Chapter 4

System Design

This chapter will focuses on parallel implementation design of tasks distribution of Quadratic Sieve algorithm. Class identifications, relationships and their functionalities will be reviewed in this chapter.

4.1 Criteria for Parallel Implementation

Not all applications are suitable to be implemented in parallel way. There are several criteria need to be met before considering a parallel approach. Below are the criteria mentioned:

a. High Computational Complexity – The application must sufficiently complex so that large computational resources are required.

b. Massively Partitioned Problem – The problem domain that the application needs to solve must divisible into relatively small subtasks that can be solved individually and independently.

c. Small Communication/Computation ratio - Typically, parallel computing is able to solve problem quickly than standalone implementation would allow. However, if the time spends on communication (getting a task from server) is significant with respect to the computation time for a given task; then a parallel implementation may actually take longer running time to finish compare to standalone version.
4.2 Space-Based Parallel Application Patterns

Space-based distributed system model provide a more flexible, simple and dynamic manner in designing a distributed application compare to other conventional parallel system such as Message Passing Interface (MPI) [SN11996] and Parallel Virtual Machine (PVM) [GEI1995]. Space-based distributed system concern about flow of objects moving into and out from space. Its design characteristics allow us to apply different patterns of usage to suite any distributed application need to be built.

There are two significant patterns for parallel implementation, they are:

a. Replicated-worker Pattern

b. Command Pattern

Other patterns for non-parallel implementation can be found in [FHA1999]. Patterns above can generally called 'Master-Slave Pattern'. Two of these patterns are slightly different on the way how slave may behave.

In Master-Slave Pattern, there is typically one master process with multiple numbers of slaves. The objective of this pattern is to bring a problem into smaller and independent tasks which then deposits into a space in the form of objects. The master process is in charge of decomposition of big problem into sub-problems by mean of tasks. Slave will get the tasks from space, perform the computation intensive jobs and send back results into the space. Later, master process will collect the results from space and combine them into a meaningful overall solution. There is no direct interaction between master and slave.
The Replicated-Worker Pattern is customized to suite certain problem domain, if the problem domain changed, the slave process's module will need to be replaced. Command Pattern however, apply a more generic approach where master process is able to pass any 'type' of task into space and the slave will carry out the tasks by invoking a well known method(s) without needing to know how the task is to be carried out. Command Pattern is well suited for creating a generic or 'reusable' compute engine that capable to perform any type of computation. Master-Slave Pattern is useful for building parallel application that can speed up solutions to a large scale of compute-intensive problems.

![Diagram of Replicated-Worker Pattern](image)

**Figure 4.1** Replicated-Worker pattern
4.3 Chosen Factorization Algorithm

This thesis will study and implement the general Quadratic Sieve (QS) algorithm.

There are several reasons for choosing this algorithm as our computation problem:

a. It is simple and easy to understand.

b. Efficient for finding prime factors for number above 10 and below 100 digits length. This suite the project scope.

c. The algorithm is highly suitable for parallel execution. This means the algorithm can be separated into smaller parts and executed by different processes.
4.4 QS-Space Architecture

As mentioned in section 4.2, Command Pattern is well suited for building flexible and dynamic Master-Slave model of parallel and distributed system. This pattern is considering the most brilliant approach in making code debugging and deployment easier and time saving. Using this design, the compute-intensive part of the QS algorithm will be coded in a mobile object called Task Entry. The Task Entry carries all the necessary instructions and configuration data, deposited in space and waiting for slave process to pick up. Operations defined in the Task Entry will be performed in slave’s address space.

The whole design of QS-Space consists of two tiers shown in figure 4.3. Master and slave module doesn’t communicate directly but indirectly via space. Both modules are independent from each other.

![Diagram of 2-tier architecture of QS-Space](image)

Figure 4.3 2-tier architecture of QS-Space

Because the core operation of the system is concentrated mainly on the computation intensive part of the Task Entry, therefore, most of the debugging works are estimated to be happening there. Frequent updating of codes will make life harder especially working with a distributed system. Fortunately, the space-based architecture’s shared
provides the way where codes (and modified codes) can be accessed seamlessly by all slaves. Clearly, the manual distribution of modified codes will totally be eliminated with the help of network. Slave processes that resided in remote location will suffer minimum impact of code updating.

Generally there are two ways to design distributed application namely peer-to-peer approach and master/slave approach (Figure 4.4 and 4.5). In peer-to-peer approach, computational cluster consists of computers with equal rights. However, this design suffers from sophisticated communication protocol and hard to build. In this project, master/slave approach will be used.

![Figure 4.4 Peer-to-peer design](image)

55
There are three main components that build up QS-Space. This can be depicted as following:

All these three main components must coexist but operate independently from each other. Absence of one or more components won’t directly bring down the whole system but will affect other components in certain extend.

Since each task entry represents an independent process which will be carried out by a slave, hence, no dependency of other process(s) will happen here. In other word, a process won’t need to wait other process or processes to finish before it can proceed with its operation. However, the system design described in this thesis will involve
certain extend of resource locking which will be fully explained in chapter 5. A more
detailed explanation of sub-components for Master and Slave modules will be
discussed later in the following sections; Space component however, is part of
reference implementation from Sun Microsystems and will be discussed in next
chapter.

4.4.1 System Operational Analysis

To start a factorization process, the user must feed master component with a large
composite number, $N$. This number should not exceeding 60 digits in length. Pre-
examination must performed by the system for compositeness of a number before
acceptance for subsequent operations. User can however, setup a list of numbers to be
factored and ask the system to perform consecutively. Other than that, user is allows
to determine and limit the number of slaves that will participate in a compute session.
However, if user doesn’t want to limit the number of slaves then the system is allows
utilizing as many slaves it can find in the network.

When the system is properly configured, a database connection to a local/remote
database system will be established next. This database system will maintain any vital
information obtained or generated during an operation session. The data kept in the
database may also help the system to resume the process from its last point of failure.

The system is capable of performing self initialization of all parameters needed for a
particular operation session. This includes composite number, $N$, related parameters
such as factor base (FB) size and sieving interval of QS algorithm. These two values
must be the optimum value for best result and will explain in detail in next chapter.
After system initializing, the system is now ready to move into the QS algorithm execution stage where two main operational threads will be created and run concurrently:

a. The task entry generator thread is responsible for creating and control the amount of task entries written into space.

b. Result entry collector thread responsible for taking any result entry out from space that returned from slaves as the result of computation. The result entry is examined and information inside is retrieved and saved into database system.

When enough data has been collected (e.g. >FB+30), two of the live threads mentioned above will be terminated and system will start perform final calculation of QS algorithm (Gauss elimination for solving linear system) to retrieve two prime factors of $N$.

Total parallel sieving time, Gauss elimination time and other computation time will be displayed on the system console. Finally, result will be stored in database, system log will be saved and an email notification of result will be sent to user. Figure 4.7 shows the interaction between components.

4.4.2 Data Type

There are several data types that are significant and important to the system operation which worth mentioned here. They are,

a. Big Number

The system need to process an unusual long number where common/primitive data type is not able to handle (e.g. integer, double and long).
b. Entry

Communication between master and slave is through exchanging of object via space called *Entry*. Base on the *Command Pattern* definition, an entry contains attributes and functions that capable of performing a meaningful calculation independently without much supervision from a slave.

![Sequence diagram for overall system process](image)

Figure 4.7 Sequence diagram for overall system process

In contrary to above definition, an entry also acts as messenger that carries information to all the receivers who need it. It doesn’t carrying executable instructions but only attributes.

### 4.4.3 Classes Identification and Design

After examined the use case scenarios, system requirements and capabilities discussed in chapter 3, an in-depth study of object-oriented modeling of QS-Space system has
been conducted. Figure 4.8 illustrates the identified classes, relationships and functionalities which build up the system. Detailed technical explanation of the functional components and interconnected relationship within components will be fully discussed in chapter 5. Below is the description for each class contained in Figure 4.8. Relationship between classes describe in this section involve the understanding of object-oriented programming language in the context of Java.

a. **QSMasterPanelWindow**

   A GUI class. It provides necessary graphical components to let user interacts with the underneath functional components.

b. **QSMaster** extends **QSMasterPanelWindow**

   This is the master module of QS-Space application. Its main responsibility is to coordinate the compute server. Following listed several primary responsibilities:

   i. Direct interaction with user actions and collect all the information needed from user (e.g. composite number to be factored, number of slave required, database connection setup details, administrator/user’s email address and etc).

   ii. Execute the QS algorithm operations – decompose a factorization problem into subtasks that can be distributed to Slave for computation. Collect back subtask results and combine to calculate for answer- two prime factors. Sub-tasks distribution also involves the coordination of number of Slaves.

   iii. Maintain persistent data storage – Configuration details and system state need to be maintained in the form of persistent storage. This feature will enable future runs of system to be preserved.

   iv. Control remote access of Master system.
v. Provide remote notification of result via e-mail.

vi. Supervise consecutive launch of system runs. A single completed system run is considered as solving one factorization problem.

There are three main classes that provide QSMaster a basic implementation of core operations; they are NodesCoordinator, TaskGenerator and ResultCollector. QSMaster will create these three instances of thread if there is only one composite number to be factored, otherwise, ConsecutiveBatch will take the responsibility.

c. ConsecutiveBatch

Responsible for coordinating and executing a successive number of system runs. It can control the creation and destruction of instances for three main classes mentioned above.

d. NodesCoordinator

Responsible for discovering and gathering/recruiting enough number of slaves as required before a system run, monitoring changes and maintaining number of slaves along the system run. Major involvement in self-healing mechanism number of slaves in any situation where insufficient number of slaves may occur (where slave process may be terminated) and trying to help the system return to its previous state by recruiting any available free slave processes attached to the network.
c. **TaskGenerator**

Task entries (QSTask objects) are generated and control by this class. Each task entry created by TaskGenerator will be assigned with the proper attributes and deposited inside a space by the TaskGenerator. Total amount of entries in the space should be maintained at a minimum level in order to avoid exhausting of task entries. This design will be able to avoid the delay or extra time needed in during task entries distribution operation.

f. **ResultCollector**

Responsible for detecting the existence of result entries (QSResult objects) in a space generated by slaves and takes them out from the space. When the amount of result entries collected exceeding the number desired, TaskGenerator will be notified and generation of task entries will be terminated.

g. **EmailGenerator**

The result of each system run can be fetched to user’s email account with the help of this class.

h. **QSTask**

A special entry that designed to carry self-executable instructions which will perform the heavy part of the QS algorithm – Sieving; in order to find any B-Smooth numbers (refer chapter 2). The slave module will take this entry out from the space into its address space and calling the execute() method of QSTask. QSResult objects will be returned as the result of execution. QSTask also carries
certain configuration data that are usable by QSSlave in order perform the task
efficiently.

i. QSSResult

This entry contains only attributes that hold the result of each task performed. It
also holds the system operational properties such as sieving time, slave’s
operating system architecture data, and IP address. No executable instructions
contained.

j. SpaceAccessor

Master and slave module will need to consult with this class in order to
communicate with space server. SpaceAccessor is an integration of processes for
JINI’s discovery and JINI’s lookup for space and transaction manager service
item.

k. MathUtils and MatrixOperator

These two classes provide all the necessary mathematical methods for master
module and slave module. Slave module will only require MathUtils class.

MathUtils contains methods for performing BigInteger calculation of square root,
legendre symbol calculation and primality testing. MatrixOperator is responsible
for performing Gaussian elimination of all the B-Smooth numbers returned by
slaves. By performing Gaussian elimination, relations that produce a perfect
square will be obtained.
1. **MasterRemoteController**

Provide gateway for remote user to communicate with the master module. It is responsible for administrating access control and data access.

m. **QSSlave**

This is the slave module of QS-Space application. The **QSSlave** processes are deployed in different PCs with network connectivity. Each process can be positioned at different network topology and only one **QSSlave** module is to be installed in each PC.

With the help of **SpaceAccessor**, **QSSlave** retrieves a task entry from the space and call the `execute()` method. Any **QsResult** object returned from the method call will be written back to space waiting for pickup by master module.

### 4.5 User Interface Design

#### 4.5.1 GUI for Master Module

The Graphical User Interface (GUI) for the master module must be able to provide crucial information and accurate statistical data that reflect the system activities.

There are two distinct types of information – input (from user to system) and output (from system for user). The inputs should be entered in some sort of tabular form or text boxes. The form must able to describe itself and lead the user to key-in all the information required by the system. Each data entered must undergo some verification mechanism to ensure the correctness of value. Confirmation of each action given is essential to prevent from user's mistake and unwanted interruption.
The system should allow system's messages, progression status and events to be displayed on the GUI and each message should start by a time stamp. The messages must also include the result of factorization process where two prime numbers will be obtained. All messages captured by each system run shall be logged in a file and store in hard disk.

Generally, the GUI is not for the novice user because it is a special purpose application designed for solving a complex mathematical problem. Therefore, user should undergo some training and gain some knowledge before able to operate the system.

Figure 4.9 depicts the GUI components of master module and Figure 4.10 illustrates the design of main user interface for master module coded in QSMasterPanelWindow class.

Figure 4.9 Master module GUI class diagram
The **QSMasterPanelWindow** represents the main GUI (or control panel) for master module as shown in Figure 4.10. There are five other components that handle user inputs or perform graft plotting. The ability of producing visual output of system data is essentially important for instant analysis to user. Figure 4.17 and 4.18 show the dialog boxes of real-time graphs for system operational information. **ParallelismChart** is a useful GUI component that allows user to analysis the parallelism sieving operation. Each vertical colored bar represents the total sieving time for each task performed by a slave.

By the way, a remote access to **QSMaster** also provided by a remote GUI class called **RemoteQS**. This GUI is actually a Java Applet which embedded in a HTML file hosted By a HTTP server. **RemoteQS** provides all the necessary functionalities to access **QSMaster** processes remotely but in a limited fashion (see Figure 4.19).
Figure 4.10  GUI for QSMaster control panel

1. A unique number for each system run
2. Input box for number to be factorized
3. Generate a new composite number
4. Clear the input box contents
5. Watermarking level value for task entries
6. Input box for factor base size
7. Input box for sieving interval
8. Check this box to initialize remote access
9. Clear all the information in database
10. Clean all incomplete system run data
11. Total memory currently available
12. Indicator for watermarking
13. Total running time
14. Stop button- terminate the operation
15. Start button- start new factorization process
16. Icon indicates that GUI interaction has been locked
17. Display current queuing process batch
18. Percent of completion
19. Total number of slaves currently participated
20. Show details for each slave
21. Discover and get response from accessible slaves in network
22. Command slave to enter standby mode
23. Command slave to terminate its program
24. Input box for total slave required
25. Statistic data in QSMaster
26. Statistic data in Space
27. Statistical data for entries
28. Display box for system messages and events
Figure 4.11 Database connectivity setup (Class: DBConnDialog)

Figure 4.12 Input box to generate a new composite number

Figure 4.13 QS-Space system configurations (Class: SysConfDialog)
Figure 4.14 Confirmation box for consecutive batch

Figure 4.15 Input form for consecutive batch composite numbers
(Class: ConBatchDialog)
Figure 4.16 Result of discovered accessible slave nodes in the network

Figure 4.17 System performance graft (Class: graftDisplay)
Figure 4.18 Parallelism chart for sieving processes (Class:ParallelismChart)
**Notes: Each colored vertical bar represents sieving time executed by a slave
Remote QS Control/Monitoring Panel

Welcome guest! Quadratic Sieve (QS) is a common method used to factor huge numbers. However, factoring large numbers with the length more than 50 digits is truly uneasy and time consuming no matter what method you are using. Therefore, distributed and parallel computing were came into picture in order to share resources to carry a common task.

Here, you are given an opportunity for taking part in my project by downloading the Remote JS26 (Java) below andrun it on your machine. The background technology is based on Sun Microsystems's JavaSpace™. Click here to go to the download page.

This Java Applet on the left is a remote controlling monitoring panel which allow user to manipulates the QS server remotely. However, visitors are not give to watch the activities of the server by login as GUEST. Please view the diagram on the right for setting up your Java Applet needed for remote code.

Figure 4.19 Remote access through Java Applet
4.5.2 GUI for Slave Module

Slave module is designed to provide no GUI component. However, system information will be displayed in the operating system console window where slave module deployed.

Figure 4.20 Slave’s activities console

4.6 Database Design

There are four tables needed to support the system, they are:

a. Batch

Holds data and status for each factorization process.

b. SieveTime

Holds the sieving times for each subtask that performed by the slaves in a single batch operation.
c. **B_Smooth_Returned**

Keep all the smooth numbers returned from slaves.

d. **SlaveInfo**

Store information about the slave

Among four tables described above, table A, B and C are contributing in restoring previously incomplete process. Tables below show the structure of each database table mentioned above.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BATCH_NO</td>
<td>varchar(20)</td>
<td>System run identification number</td>
</tr>
<tr>
<td>DigitLength</td>
<td>int(10)</td>
<td>Digit length of N</td>
</tr>
<tr>
<td>N</td>
<td>varchar(150)</td>
<td>The number to be factorized</td>
</tr>
<tr>
<td>P</td>
<td>varchar(75)</td>
<td>Prime factor obtained</td>
</tr>
<tr>
<td>Q</td>
<td>varchar(75)</td>
<td>Prime factor obtained</td>
</tr>
<tr>
<td>SIEVE_TIME</td>
<td>decimal(12,3)</td>
<td>Total sieving time</td>
</tr>
<tr>
<td>GAUSS_TIME</td>
<td>decimal(12,3)</td>
<td>Total Gaussian elimination time</td>
</tr>
<tr>
<td>OTHER_TIME</td>
<td>decimal(12,3)</td>
<td>Other computation time</td>
</tr>
<tr>
<td>AVR_RATE</td>
<td>int(11)</td>
<td>Average smooth relations found per minute</td>
</tr>
<tr>
<td>FACTORBASE</td>
<td>int(11)</td>
<td>Factor base assigned</td>
</tr>
<tr>
<td>TOTALSIEVE</td>
<td>int(11)</td>
<td>Sieving interval size assigned</td>
</tr>
<tr>
<td>TotalSlave</td>
<td>int(11)</td>
<td>Number of slave process involved</td>
</tr>
<tr>
<td>DATE</td>
<td>date</td>
<td>Date of the system execution</td>
</tr>
<tr>
<td>TIME</td>
<td>time</td>
<td>Time of the system execution</td>
</tr>
<tr>
<td>STATUS</td>
<td>varchar(10)</td>
<td>Indicator of system state. Acceptable value are INITIAL, PROGRESS and COMPLETED</td>
</tr>
</tbody>
</table>

Table 4.1 **Batch Table**
Table 4.2 SieveTime table

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BATCH_NO</td>
<td>varchar(20)</td>
<td>Refer Table 4.1</td>
</tr>
<tr>
<td>INIT_OFFSET</td>
<td>bigint(20)</td>
<td>Sieving interval offset</td>
</tr>
<tr>
<td>SIEVE_TIME</td>
<td>decimal(12,3)</td>
<td>Time taken to complete finding smooth number within a given interval</td>
</tr>
</tbody>
</table>

Table 4.3 B Smooth_Returned table

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BATCH_NO</td>
<td>varchar(20)</td>
<td>Refer Table 5.1</td>
</tr>
<tr>
<td>DATE_RECEIVED</td>
<td>date</td>
<td>Date result received</td>
</tr>
<tr>
<td>TIME_RECEIVED</td>
<td>time</td>
<td>Time result received</td>
</tr>
<tr>
<td>SMOOTH_X</td>
<td>varchar(30)</td>
<td>Smooth number returned</td>
</tr>
<tr>
<td>INIT_OFFSET</td>
<td>bigint(20)</td>
<td>Sieving interval offset</td>
</tr>
<tr>
<td>IP_ADDRESS</td>
<td>varchar(10)</td>
<td>IP address of the slave that return the result</td>
</tr>
</tbody>
</table>

Table 4.4 SlaveInfo table

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IP_ADDRESS</td>
<td>varchar(15)</td>
<td>IP address of the slave</td>
</tr>
<tr>
<td>BATCH_NO</td>
<td>varchar(20)</td>
<td>Refer Table 4.1</td>
</tr>
<tr>
<td>HOST_NAME</td>
<td>varchar(15)</td>
<td>Host/Computer name assigned to the slave</td>
</tr>
<tr>
<td>OS_NAME</td>
<td>varchar(20)</td>
<td>Operating system name</td>
</tr>
<tr>
<td>OS_ARCH</td>
<td>varchar(10)</td>
<td>Operation system architecture</td>
</tr>
<tr>
<td>OS_VER</td>
<td>varchar(10)</td>
<td>Operation system released version</td>
</tr>
<tr>
<td>JVM_VER</td>
<td>varchar(15)</td>
<td>Java virtual machine version</td>
</tr>
<tr>
<td>DATE_JOINED</td>
<td>date</td>
<td>Date participated</td>
</tr>
<tr>
<td>TIME_JOINED</td>
<td>time</td>
<td>Time participated</td>
</tr>
</tbody>
</table>

4.7 Chapter Summary

This chapter focuses on identification of entities involved in QS-Space design where detailed system requirements have been discussed in chapter 3. Realization of conceptual design can easily be implemented as classes, functionalities and their relationships had been defined using object-oriented methodology. Two main patterns are involved in building space-based parallel application; they are Replicated-worker Pattern and Command Pattern. These two patterns use the same communication
protocol but using different approach for sending commands to slave.

Chapter 5 will discuss in-depth of how QS-Space is to be implemented and controlling of entries happen in space. A space-based parallel technology called JavaSpaces provided by JINI technology software from Sun Microsystems will be used to implement QS-Space application.