14] mpiBLAST, http://mpiblast.lanl.gov/.

[15] MPI, http://www.lam-mpi.org/.

[16] I. Foster, “The Grid: A New Infrastructure for 21st Century Science,” *Physics Today*, 55(2):42-47, 2002.

[17] I. Foster and C. Kesselman, *The Grid 2: Blueprint for a New Computing Infrastructure* (*Elsevier Series in Grid Computing*), Morgan Kaufmann, 2nd edition, 2003.

[18] GT4, http://www.globus.org/

[19] C. T. Yang, Y. L. Kuo, K. C. Li, and J. L. Gaudiot, “On Design of Cluster and Grid Computing Environments for Bioinformatics Applications,” *Distributed Computing - IWDC 2004: 6th International Workshop*, Lecture Notes in Computer Science, Springer-Verlag, Arunabha Sen, Nabanita Das, Sajal K. Das, et al. (Eds.), Kolkata, India, vol. 3326, pp. 82-87, Dec. 27-30, 2004.

[20] R. Nobrega, J. Barbosa, and P. Monteiro, “BioGrid Application Toolkit: a Grid-based Problem Solving Environment Tool for Biomedical Data Analysis,” *VECPAR*, 2006

[21] P. Bala, J. Pytlinski, M. Nazaruk, V. Alessandrini, D. Girou, D. Erwin, D. Mallmann, J. MacLaren, J. Brooke, and J. Myklebust, “BioGRID - An European Grid for Molecular Biology,” *Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing*, 2002, pp. 412.

[22] C.T. Yang, Y.L. Kuo, and C.L. Lai, “Designing Computing Platform for BioGrid,” *International Journal of Computer Applications in Technology*, 2005, vol. 22, no. 1, pp. 3-13.

[23] WSRF, http://www.globus.org/wsrf/.

[24] I. Foster and C. Kesselman, “Globus: A metacomputing infrastructure toolkit,” *The International Journal of Supercomputer Applications and High Performance Computing*, 11(2):115–128, summer 1997.

[25] N. Karonis, B. Toonen, and I. Foster, “MPICH-G2: A Grid-Enabled Implementation of the Message Passing Interface,” *Journal of Parallel and Distributed Computing* (*JPDC*), Vol. 63, No. 5, pp. 551-563, May 2003.

[26] C. T. Yang, Y. L. Kuo and C. L. Lai, “Design and Implementation of a Computational Grid for Bioinformatics,” *Proceedings of the 2004 IEEE International Conference on e-Technology, e-Commerce and e-Servic*e (*EEE* 04), pp. 448-451, Grand Hotel, Taipei, Taiwan, March 28-31, 2004.

[27] NCBI, http://www.ncbi.nlm.nih.gov/.

[28] http://www.oasis-open.org/

[29] http://www.epm.ornl.gov/pvm