JAVA DYNAMIC CLUSTERING PLATFORM

By

Muhammad Ali Khan

(2000-NUST-BIT-817)

A project report submitted in partial fulfillment of

the requirement for the degree of

Bachelors in Information Technology

In

NUST Institute of Information Technology

National University of Sciences and Technology

Rawalpindi, Pakistan

(2004)

DEDICATED TO MY PARENTS

CERTIFICATE

It is certified that the contents and form of the project report entitled "Java Dynamic Clustering Platform" submitted by Muhammad Ali Khan has been found satisfactory of the requirement of the Bachelors in IT Degree at NUST Institute of Information Technology.

COMMITTEE

1. Advisor sign _____

   Mr. Saqib Mir

2. Co-Advisor sign _____

   Mr. Shahzad Khan

3. Committee Members

   Mr. Zaheer Abbas sign _____

   Mr. Ahsan Ahmed sign _____

ACKNOWLEDGEMENTS

Praise be to Allah the all powerful. We are mere creatures of Allah and act in the capacity that he has given us. Our mind and our soul are a reflection of Allah.

I would like to thank my advisor, Mr. Saqib Mir who has guided me through the course of the project. I was almost at a dead end with this project before I met him and he has shown me many new dimensions to explore.

I would like to thank my family members especially my parents who have given me the chance to be the man that I want to be and to pursue the paths that I have chosen for myself. I have not done what they wanted me to but still they have always supported me and have been with me in my ups and downs.

I would also like to thank my friends who have given me strength and confidence in my abilities and have encouraged me to work on this project for such a long time.

Table of Contents

LIST of figures

LIST OF TABLES

ABSTRACT

JDCP project aims at developing a platform for creating high performance computing clusters. It aims at exploiting the existing hardware resources such as PCs and normal networking and combines them to create a single larger virtual processing machine. The initial distributed computing system was available before the start of the project. This basic platform enabled simplest distributed computing. During the course of the project new features were added to this existing system. The features added include cluster monitoring, adaptive real-time scheduling, information services, fault tolerance, batch job support and certain optimizations to reduce network data transfers. Detailed discussion on these features constitutes a major part of this report.

Firstly the report gives a brief account of the initial system. Afterwards a comparison study is given that compares the JDCP system to existing clustering systems available and discusses how JDCP differs in many aspects of a distributed computing system. Once the significance of JDCP is explained, the report discusses in detail the features added into the system during the course of the project. This discussion not only includes the work done but also includes the need and benefits of these features. In the end of the report testing results and a brief discussion on them is given and on the basis of these results a concluding discussion is done on the whole project.

Chapter 1

PROJECT DESCRIPTION

1.1 INTRODUCTION

The project aims at developing a platform for creating high performance computing clusters. The platform can be used to create single-site and multi-site clusters and will be capable of combining computing resources of heterogeneous systems present on LANs and the Internet to create a larger processing machine capable of providing real-time services to clients.

Servers act as a bridge between the worker and the client machines. The worker machines run the worker software that connects to the server and lends its resources. Clients to the clusters will also be programs rather than users and the programs will send processing requests to the server during their execution. Hence the programs will divide themselves to run in parallel on the worker machines of the cluster.

In addition to this simple distributed processing the system will also provide features like cluster monitoring by the server, adaptive real time scheduling, security, information services and fault tolerance. A very simple distributed computing system is available before the start of the project and the features listed above will be added during the course of the project.

The whole system is developed in Java from scratch using only the core Java technologies like sockets, threads and JNI. Hence the system is portable to any

platform of course with restricted features. Fully featured version is available to Windows 2000 onwards and Linux 9.0.

1.2 SIGNIFICANCE

The platform will provide new opportunities of research in distributed computing. The system differs from existing solutions mainly through a different programming model and the use of Java technology. Currently all major high performance clustering systems are for high performance C and other older languages thus by leveraging the Java technology a new area of research can be opened which has its benefits for example support for object orientation, simplicity of installation/usage and security options. Platform independence also allows the system to be deployed in any working environment with minimum effort and disturbance. Existing solutions require operating systems of their choice and also a lot of configuration which is not required by JDCP. The following discussion elaborates the above mentioned points.

1.2.1 New Programming Model

The JDCP platform provides a new programming model in which the client machine is constantly connected to the server and submits jobs, processes and gets the results all through the API provided with the system. The clients in all major clustering systems are users not programs. The user submits programs to be processed remotely and gets the results through files produced by these programs but in JDCP the program is the client and it submits processing requests and the gets the results back thus

making asynchronous remote procedure calls. The following table (Table 1) compares the two approaches.

Table 1 Comparison of JDCP and traditional programming models

Traditional Approach	*JDCP Approach*
A single program runs at the speed of one processing machine	A single program runs on all the cluster thus running faster than it would on a single machine
Programs submitted as jobs so its easy to run existing application on the cluster	Procedures submitted as jobs. Existing programs have to be modified to run on the cluster
Parallelism is achieved through submitting multiple programs	As single program runs in parallel on the worker machines
Jobs have to be faceless because they are separate programs and results are received as files	Complete GUI based programs can run on the cluster much faster than they would on a single machine
Batch jobs may be processed late on the cluster so once submitted a user should forget about the job for sometime	Real time jobs are processed and returned to the client program as soon as possible so interactive programs run best on a JDCP cluster.

1.2.2 Pure Java Implementation

The system is completely made in Java except for a few platform dependant native implementations for collection system metrics. This makes it possible for the

cluster made by this platform to have both Windows and Linux machines running at the same time as workers for the cluster. The Java language allows exchange of objects and other data across machines having different operating systems. This feature is particularly useful when creating clusters from existing networked environments such as a university LAN.

1.2.3 Completely Object Oriented

The system accepts complete object oriented jobs. This means that the functions are called on objects; objects can be passed as arguments to the function and also returned as results. Static methods can also be called. This is particularly useful when converting existing applications to run on the new platform. This feature also makes programming easier because programmers are usually more accustomed to OOP nowadays.

1.2.4 Real-time Callbacks

The jobs submitted to the system are actually functions called remotely hence the same methods can be called locally and remotely which makes it much easier to convert existing application to run on the platform. This also allows the application to execute both locally and remotely at the same time.

1.2.5 Simple to Use

The technology is very easy to use. There is no installation requirement any component of the system. You simply copy the application in any folder, set the Java virtual machine paths and it starts working. Additionally the programming model used by the system is also very easy. The client API to access the system consists of no more then 10 functions. An average programmer can learn and use the technology

within an hour and start making programs that can fully exploit the resources of the cluster.

1.2.6 Security Restrictions on Code

By leveraging the Java technology the system offers complete security. The Java Sandbox technology allows the worker to run the client jobs inside security restrictions. This means that any malicious client program will not be able to harm the worker machine as it runs inside the Java Sandbox.

1.3 SIMILAR TECHNOLOGIES

1.3.1 Sun N1 Grid Engine

N1 Grid Engine 6 enables enterprises to build grids that make them more productive. Enterprises can monitor and select the optimal usage of computer resources on most commercial operating systems and platforms. It has a new feature ARCo (Accounting and Reporting Console) that feeds accounting information into a relational external database [1].

But the product is not free and has to be licensed from Sun Microsystems. Moreover the system again differs from JDCP because the jobs are native and not remote procedure calls as in JDCP. Security is also not addressed because the main aim of N1 Grid Engine is to allow computers from a single organization to work in collaboration hence only accounting is done while in JDCP complete security is implemented because the clients are not considered a part of the cluster organization.

1.3.2 Globus

The Globus software toolkit facilitates the creation of usable Grids, enabling high-speed coupling of people, computers, databases, and instruments. With Globus, you can run your gigabyte-per-time-step dataset job on two or more high-performance machines at the same time, even though the machines might be located far apart and owned by different organizations [2].

1.3.3 XtremWeb

Like the other Distributed System Platforms, an XW platform uses a) remote resources (PCs, workstations, servers) connected to Internet or b) a pool of resources (PCs, workstations, servers) inside a LAN. Participants of an XW platform cooperate by providing their CPU idle time. XtremWeb belongs to the so called Cycle Stealing Environment family [3].

XtremWeb differs from JDCP because the jobs submitted to XtremWeb are batch jobs that have been written in native languages while jobs in JDCP are actually remote procedure calls and are implemented in pure Java.

1.3.4 NetSolve

NetSolve/GridSolve is a project that aims to bring together disparate computational resources connected by computer networks. It is a RPC based client/agent/server system that allows one to remotely access both hardware and software components. The purpose of GridSolve is to create the middleware necessary to provide a seamless bridge between the simple, standard programming interfaces and desktop Scientific Computing Environments (SCEs) that dominate the work of computational scientists [4].

Again NetSolve differs from JDCP because of the jobs submitted to the system. NetSolve accepts jobs in C, FORTRAN and MATLAB while JDCP accepts jobs in Java. Moreover NetSolve does not include security in its infrastructure while JDCP does so because of the use of Java technology.

1.4 TECHNOLOGIES USED

1.4.1 JNI (Java Native Interface)

JNI is an integral part of the Java language. The JNI technology allows Java applications to call functions implemented low level languages native to the platform on which the Java application is running. This is essential for Java because the Java application runs in a virtual machine and in order to access local resources such as files etc native methods have to be called which contact the operating system and get the resources. JNI works by loading dynamic link libraries from the local system and once the library is loaded methods present in it can be called by the application that loaded the library. Once a native method is called control is shifted to the native code and all the restrictions put by the virtual machine are gone. The native method runs as any other native implementation with full control of local resources.

1.4.2 PDH Library

Performance data helper library is provided by Microsoft. It helps developers in extracting system metrics information. The library provides extensive information about all aspects of a Windows machine including CPU usage, process information cache commit charge etc. The PDH library provides a snap shot of information and it does not calculate statistical information so for information collected over a period of

time the application must take multiple snap shots through the PDH library and then calculate the statistics. The PDH library also provides helper functions to calculate these statistics but they have to be collected by the program and provided to the library.

1.4.3 Linux OS Metrics

The Linux operating system collects, calculates and stores many system information related metrics. These are collected for the use of the operating system and also for applications. The /proc directory is a memory resident directory that contains all these metrics in addition to all the process data. To access this data the files in the /proc directory are opened by the programs and textual information in these files is retrieved. The /proc/cpuinfo file contains the CPU related static information that is processor speed.

The /proc/stat file contains information about the current usage of resources including the CPU usage. Data from these files is extracted by the Java program by opening these files.

1.4.4 JCE (Bouncy Castle Implementation)

The Java Cryptography Extension (JCE) is now a core part of Java SDK 1.4. Basically, it's a set of packages that provide a framework and implementations for encryption, key generation and agreement, and Message Authentication Code (MAC) algorithms [5]. A provider is the underlying implementation of a particular security mechanism. There are several providers, some of which are freely available and others that are quite costly. Companies that offer providers include IBM, Bouncy Castle, and

RSA. I have used Bouncy Castle provider for use in JDCP because it is freely available.

1.5 STARTING SYSTEM

The project started with a simple distributed computing system already working and this system was to be enhanced during the course of the project. Here I present a brief description of the initial system because it is necessary for understanding the features added during the project.

1.5.1 Deployment View

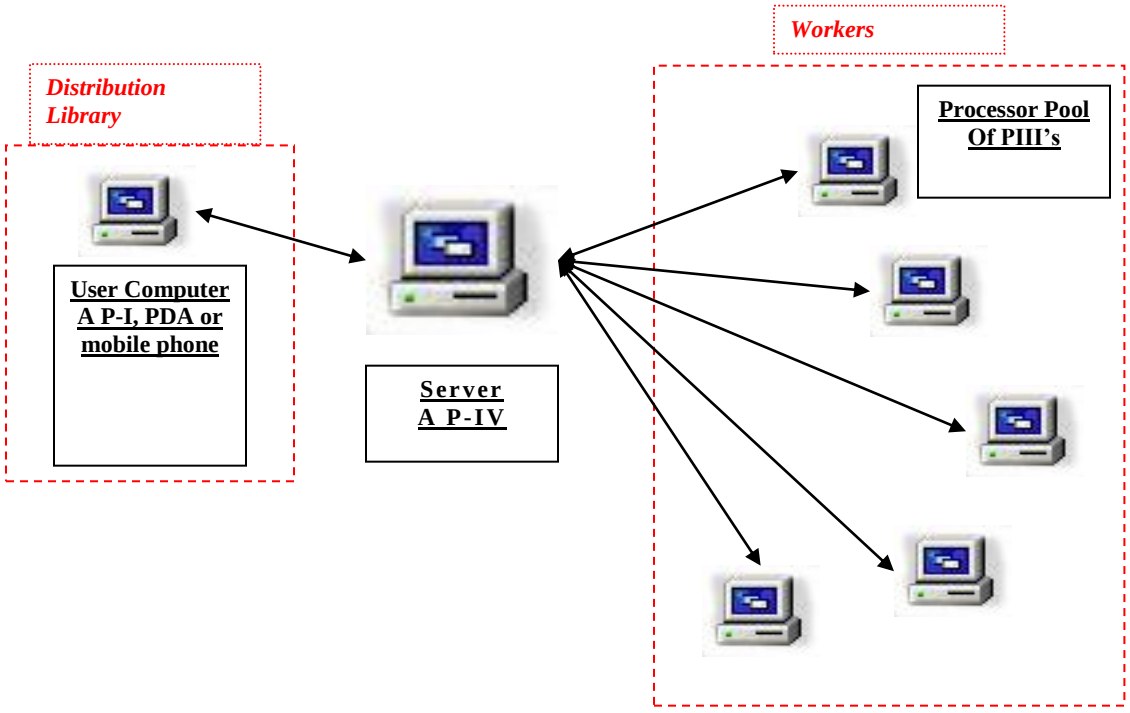


Figure 1.1 Deployment view of the initial System

1.5.2 System Components

Three major entities as shown in the deployment diagram work together to enable the distributed computing system to function. A brief description of these entities is as follows.

1.5.2.1 Distribution library

The distribution library is the only thing that interacts with the user. It is through the library function calls that the user can submit a job and extract the results. Then the user gives the required class to the library so that the library can transfer this class to the server which in turn can pass it to the worker on which actual processing will take place. After passing the class the user can call any function of that class through the library. Now the library itself is responsible for communication between the server and itself. One or more distribution handlers can run on each user machine.

1.5.2.2 Distribution server

The distribution server interacts with the distribution library as well as the worker. It acts as the bridge between the two entities. The distribution logic and command is concentrated in the distribution server. Running on a powerful machine the server is the center of all data transfer. Only one distribution server runs on one designated machine.

1.5.2.3 Worker

It performs the actual work. Executing functions on the processor of a machine in the processor pool. It performs the tasks it is assigned and returns the results to the distributions server which in turn passes these to the distribution library. The worker must be provided with the class whose function it is supposed to execute. Once the

worker has the class it can surely call the function of that class through the java virtual machine. One worker runs on each machine in the processor pool.

1.5.3 Working of the Starting System

- The user imports our distribution library classes
- User initializes the distribution handlers in the distribution library so that they can handle all the communication with the server including all the logic for transferring the requests, getting the results back from the server, storing them for future reference, handle failures in communication and any other means to hide the complexities from the user program.
- User program calls a method in the distribution library to pass the class information to the distribution handler which in turn transfers this information to the server so that it can use this information to process the future user requests.
- User program calls a method in the distribution library to execute a particular function present in the class previously passed to the distribution handler.
- The library passes the information to the distribution server which adds the request in its execution queue.
- When the turn of this request comes the distribution server will pass the order of execution to a work horse and will also pass the class info required for execution.
- The worker will actually execute the function and will return the result to the distribution server.

- The results returned will be stored in the result dump by the server and will be passed to the library and hence to the distribution handler which called the function.
- The Library stores the results in its result dump.
- The user can extract these results from the library's result dump by calling the appropriate method provided in the library through the distribution handler.

1.6 FEATURES ADDED DURING THE PROJECT

As mentioned in the introduction the project aims to add features to the existing basic platform. Here is a list of the features that were added during the course of the project. Detailed discussion on these features is done next in the report.

Cluster resource monitoring

- A total of 14 metrics as yet
- Input to the scheduling process
- Will improve fault tolerance

Adaptive real-time scheduling

- Support for a number of different scheduling policies
- Will adapt to changing number of workers

Information services

- Servers can advertise their resources to information servers
- Clients get server information and decide

Job dependency optimizations

- Pre-defined job dependencies using DAGs (Directed Acyclic Graphs)

- Aimed at reducing number and size of network transfers

Security

- Authentication for clients and workers using passwords and digital signatures
- Encryption of network messages at various levels of security
- Code restrictions at workers using Java Sandbox

Fault Tolerance

- Worker crash recovery
- Server crash recovery
- Network problems (Timeouts, dropped connections, message corruption etc.)

Batch support

- Batch job creation and submission
- Batch job result storage and transfer on demand
- Resource bundling e.g. files needed by the batched jobs

Development of sample applications for the platform

Chapter 2

CLUSTER MONITORING

2.1 INTRODUCTION

Cluster monitoring features aim at making the server more aware of the resources connected to it that is the worker machines. Cluster monitoring includes server side work which comprises of collecting and storing metrics which are mentioned later. The worker side work includes collection of machine-specific data such as CPU speed and usage at periodic intervals. This is done through the help of the operating system. On a Linux platform a pure Java implementation works but this implementation is dependant on the files produced by the Linux operating system so this implementation also becomes platform dependant. On a windows platform the PDH library and win23 functions are used which of course are platform dependant. Hence the worker side on both platforms is platform dependant.

Once these metrics have been collected the server stores them in data structures so that they can be used in the future that is when creating scheduling information to be passed to the schedulers. This information is necessary for the scheduling process because unless the scheduler knows what resources are connected to it, it surely cannot create good schedules that use these resources to the maximum.

2.2 BENIFITS

2.2.1 Input to Scheduling Process

Unless the server knows what resources are connected to it certainly it cannot create schedules that utilize these resources to the maximum. Hence the cluster monitoring metrics go as input to the next step, the scheduler development. The metrics are chosen especially to be the input to the schedulers but they are not dependant on the scheduler currently working. The metrics are independent of the scheduler and are maintained by the server even if the current scheduler does not want the metrics. This is because the schedulers in the JDCP system are not pre-configured and they can be changed at runtime so that system has to keep the metrics in case the next scheduler would want these metrics.

2.2.2 Server Side GUI Allows Up-to-Date Monitoring

A GUI at the server side allows the cluster administrator to see all the information about the workers currently connected and disconnected from the server. This GUI allows the administrator to identify problems with worker machines.

2.2.3 Profiling is Possible

Profilers are not currently supported by the system but as all the information is already at the server profiling of this information is very easy.

2.2.4 Better Fault Tolerance

Fault tolerance is improved not only the administrative information displayed on the GUI but also because of periodic messages sent by the system in order to get the

updated information. This helps in identifying the machines that have crashed as these will not respond to the information update request.

2.3 DEVELOPMENT

2.3.1 Worker State Maintenance

The system maintains a state for each connected worker. The states maintained are connecting, connected, disconnected and removed. Following table gives a description of each of these states

Table 2 Client states at server

State	*Description*
Connecting	At this state the worker has started a connection to the server but the connection is not complete. After the initial connection messages are exchanged the worker changes into a connected worker. At the connecting stage the worker is given no jobs.
Connected	At this stage the worker is fully connected to the server and the server assigns it the jobs to be processed
Disconnected	This is that state when the server/worker connection has dropped but the server has yet not recovered the work assigned to the worker. No jobs are assigned to the worker in this state.
Removed	This is the stage when the server has removed all the jobs assigned to the worker and has rescheduled them to the

	remaining workers. This worker is not given any jobs at this state

2.3.2 Metrics to be Collected

First task in cluster monitoring was to determine the metrics to be collected. As the metrics go as input to the schedulers the scheduling algorithm dictates this decision. Metrics collected at the worker side are transferred to the server so that the server has all the metrics which it stores for future use by the schedulers.

2.3.2.1 Worker side metrics

The worker collects information according to the operating system it is running on. So this part includes contacting the operating system to get the system resource specific information.

- CPU speed
- CPU Usage
- Data transfer rate

2.3.2.2 Server side metrics

The server metrics are not dependant on the platform and all of these are collected statistically or by other mechanisms explained later but not through any platform dependant manner

- Data reception rate of each worker
- Average processing time of each job
- Average processing time of all jobs by each client

- Average result size of each job
- Average result size of all jobs submitted by each client
- Network latency to each worker
- Expected processing time of jobs already assigned to workers
- Transfer size of each job that is to be scheduled

2.4 COLLECTION MECHANISMS

The server sends periodic messages to the workers to get their system information and send it back to the server which in turn stores it in its data structures for further use. The request to get the information is done by the server the worker only replies back after getting the information from the system. The above mentioned mechanism is explained in the following diagram.

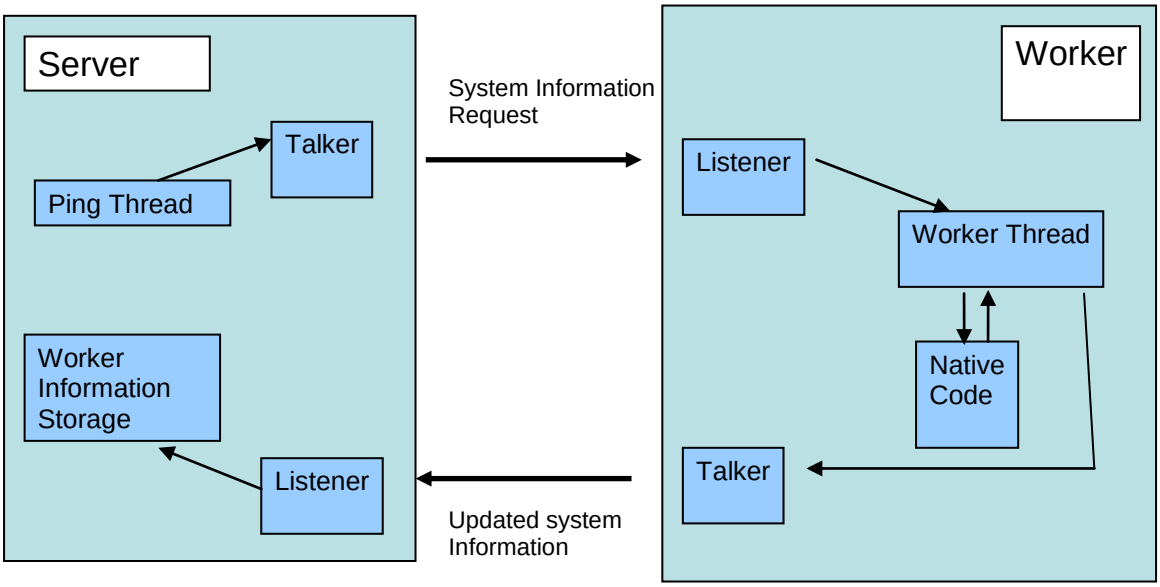


Figure 2.1 Periodic metrics collection by server

2.4.1 CPU Speed

CPU speed of the worker is of course collected by the worker and it sends it back to the server. On windows platform the actual data comes from the worker's system registry.

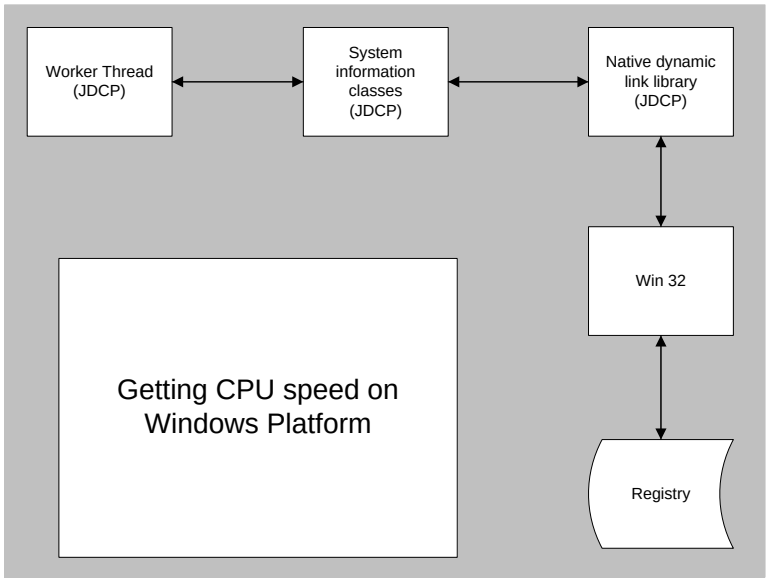


Figure 2.2 Getting CPU speed on Windows

On a Linux platform the data comes from /proc/cpuinfo file

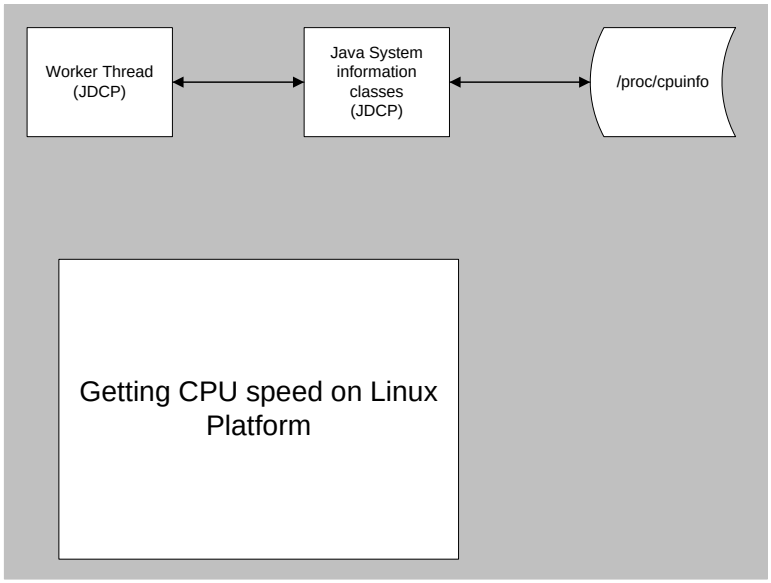


Figure 2.3 Getting CPU speed on Linux

2.4.2 Percentage CPU Usage

The CPU usage metrics is also collected by the worker and is sent to the server. On a windows platform the CPU usage is found through the use of PDH library that comes with the Windows platform. The PDH (Performance Data Helper) library is provided by Microsoft to allow developers to get system metrics. The JDCP system uses dynamic link library to connect to the PDH library and calls its functions to collect the data. The PDH library does not provide statistical information. It provides just a snapshot. This is why multiple snapshots are taken by the JDCP native library and the statistical data collected is used to calculate the actual CPU utilization over a period of time which in JDCP is 2 seconds.

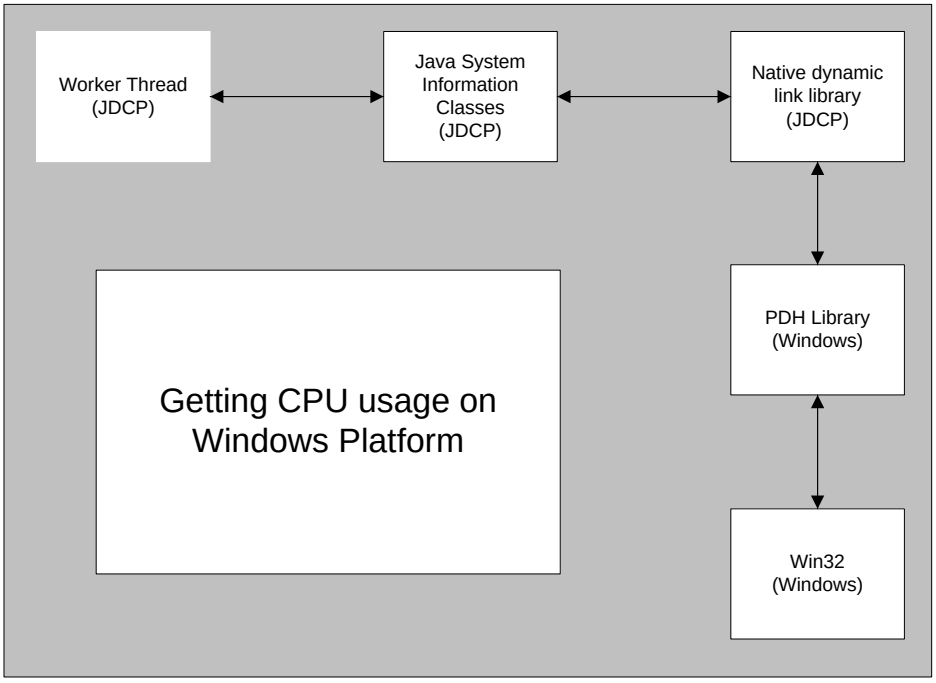


Figure 2.4 Getting CPU Usage on Windows

On a Linux platform the /proc/stat file contains information about the CPU usage. The JDCP system opens this file and reads the information again over a period of time to get the CPU usage. In this case the time is again 2 seconds.

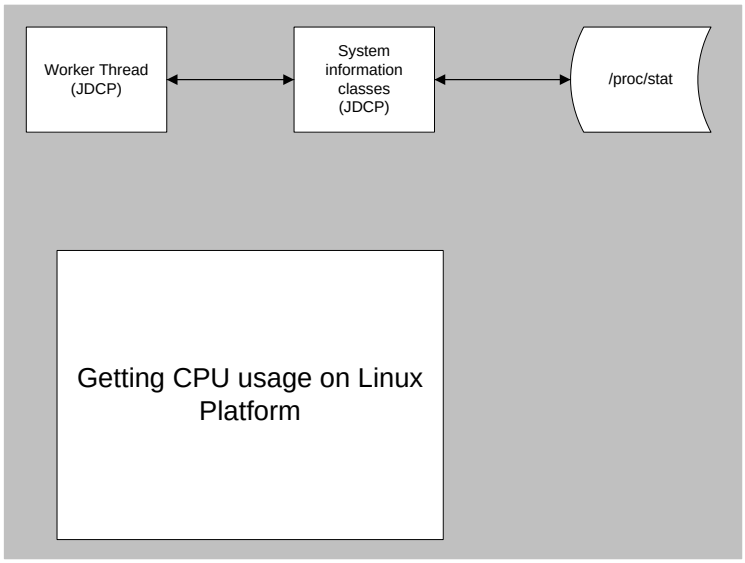


Figure 2.5 Getting CPU usage on Linux

2.4.3 Data Transfer Rate

Data transfer rate is calculated by recording the time and size of messages being transferred from the worker to the server. The worker has storage for size and time taken to transfer messages. Over a period of time the data collected is processed to find the average data transfer rate of the worker. The storage and calculation is done using ring buffer techniques.

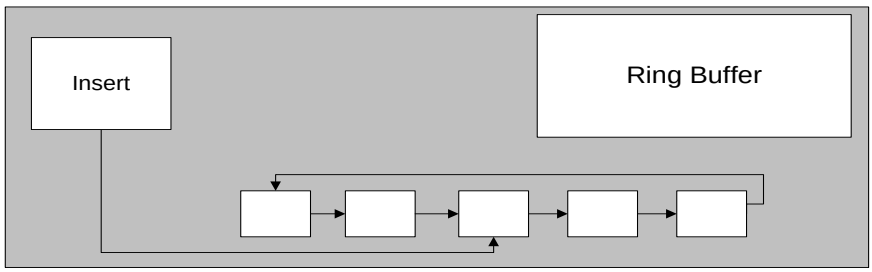


Figure 2.6 Ring buffer

2.4.4 Network Latency to Each Worker

Network latency is calculated as average of the time taken to transfer a message and the time taken to receive it. The mechanism used to calculate this metric is explained in the diagram below. The server puts a time stamp on the ping message and sends it to the worker, the worker puts and timestamp on the reception of the message. Once processed and returned to the server, the worker again puts another timestamp on the message. The last timestamp is put by the server when it receives the reply. Now the server calculates network latency as

Latency = [(Worker receive time – server send time) + (Server receive time – worker send time)] / 2

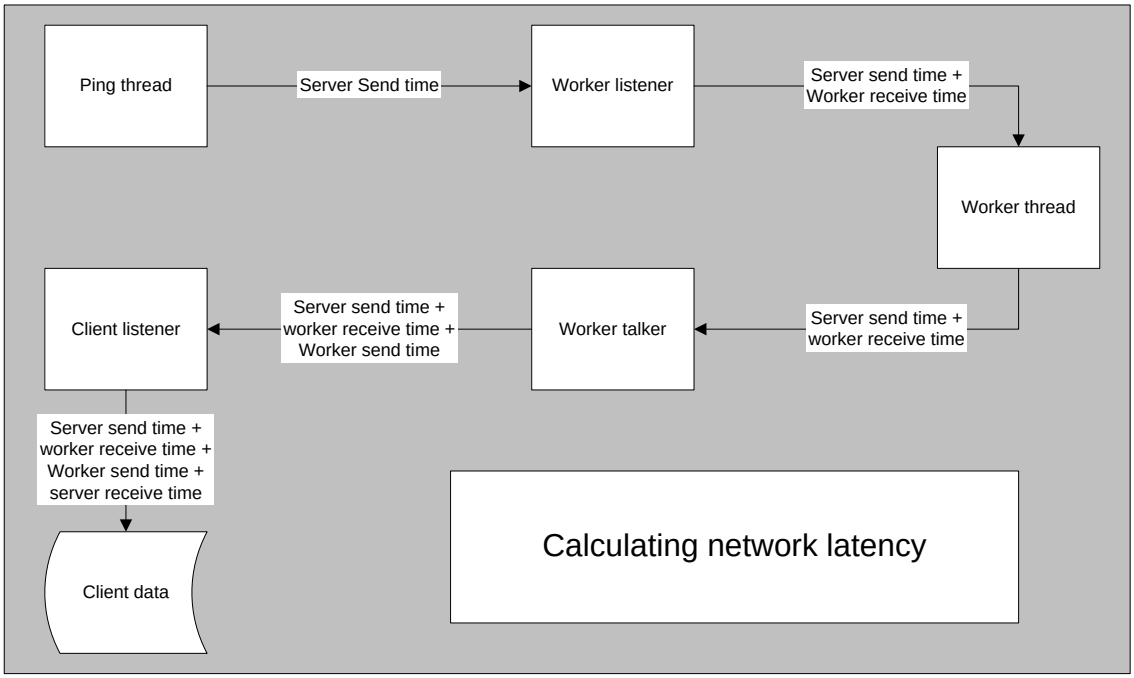


Figure 2.7 Calculating network latency

The above mentioned mechanism is used because it eliminates the errors that could happen due to clock skews on either side that is the server and the worker

machines. Hence this mechanism does not need the synchronization of server and worker clocks.

2.4.5 Data Reception Rate

This metric is collected by the server separately for each worker. Just as the worker stores the time and size of data transfers, the server does the same for each worker separately. Thus creating ring buffers for each connected worker.

2.4.6 Other Metrics

The following metrics are collected using similar ring buffer techniques as described above in cases of data transfer rate and data reception rate.

- Average processing time of each job
- Average processing time of all jobs by each client
- Average result size of each job
- Average result size of all jobs submitted by each client
- Expected processing time of jobs already assigned to workers

Chapter 3

SCHEDULING

3.1 INTRODUCTION

The project aims at developing a platform for creating high performance computing clusters. The platform can be used to create single-site and multi-site clusters and will be capable of combining computing resources of heterogeneous Scheduling process involves assigning available jobs to available worker machines. This assignment of jobs is done is such a way that the performance that is processing power extracted from the available resources is maximum.

In a clustering system such as JDCP in which jobs are real-time processing requests and relatively small as compared to large batch jobs scheduling needs that are a little different. First of all this real time assignment of jobs has to be done after small periods of time. This makes the processing complexity of the scheduling process an important issue. For example if the number of jobs and number of workers is both high then a very complex scheduling process may put too much burden on the server and hence despite having abundant resources the server may not perform as well. In such situations simple schedulers may work better. But then there are also situations where the resources connected to the server have a high variance. This means that the worker machines may have different processing capacities. Moreover the jobs submitted by the clients may also have a high variance that is one job may be of a much larger size than the other. The third type of variance is however the most important, the network

capacity. This includes data transfer and reception rate and network latency. In distributed systems these factors play the most important role. If the cluster machines have low network capacity this can hamper the performance of the cluster drastically. Of course if the machines do not get the data to process fast enough no matter how fast they process they will not be providing much power to the cluster. In situations where the above mentioned three variances are high simple schedulers may create schedules that utilize only a fraction of the total cluster power. For these situations more advanced schedulers are needed. The JDCP system caters for both these situations through having schedulers added as plug-ins into the system. This way the system supports multiple schedulers at the same time. Five schedulers have been developed for the JDCP system. These schedulers and the scheduling algorithm that they use are mentioned in this section of the report.

3.2 BENIFITS

3.2.1 Better Utilization of Cluster Resources

The schedulers create schedules based on the information provided to them by the system. By using this information the schedulers calculate various ways in which the available jobs can be scheduled on the available worker machines. This way the schedulers come up with the best schedule possible. This results in better utilization of the cluster resources.

3.2.2 Better Control at the Utilization of Cluster

Schedulers aim to optimize the resource usage of the cluster but this can be done in a number of ways for example some schedulers may aim to provide the clients with minimum response time while others may aim at increases the overall throughput of the cluster. This way the cluster administrator can tune the cluster in order to control the usage of the cluster resources.

3.2.3 Multiple Scheduling Policies

The JDCP system allows schedulers to be added as plug-ins into the system rather then they being integral part of the system .The system provides a Java interface to implement. Any class that implements this interface can act as a scheduler for the system. This allows anyone other than the main developer to create extensions to the system. The programmer making the new scheduler has to know nothing much about the rest of the system either in order to create fully fledged schedulers.

Another benefit of this approach is that multiple schedulers can be added to the system at the same time so that the administrator of the cluster can choose which scheduler to use at any given time.

3.2.4 Dynamic Changing of Scheduler at Runtime

This feature allows runtime changing of the scheduler. Nothing other than a function call has to be done in order to change the scheduler. This removes the need to shut down the server then change the scheduler and then restart the server.

3.3 JDCP SCHEDULING

3.3.1 The Scheduling Loop

The scheduling loop keeps on running until the server process is running. This loop is responsible for the collection of scheduling information, passing this information to the scheduler and getting the schedule back from the scheduler. Then the jobs are dispatched to the workers according to the schedule. Once dispatched the loop checks for the change of scheduler by the user. If the scheduler has been changed the new scheduler is installed as the new system scheduler and the old scheduler is discarded. Then the loop waits for a configured time so that more jobs accumulate in the client queues and then the loop runs again.

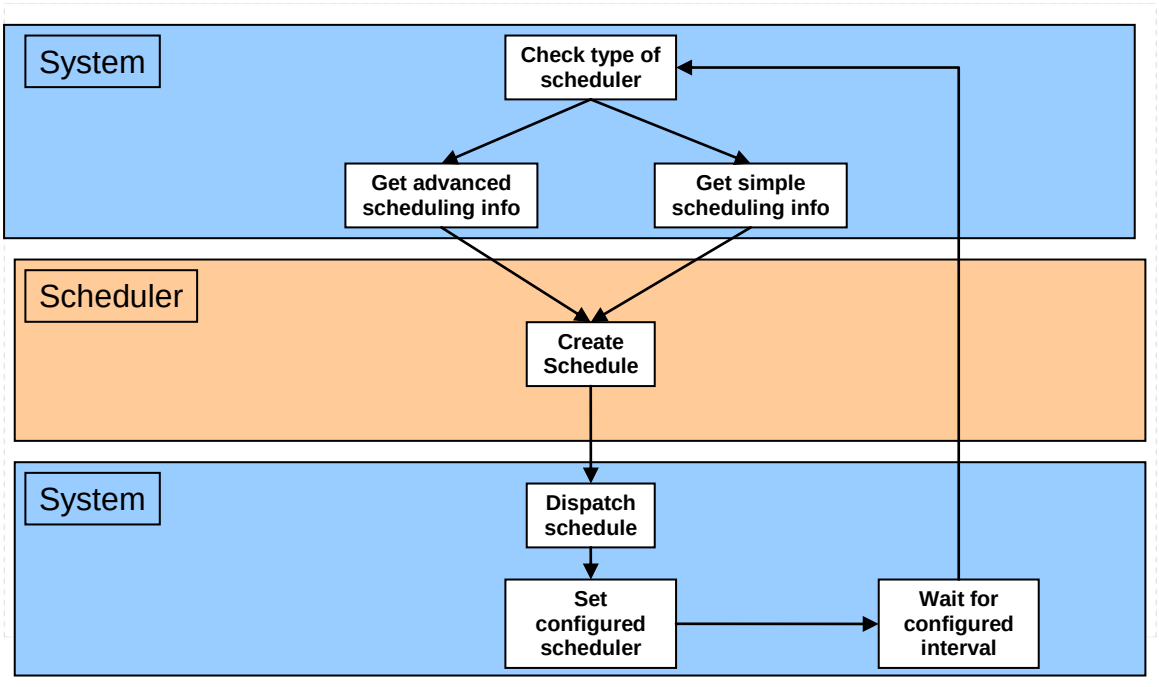


Figure 3.1 Scheduling loop

3.3.2 Inputs to Scheduling Process

- CPU Speed
- CPU Usage when not performing any job
- Network Delay
- Network Data Transfer Delay per Byte
- Network Data Reception Delay per Byte
- Expected Processing Time of Jobs Already at the Worker
- Average processing time of this method
- Average processing time of all methods called by this client
- Arguments size
- Average result size of this method
- Average result size of all methods called by this client
- Default expected processing time in system configuration
- Current jobs submitted by all the clients
- Current number of workers

3.3.3 Scheduling Algorithm

3.3.3.1 Get input data

Five methods are called by the client. Average processing times of these methods are

10 5 ? 5 10 5

? Means that this method has not been called before so it's processing time is not known.

On average all the methods called by this client took 7 processing time so we assign this value to the unknown method.

Hence expected processing times are

10 5 7 5 10 5

There are three workers available with processing capacity as follows

500 1000 1500

3.3.3.2 Normalize processing capacities of workers

Average processing capacity = 3000/3 = 1000

Hence relative processing capacities of the three workers are

1 2 3

3.3.3.3 Generate the expected time table

To find the expected time a worker will take to process a certain method we use the following formula

E = (Expected Processing time of the method*Average CPU capacity/Worker CPU capacity / CPU idle time %) + (2*Network delay) + (Argument size of method in bytes * Reception rate of worker) + (Average result size of method in bytes * Transfer rate of client)

Using this formula the scheduler will calculate the callback times of all the methods on all the processors thus generating a table. We assume here that the table generated by this formula is as follows:

Table 3 Expected time table

Processor	*1*	*2*	*3*	*4*	*5*	*6*	*Sum*
1	25	12	15	12	25	10	100
2	15	7	10	7	15	7	60
3	8	5	7	5	8	5	38

3.3.3.4 Adjustments with current job load on the workers

But here we have not considered the jobs already at a particular worker. We assume here that the total expected processing time of all the jobs at the three workers are

10 8 12

This processing will be done by the workers before they start working on the next scheduled work and hence this must also be added to each workers working time. Hence the new total working time of the workers becomes

110 68 50

These are the times the workers will take if all the work is given to each single worker. In other words processor 3 will complete all the work in 50 processing time. While if all the work is given to worker 1 it will do the work in 110 processing time.

3.3.3.5 Divide percentage of total work to workers

Now we have to divide the actual total work amongst these workers such that maximum throughput is achieved. Clearly worker 3 must get the highest share of the work, then worker 2 and the least work will go to worker 1. This is how we achieve that, take reciprocal of all the processing time and add them.

1/110 1/68 1/50

0.009 0.015 0.02

Sum = 0.009 + 0.015 + 0.02 = 0.044

To get the share of each processor calculate

0.009/0.044 0.015/0.044 0.02/0.044

20% 34% 46%

Here we present two approaches namely biased and unbiased. The comparison on these two approaches is given later.

3.3.3.6 Biased approach

Biased approach says that we sort the jobs according processor whose job assignment we are currently calculating. Here we calculate job bias according to processor 3 because this is the first processor we assigning jobs to.

Bias formula is given below

Bias = (job time on processor / sum of job times on all the processors)

Table 4 Expected times summed

Processor	*1*	*2*	*3*	*4*	*5*	*6*	*Sum*
1	25	12	15	12	25	10	100
2	15	7	10	7	15	7	60
3	8	5	7	5	8	5	38

Bias according to worker 3, the worker whose currently being assigned the work

Table 5 Biases

	8/(25+15+8)	5/(12+7+5)	7/(15+10+7)	5/(12+7+5)	8/(25+15+8)	5/(10+7+5)	
	0.16	0.2	0.22	0.2	0.16	0.22	

The less the value of the bias the better it is to schedule the job on that particular processor. Now its time to assign actual jobs to the worker 3. The jobs are

10 5 7 5 10 5

At least 46% of the jobs will go to worker 3. We assign jobs sorted in ascending order according to the bias. The new sorting results in job sequencing as

10 10 5 5 7 5

46% of the jobs is (10 + 10 + 5 + 5 +7 + 5) * 46/100 = 42 * 0.46 = 19.32 = 19

So at least 19 processing will go to worker 3. This limit is reached by assigning the first 2 jobs to the processor as 10 + 10 = 20 >= 19.

So these two jobs will go to worker 3. Rest we have 4 jobs left

5 5 7 5

Again we will calculate the bias according to worker 2 using the remaining methods. Here we assume that the sequence changes with the new bias and hence we get the new sequence as

5 5 5 7

34% processing must go to worker 2 so we calculate the processing time as

42 * 0.34 = 14.28 = 14

So at least 14 processing time will go to worker 2. This amounts to the first 3 jobs as 5+5+5=15 >=14. Hence first three jobs go to worker 2. The jobs left for the last worker automatically go to worker 1. Hence last job goes to worker 1.

3.3.3.7 Unbiased

We do not calculate the bias and simply assign the jobs in sequence so that the percentage that each worker should get is given to it. Here we do not sort the jobs and simply assign

0.46 * 42 = 19 to worker 3

0.34 * 42 = 14 to worker 2

rest to worker 1.

Hence the jobs are as they were at the start

10 5 7 5 10 5

To complete at least 19 we must give 10+5+7 = 22 > 19 to worker 3.

5+10=15>14 to worker 2 and rest 5 to worker 1.

3.3.3.8 Normalizing load on workers

All the steps performed till now ensure the maximum throughput but the response time may not be the best because of the existing load on the workers. For example if any worker is congested already the algorithm will still most probably assign it some job, which will translate into bad response time for the real time user.

To minimize the response time for the user the jobs will again have to be re-organized. Take the following example

Worker 1	**Current load - 25**	**Job - 5**	**Job - 12**	
Worker 2	**Current load - 15**	**Job - 7**		
Worker 3	**Current load - 25**	**Job - 7**	**Job - 10**	**Job - 7**

Figure 3.2 Un-normalized load on workers

Here worker 3 is the best worker and hence is assigned the most jobs by the above scheduling process, but as the current load on worker 2 was much less than worker 3, the response time can be increased by shifting the last job of worker 3 to worker 2. The new schedule becomes

Worker 1	**Current load - 25**	**Job - 5**	**Job - 12**	
Worker 2	**Current load - 15**	**Job - 7**	**Job - 10**	
Worker 3	**Current load - 25**	**Job - 7**	**Job - 10**	

Figure 3.3 Normalizing load on workers

The throughput has reduced as worker 2 is slower than worker 3 and the job has expanded from 7 to 10 processing time but the response time has become better, that is from 49 to 42.

Here in this example a shift has been made but this can also be a swapping, if shifting is not good. This would be in the case when shifting results in a longer schedule then the original one.

The algorithm for this normalization is as follows

- Compare the highest loaded worker and the lowest loaded worker. Try to shift the last job of the highest loaded worker to the lowest loaded worker. If the shifting results in a shorter schedule then make the change permanent. Repeat step 1.
- Else if the shift makes the schedule longer then try to swap the last jobs of the two workers. If the new schedule is shorter than make change permanent. Go to step 1.
- Else Perform step 1 but change lowest loaded worker to second lowest loaded worker
- Else Perform step 2 but change lowest loaded worker to second lowest loaded worker
- Else stop normalizing as the schedule is already normalized enough

3.4 COMPARISON OF SCHEDULING APPROACHES

Here we compare round robin, biased and unbiased approaches according to the example taken above.

3.4.1 Round Robin

For round robin we assign jobs

10 5 to worker 1

5 10 to worker 2

7 5 to worker 3

So expected callback times on these workers will be (refer to table of expected times)

Worker 1 25 + 12 = 37

Worker 2 7 + 15 = 22

Worker 3 7 + 5 = 12

So total time to process all the requests will be 37 as this is the highest time required by any worker.

3.4.2 Unbiased Approach

As we have seen in unbiased approach we assigned following jobs to the workers

Worker 1 5

Worker 2 5 10

Worker 3 10 5 7

So expected callback times on these workers will be (refer to table of expected times)

Worker 1 10

Worker 2 7 + 15 = 22

Worker 3 8 + 5 + 7 = 20

So total schedule time will be 22

3.4.3 Biased Approach

As we have seen in biased approach we assigned following jobs to the workers

Worker 1 7

Worker 2 5 5 5

Worker 3 10 10

So expected callback times on these workers will be (refer to table of expected times)

Worker 1 15

Worker 2 7 + 7 + 7 = 21

Worker 3 8 + 8 = 16

So expected total time will be 21

3.5 JDCP SCHEDULERS

As mentioned previously JDCP system comes with five schedulers already implemented. A brief discussion on these schedulers follows which explains the need for having multiple schedulers instead of just one.

3.5.1 Round Robin Scheduler

This is a simple scheduler and creates schedules that distribute available jobs to workers in a round robin fashion. As explained earlier if the variance between resources is minimum then a simple scheduler like this works good enough. As an example if all the workers connected to the server are of the same CPU speed and have the same network capacity then round robin scheduling performs best without putting extra processing burden on the server machine.

3.5.2 Unbiased Maximum Throughput Scheduler

This scheduler puts less burden on the server machine as processing is less but it still creates schedules that handle variance of resources well. The schedules are aimed at producing maximum throughput from the available resources.

3.5.3 Biased Maximum Throughput Scheduler

This scheduler puts extra processing burden on the server but as explained earlier the schedules created by this scheduler are mostly better than the unbiased version of this scheduler. This scheduler also aims at maximizing the throughput of the cluster.

3.5.4 Unbiased Minimum Response Time Scheduler

This scheduler requires less processing due to unbiased approach so in cases of large number of jobs etc this scheduler may perform better. Especially if the server machine is a bit slow. This scheduler aims at reducing the response time for the clients. This is a major requirement in real time systems.

3.5.5 Biased Minimum Response Time Scheduler

This is the extra processing version of the above mentioned scheduler. This scheduler is supposed to perform the best in maximum types of job loads.

Chapter 4

INFORMATION SERVER

4.1 INTRODUCTION

This server acts as a discovery service for the clients. It provides information about currently available servers that have connected to the information server. Servers connect to the information server and advertise their resources periodically. Clients connect to get the server information and choose the server that best meets their needs

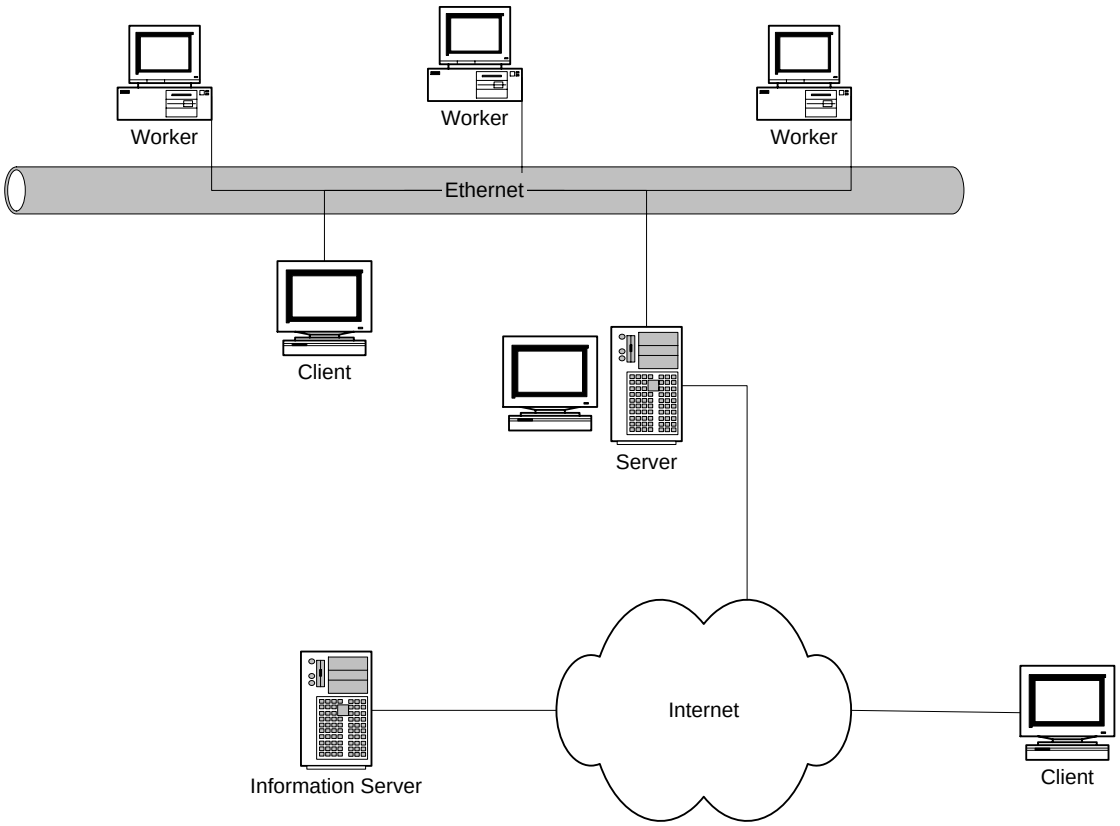


Figure 4.1 Deployment view with information server

4.2 BENEFITS

4.2.1 Discovery Service for Clients

As mentioned earlier the information servers act as a discovery service. The client machine only needs to know the IP address of this information server. Once connected to this information server the client can get information about available servers, getting their connectivity information and resource information.

4.2.2 Clients Can Choose the Server that Best Suites Them

This is a major advantage of this server. The server provides information about resources that each available server has which includes number of workers, average CPU speed, load on the server, clients already connected to the server and network latency of the worker connected. All this information can be used by the client program to change the way it sends jobs to the server. If for example the number of worker connected to the server is 20 then to achieve maximum parallelism the client must send at least 20 jobs. More advanced clients can take advantage of the network latency and other information.

4.3 SERVER INFORMATION

The information server receives server information from the connected servers and transfers it on demand. The following information is received and transferred.

4.3.1 Connectivity Information

- Name of server
- Name of organization

- IP address and socket for connection establishment

4.3.2 Resource Information

- Number of connected clients
- Number of connected workers
- Average CPU Speed
- Average CPU load
- Average network latency to workers

4.3.3 Services Information

- Security levels
- Job levels (Whether pre-defined job dependencies are supported or not)
- Scheduling policy
- Uptime
- Downtime

4.4 QUERY INTERFACE FOR CLIENTS

This is the API provided to the client to get the information about the servers that have connected to the information server. ServerInfo object contains all the information mentioned above.

String[] getAllServers();

ServerInfo getServerInfo(String name)

ServerInfo[] getAllServerInfo();

Chapter 5

FAULT TOLERANCE

5.1 INTRODUCTION

Fault tolerance is of great importance in a clustering system. Fault tolerance includes hardware failure such as network disconnection and machine failures and also software failures such as process crashes etc. Fault tolerance becomes important because without it cluster management will be impossible. If a single machine stops working the whole clusters may become useless because the server is expecting to get a job back from the failed worker and it never returns the job thus the client never gets all the results back. This is just one of the many cases that can result in bad quality of service. JDCP system has an extensive fault tolerance framework which includes handling machine failures, network failures and software failures. It is important to mention here that the fault tolerance in JDCP platform does not cater for bad networking etc. The system expects a uniform behavior from the underlying hardware to function to its full capacity. The fault tolerance mechanisms cater for recovery from failures not prevention of failures. As a result the performance may go down significantly if the hardware and other failures occur but the system guarantees recovery and thus completion of client jobs.

5.2 DEVELOPMENT

5.2.1 Handling of Worker/Server Disconnection

Worker disconnection handling involved a number of additions to the system. First was the detection of the disconnection. This is done through socket option keep alive and also through the use of a ping thread. A disconnection is not detected only when some data is transferred over the socket so this is periodically done by the ping thread. Once the disconnection is found the next step JDCP does is to change the state of the worker from connected to disconnected. This makes the worker unavailable for schedulers so no new job is assigned. The next step is to recover the jobs already transferred to the crashed worker. The worker will never return the jobs so these jobs have to be rescheduled on the remaining worker. This is done through recovering the work from the work cache and making the jobs available to the schedulers again.

On the worker side the handling for this disconnection is that the worker once disconnected periodically tries to reconnect to the server. Once the network connectivity is established again the worker automatically reconnects and becomes available to the server. The jobs lost during this reconnection are not recoverable.

5.2.2 Controller Thread for Workers

In the initial system the server and worker had only one connection. Now the server and worker have two connections, one control connection and one work connection. All work requests and results are transferred through the work connections. All the control messages are transferred and received through the control

connection. This results in quick and more robust control messaging between the server and the worker.

5.2.3 Ability to Restart the Worker From the Server

The JDCP server now has the ability to restart the worker thread on all the worker machines. This feature was added in case the worker thread in stuck in an infinite loop or is blocked on a failed IO operation. Now in such as case the server can restart the worker thread thus getting the worker out of this situation.

5.2.4 Complete Server Crash Recovery

A major part of fault tolerance was focused on server crash recovery. This is necessary because client submits jobs and disconnect from the system while their jobs get processed. If a client submits a job and the server crashes and loses the client job then this seriously degrades quality of service. Even more critical is the performance issue. The results processed by the workers are transferred to the server once completed and are stored at the server. These results are transferred to the client automatically in case of real time jobs and on demand in case of batched jobs. If the server machine crashed and loses all the results it has then of course once restarted the jobs will have to be processed again. But now the JDCP server stores these results in persistent storage and does not lose them. To achieve this task the server machine takes two types of measures. First are the measures taken during the normal processing. This includes storing scheduled jobs and results to persistent storage, deleting the completed jobs, deleting results once transferred back to the client and in case of batched jobs in case the client explicitly asks the server to delete the jobs.

The second type of measure are the once taken when the server restarts after a crash. Once restarted the server goes back to persistent storage and recovers the jobs assigned to the workers and the results that were already at the server. Once recovered the server also creates a list of disconnected client so that new clients are not connected with already n use client IDs.

Currently all of this is being done through files. This is done keeping in mind the real time processing. A database can be used instead but with all the jobs coming and going through the server this may degrade the performance. But database implementation of these mechanisms can also be tried in the future as this will result in a more robust solution. The performance difference can only be analyzed once the database implementation is ready so currently nothing can be said in comparison of these two approaches.

Chapter 6

SECURITY

6.1 INTRODUCTION

Security becomes very important when the clients of the cluster are not trusted by the cluster administration and vice versa. This adds a need for authentication both ways before clients are connected to the system. Plus it also adds a need for client code restrictions. The client code is processed remotely on the cluster machines. Thus a malicious client program can destroy the worker machines if the code is not restricted. JDCP system leverages the Java Sandbox concept to provide complete worker safety by restricting the client code completely from using local hardware resources directly. JDCP system also features remote connection by the client that is over the Internet. This adds a need for encryption because the client code, requests and results are traveling through the public Internet.

Keeping all these requirement in mind the JDCP system has an extensive security framework including Java Sandbox implementation on worker, password based and digital signatures based authentication of clients and the server and complete public/private key encryption and private session key encryption.

6.2 BENEFITS

6.2.1 Client Authentication

Authentication varies according to the security level currently set by the server. In no security option no authentication is done. In second level clients are authenticated using passwords. At the third level the clients are authenticated using digital signature mechanism.

6.2.2 Encryption of Data

Encryption is done only at the third level of security. The JDCP system encrypts all messages transferred over the network between the client and the server. No encryption is done on the worker side because they are assumed to be trusted machines and encryption also hampers performance. The encryption of initial connection establishment messages is done using RSA public/private key mechanism. Once connection is established rest of the encryption is done using DES private key cryptography. This is done because of performance reasons as DES is mush faster than RSA.

6.2.3 Security Performance Tradeoff

The JDCP system comes with three security levels. First level is no security which provides maximum performance. The second level is password based authentication. The third level contains digital signature authentication and encryption. These multiples levels are provided because security and performance are inversely proportional. It is the choice of the cluster administration to decide the security level.

6.3 DEVELOPMENT

6.3.1 No Security

This level provides no security. Users connect as default users thus connected clients cannot be distinguished from each other by the server.

6.3.2 Password Based Authentication

Provides authentication of clients and worker but passwords can be spoofed during network transmission.

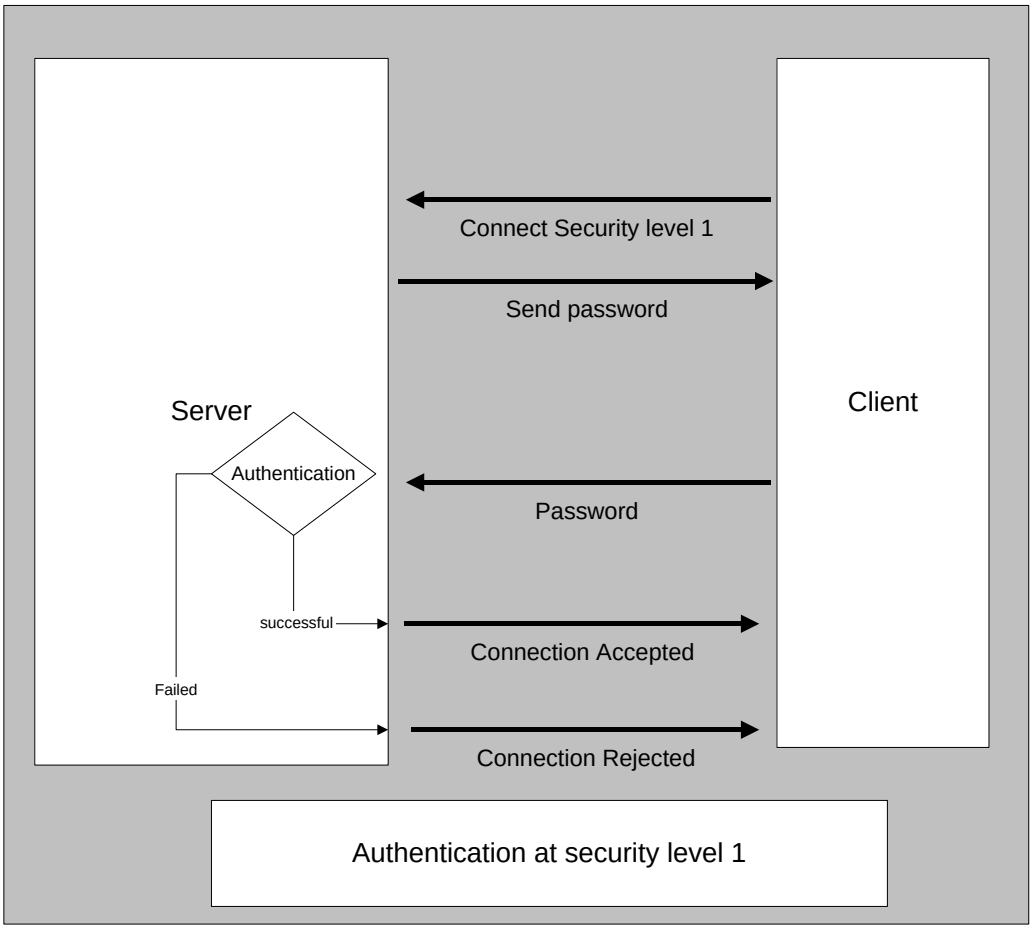


Figure 6.1 Authentication at security level 1

6.3.3 Complete Security

Authentication will be through digital certificates. Session key will be used for encryption of processing messages. Initial messages will be encrypted using public/private keys and authentication will be through digital signatures. Once connection is established and session key has been transferred all further communication will be done using private key cryptography on the session key.

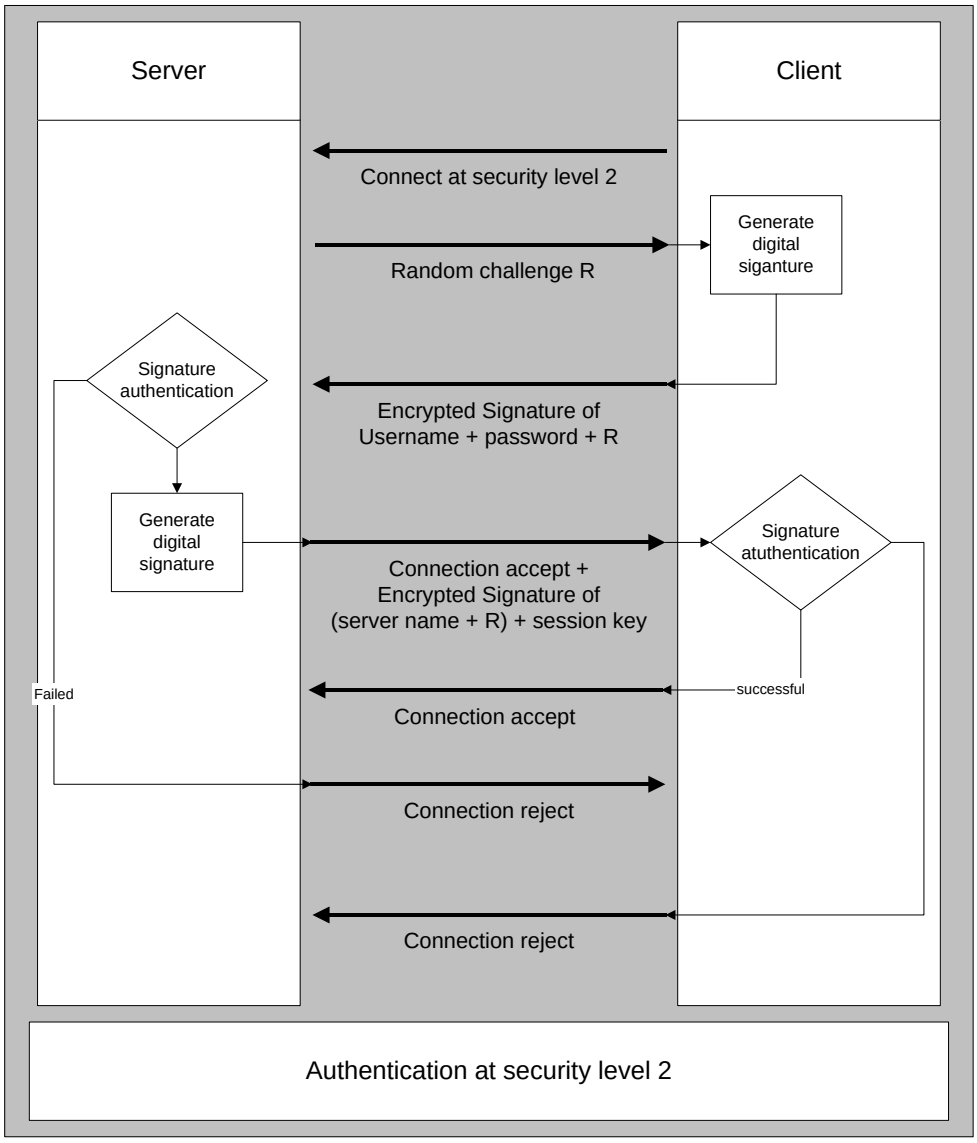


Figure 6.2 Authentication and encryption at security level 2

Chapter 7

BATCH SUPPORT

7.1 INTRODUCTION

JDCP system provides batch support. This means that the system accepts batch jobs and the client can submit these jobs and then disconnect from the system. The client can then reconnect at a later time and request its job results back. This does not include supporting native binaries. Batched jobs are still Java method requests. The batch jobs in JDCP are not treated separately during the scheduling process. The difference lies in the transfer and deletion of jobs at the server side. Results for real time jobs are automatically transferred back to the client machine but batched results are not transferred automatically. The client has to send a request to get the results back. Results for the batch jobs are also not deleted automatically by the server as they are in case of real time jobs.

7.2 BENEFITS

7.2.1 Batch Processing

As client reconnection has been added as part of the batch support feature, client can now submit all the jobs that they wish the server to process and then disconnect. They connect at a later time and get the results of their jobs transferred to them. In the initial system client had to remain connected to get their results back as

the server did not have the feature to reconnect clients and it accepted every client connection as a new client.

7.2.2 Handling of Server/Client Disconnection

The client reconnection feature was mainly added for the above mentioned batch jobs but it also helps in cases when the client/server connection fails and the client has submitted real time jobs. The client can now reconnect as the disconnected client and the server will automatically transfer the results of its real time jobs back to the client machine.

7.2.3 Multiple Input Files

Now the JDCP client can attach multiple files to each submitted job. This is a necessity for jobs that require more than one class file and an option for jobs that wish to process file data from files stored on their hard disks. Now video and image files and any other type of file can be attached to a job and these files are transferred and processed remotely by the JDCP system.

Chapter 8

JOB DEPENDENCY OPTIMIZATIONS

8.1 INTRODUCTION

Invariably the results from one job go as input to the second job in a program just as an object returned by one method is passed as argument to another method. But to do this transfer in a distributed system like JDCP is not easy. As the processing calls are asynchronous the client does not have any control over a submitted job. The server processes each job separately. To implement this output as input function explained above the JDCP system now provides job dependency optimization. The optimization focuses on reducing the number of network transfers. This will lead to less time required for sending and receiving the requests and results and also the idle time wasted due to workers having no work to do and reading a particular request from the server.

8.2 BENEFITS

8.2.1 Reduced Transfers

In the current systems all jobs submitted by the clients are separate in all aspects for the JDCP system. The input, processing and output of a job is not connected to any other job. But as we all know for the client these jobs are related because they are part of the same applications and the results from all these jobs are combined at the client. Jobs can be related to each other in the following ways

- The output of one job goes as input to another job
- The object processed by one job can be used in another job as the base object on which the method is called

In the current system this decision of providing output as input is done by the client. First the result of the provider job is retuned to the client which inserts the result of this jobs as input to the next job and then submits the new job to the server. This result is less processing time and large network transfer time for this whole processing sequence. If the server is able to handle these output as input functions this will lead to lesser network transfers and hence an improved system performance. The reduction in network transfers is shown in the following diagram.

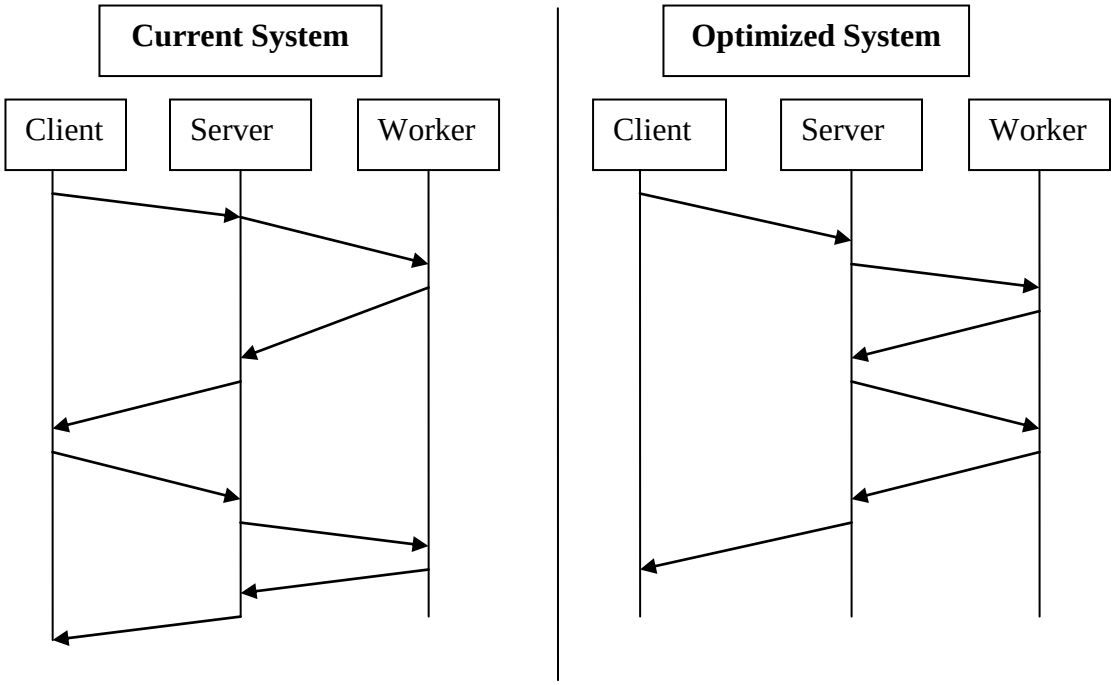


Figure 8.1 Reduction in messages through job bundling

The numbers of network transfers have been reduced from 8 to 6 that is 25% reduction of number network transfers. This might not seem a big optimization to the

system but as the number of jobs increases the amount of optimization also increases for example for 10 jobs the bundled together with dependencies makes a reduction of 80 to 42 that is a 47% reduction of number of network transfers. One aspect of this reduction is worth mentioning that the optimization reduces the number of network transfers not the amount of data transferred. So this is useful in cases when the client to server network latency is of a large magnitude.

8.2.2 Bundled Jobs

This optimization also provides the client with an API to bundle multiple jobs together hence the client can create a bundle at its own side and transfer this whole bundle to the server for processing. After this job/dependency bundle has been transferred to the server the client program can even disconnect from the server and reconnect at a later time to get the results of all the submitted jobs.

8.3 DEVELOPMENT

8.3.1 Defining a Dependency

Dependencies will be defined by the client using four parameters

- Token of method request providing the input
- Token of the method request receiving the input
- Argument number of the receiving request that is to be provided
- Whether the result or the object from the provider request is the input

Hence if the dependency in normal code is like

```
Int x=Aclass.getInteger();
```

```
Aclass.processInteger(x);
```

The translated code for JDCP will be

```
Int tokenProvider=executeRequest("AClass", "getInteger");
Int tokenReciever=executeRequest("Aclass", "processInteger");
addDependecy(toeknProvider,tokenReciever,1,"result");
```

8.3.2 Job Submission

Client submits a bundle of jobs and the dependencies in them. The bundle contains the requests and dependency objects

```
BundledRequest
{
        WorkRequest[] requests;
        Dependency[] dependencies;
}
```

The client requests will not be transferred immediately to the server instead they will be added to the bundle. Once the bundle is ready the client will call a method to transfer the whole bundle to the server.

8.4 PROCESSING APPROACHES

The server now has to schedule the jobs while keeping the dependencies in consideration. This job is best done by the scheduler because it can fully exploit the information to create optimized schedules but if the processing of these dependencies is done by the system instead of the scheduler then the there are two ways of handling the dependencies

8.4.1 Turn by Turn Processing

The jobs that are dependant on other jobs will not be added to the schedulable jobs list hence the scheduler will not schedule these jobs. Once the jobs that are acting as the providers for the dependant jobs get processed and returned, the system will add these dependant jobs to the schedulable queues also and hence the jobs get scheduled turn by turn. In this case the scheduler will not be changed at all while the system change will be of some magnitude. The drawback of this approach is that dependant jobs are processed much later in the system which reduces response time and in certain cases the throughput also.

8.4.2 Bundled Scheduling

Bundled jobs will be transferred to the workers as a unit and hence one bundle gets executed only at one worker. The scheduler will have to change a little because now it has to schedule not only simple requests but also bundled requests. The complexity of the system is reduced in this option as handling of bundled jobs will only be slightly different from simple jobs. The most obvious drawback of this approach is reduced parallelism which in extreme cases becomes zero parallelism.

One question may arise in this type of processing. What is the use of this bundling as the client can perform the same thing without bundled jobs? To explain the above question analyze the following code

```
//……..
Object object=getObject();
Object=processObject(object);
//……..
Object getObject()
{
      return new Object();
}
Object processObject(Object o)
{
          //        Do the processing
          //        return the object
}
```

Sample code that can benefit from job bundling

Here the job in JDCP will be scheduled in a bundle containing the two jobs and one dependency. But the client can do the same thing without the dependency like this

```
//……
```

```
Object object=getAndProcess();
//……

Object getObject()
{
      return new Object();
}
void processObject(Object o)
{
          //         Do the processing
}

Void getAndProcess()
{
          Object  object=getObject();
          processObject(object);
}
```

Implementing job bundling without JDCP bundling API

This shows that the client can itself implement the dependencies that are valid for both local processing and also for JDCP processing. This means that the client gets no benefit from JDCP bundling feature but this is not always the case because of the following reasons

8.4.2.1 Intermediate results

This means that the results from the providing method may have more use than just be input to another method. The results may be used for maintaining a log and also for further processing. To get both jobs done is not possible with the above mentioned approach but it is possible using the JDCP system. The client just has to mention that the result from the providing method is to be transferred back to the client and also used as input to the receiving method request.

8.4.2.2 Addition of new methods

- The class that needs this additional method may not be have been written by the client hence addition would not be possible unless source is available
- The methods called may be protected and the class may be final
- The class may be archived
- The class may be from rt.jar hence unchangeable

8.4.3 Combining the Two Approaches

The above mentioned approaches can be combined such that simple jobs and bundled jobs both can be scheduled at the same time. The scheduler will have to be capable of handling both types of jobs. Bundled jobs will be broken into simple jobs by the system if any of the following condition is satisfied.

- A bundled job is too large to be scheduled on a single worker.
- The number of bundled jobs is less than the number of workers.

The system will not add the dependant jobs to the schedulable job list. Hence the scheduler will get only bundles and simple jobs that are schedulable that is are independent entities

This approach leads to better utilization of worker resources as more jobs are schedulable at a given time. Moreover performance increases because number of network transfers is reduced. This approach can be taken as an increment to the turn by turn scheduling so it will be better to leave this for the moment and first implement the turn by turn scheduling then if time permits the second approach can be added.

8.5 DEPENDENCY HANDLING BY SCHEDULERS

The main motivation of behind allowing the scheduler to handle the dependencies is that it can take benefit of these dependencies to create maximum parallel schedules by bundling jobs together in such a way that the networks transfers are reduced to a minimum and the worker processing time is made maximum. If the above mentioned benefits are not exploited by the scheduler than there is no need to allow it to handle the dependencies.

8.5.1 Job Bundling

If related jobs (jobs having dependencies between them) are sent together to the worker by the server as they were sent by the client to the server then the number of network transfers can be reduced even more as explained by the following diagram

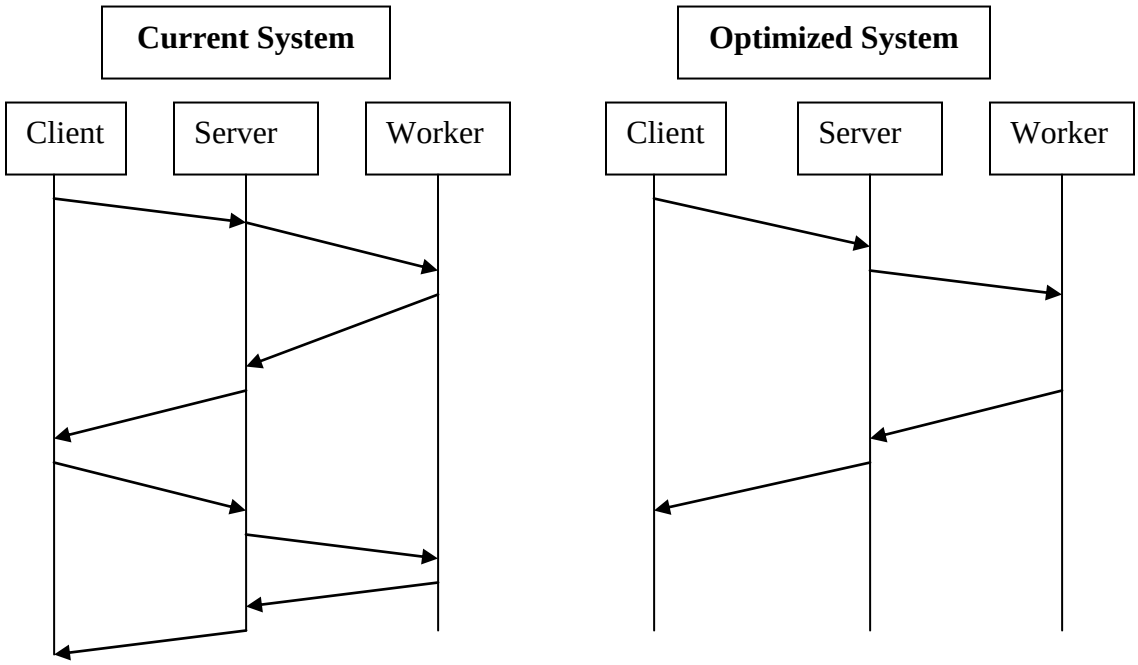


Figure 8.2 Advanced optimizations by dependency aware schedulers

Here as we see the network transfers have been reduced from 8 to 4, a reduction of 50%. But the above example is very simple, the actual situations for the scheduler will not be that easy for example if 50 jobs of different expected processing times are to be scheduled on 7 workers then what jobs should be bundled together is a complicated matter. The following discussion elaborates the efforts required to achieve a good bundling solution. This solution becomes even more complicated because schedulers can be added as plug-ins to the system and the system has to prepare the information required for this scheduling irrelevant of the scheduler being used.

8.5.2 Necessary Modifications

The following changes will have to be made to the system to enable this feature

- Client has three queues instead of one
- Simple request Q already working

- Bundled request Q new to the system
- Unschedulable jobs Q (These are the jobs that have as input results from jobs that are already running on workers, hence these can neither be bundled or be scheduled)
- Before scheduling From_Client_To_WHorse asks a new entity the GraphManager to create dependency graphs of each client's requests.
- GraphManager gets the requests from each client and creates bundled requests depending on the job dependencies between the jobs. It will add unschedulable jobs to the appropriate Q. Jobs from this Q will not be added to the simple and advanced scheduling info objects.
- From_Client_To_WHorse creates the Simple or Advanced Scheduling Info Objects including both simple jobs and bundled jobs. The inputs to jobs coming as results from other jobs will also be put as arguments to the jobs.
- Scheduler has the options of scheduling solitary requests and bundled requests. It also has the option of scheduling new bundles created by the scheduler and also solitary jobs inside a bundle.
- The schedule will be transferred to the GraphManager which will analyze the schedule and will modify the graph to incorporate the changes made by the scheduler, to add newly created bundles, delete the old bundles and check for the validity of the schedule in relation to the graph and its dependencies. It will throw an Exception if any rules of graph dependencies are violated.

- From_WHorse_To_Client will add the results received from the workers. It will decide what to do with the result that is whether they have to be deleted from the server or they are to be added to persistent storage for future reference.

8.5.3 Load Analysis

The scheduler will first analyze the load and available resources so that better bundles can be created which result in maximum throughput and response time for the clients.

The characteristics to analyze are

- Capacity of workers
- Number of bundles
- Processing time of bundles on various workers
- Levels of job dependency

For an ideal situation we will have

- Number of bundles equal to $\sum_{(i=0 \text{ to } n)} \frac{\text{Worker CPU Speed}}{\text{Lowest CPU Speed}}$
- All bundles should have the same expected processing time.
- Only one level of dependency between the bundles

8.5.4 Normalization of Bundles

To get close to the above described ideal situation we have a simple algorithm

- Add the expected processing time of all the jobs
- Divide this by the number of ideal jobs to get the ideal bundle processing time
- Pick bundles one by one and check
    - If the bundle's expected processing time is more than 2 times the ideal bundle processing time then break the bundle (Bundle breaking algorithm is described later)
    - If the bundle processing time is between 2 times the ideal time and half the ideal time then accept the bundle
    - If the bundle time is less than half the ideal time than merge with most appropriate bundle (Most appropriate bundle algorithm is described later)

8.5.5 Analysis of Scheduler Handling of Dependencies

The above mentioned algorithm is not very difficult to implement but the following aspects should also be considered

- The two sub algorithms that is the bundle breaking algorithm and bundling creating algorithm are not as easy because they involve heuristic analysis
- I will have to read a number of research papers that people have written on this subject to get a better understanding of bundling algorithms
- I will also have to do my own research according to the JDCP system because it differs from the existing solutions in a number of ways.

- To have an effective algorithm for JDCP a lot of testing with different sample jobs will also have to be done to test the effectiveness of the new optimizations in varying situations.

8.6 CONCLUSION

- Turn by turn scheduling is the easiest to implement and provides a good optimization especially for a large number of jobs. Turn by turn scheduling also requires less processing and this also makes it a good candidate if the number of jobs is large and the server machine is relatively slow
- Bundled scheduling reduces the number of transfers but it also reduces parallelism. In fact in certain cases when all the jobs are dependant on each other, this approach provides zero parallelism. Hence this approach is not good in most situations. It is relatively easy to implement.
- Dependency handling by the schedulers provides the maximum optimization but it is also the most difficult. The amount of work needed to implement this approach is too large to be completed within the current time frame and can delay the completion of the project so currently it is not feasible to take this approach.

As a conclusion to the above discussion I chose to implement turn by turn scheduling. Scheduler handling of dependencies provides the maximum optimization so this must be kept as a possible future enhancement of the system

Chapter 9

TESTING

9.1 INTRODUCTION

Testing is of course an integral part of the software development life cycle. Testing a clustering system like JDCP is very different from testing a normal desktop application because desktop applications should meet the requirements laid down at the start of the project and conformance to these features is all that is to be checked. In a high performance clustering system testing not only includes conformance to the initially stated requirements but also includes performance testing under various loads and various hardware environments. This makes JDCP testing a lot more difficult than a normal system. As JDCP scheduling caters for large variances in hardware resources, complete testing was not possible unless this wide range of hardware was actually available. Hence only some features of JDCP are tested and most of the testing was done for conformance to requirements rather then performance.

9.2 TESTING ENVIRONMENT

9.2.1 Single Workstation

Conformance testing was done only a single workstation. This included testing features like cluster monitoring, scheduling, batch support, fault tolerance, security and information services. Only the proper working of these features was checked and no performance data was collected.

9.2.2 NIIT Lab

Performance testing was done in the NIIT lab with a maximum of six worker machines. Server, client and information services were run on the same workstation. This was done only for simplicity of installation because otherwise the testing would require eight workstations.

9.3 TESTING SOFTWARE

As the clients for JDCP are programs so it was best to use client programs was testing purpose so that exact measures of performance received by the clients can be calculated. For this reason the following client programs were made that check different performance measures for any cluster created through JDCP system.

9.3.1 Worker Performance Monitor

This application first locally encrypts a given data and notes its processing time, then encrypts the same data remotely on the cluster and measures the performance advantage achieved through the use of the cluster. As it encrypts the data it draws a graph dynamically that shows the performance achieved through the cluster encryption as a percentage of the local encryption time. The following snapshot is taken when the application has yet not encrypted any data on the cluster. The blue line shows the ideal line which shows 500% performance increase if five workers are connected. Of course the actual results will always be below this ideal line as the network transfers of data reduces the performance.

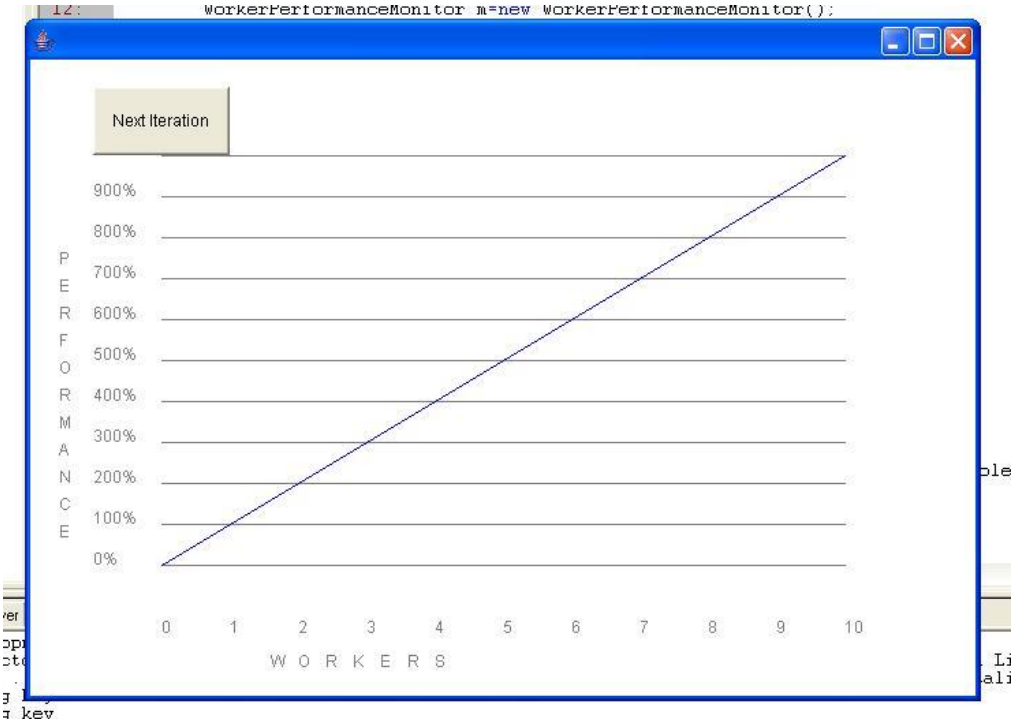


Figure 9.1 Worker performance monitor GUI

9.3.2 Data Performance Monitor

This application tests the performance of the cluster with same resources but different data sets. The application basically determines how much the cluster performance degrades as the data volume in the jobs increases. This measures the efficiency of the system at varying loads. The application sends jobs with 1MB, 2MB and 3MB and calculates bytes encrypted per second. This encryption is not related to the JDCP security encryption, it is just the client application that encrypts remotely. The graph line should a straight in ideal condition but this is not possible because as more and more data is transferred over the network the more the performance degrades as more time is spent in transferring data.

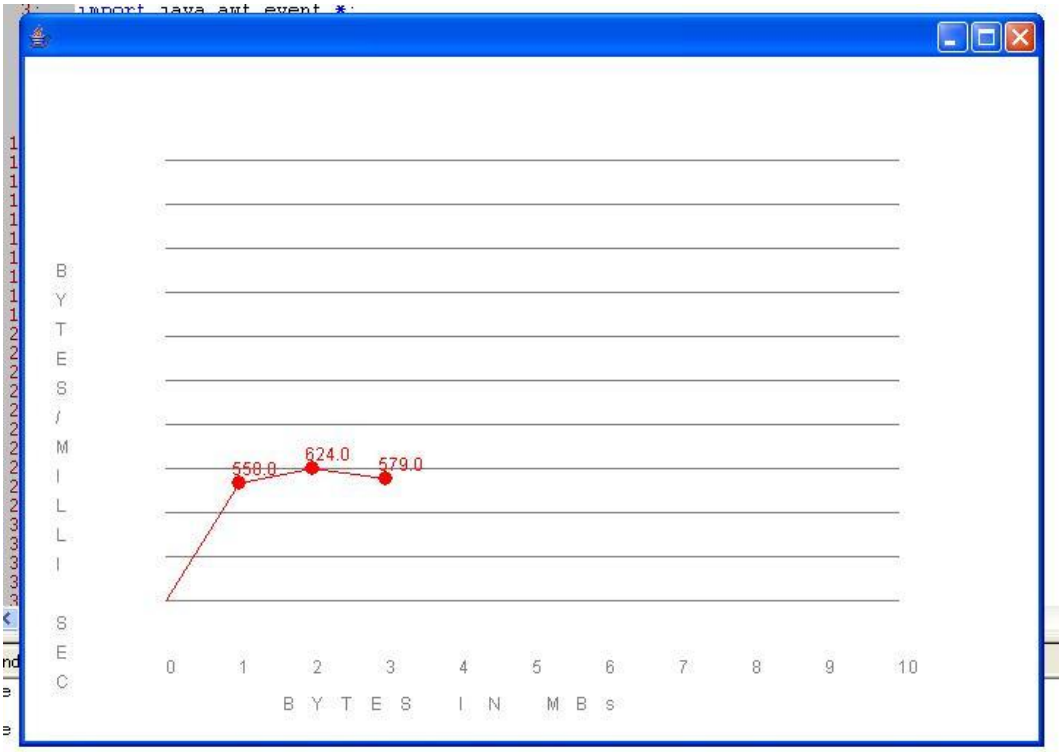


Figure 9.2 Data performance monitor GUI

9.4 TESTING RESULTS

9.4.1 Worker Performance Monitor

Worker performance monitor was run in the NIIT lab and showed the expected results. First the local encryption time was calculated and then the cluster with one worker was given the same task. The result showed almost 100% performance. One by one more workers were connected and the same jobs were given to the cluster. The following snapshot displays the graph plotted by the client application. For four workers the ideal performance was 400% and the actual performance achieved by the cluster was almost 390%. The actual line was a curve which was expected as more workers results in more data transfers and hence reduced performance.

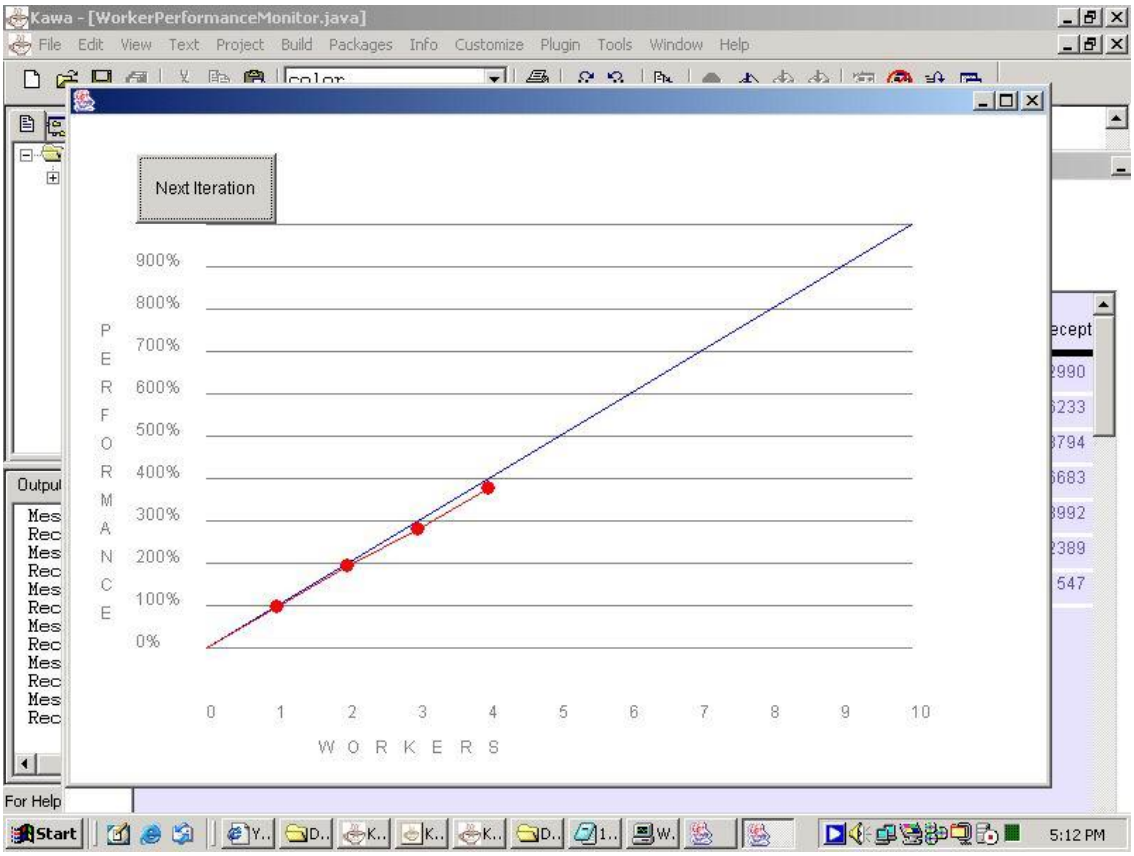


Figure 9.3 Worker performance monitor results

9.4.2 Data Performance Monitor

The results by data performance monitor were also found to be close to the expected. The bytes encrypted per second for 1MB, 2MB and 3MB data sets shows minimum variation. This means that data handling by the system does not vary with different data sets. The initial performance of 1MB also includes remote code transfer and registration. This shows that code registration also takes minimum time and may not result in significant performance loss.

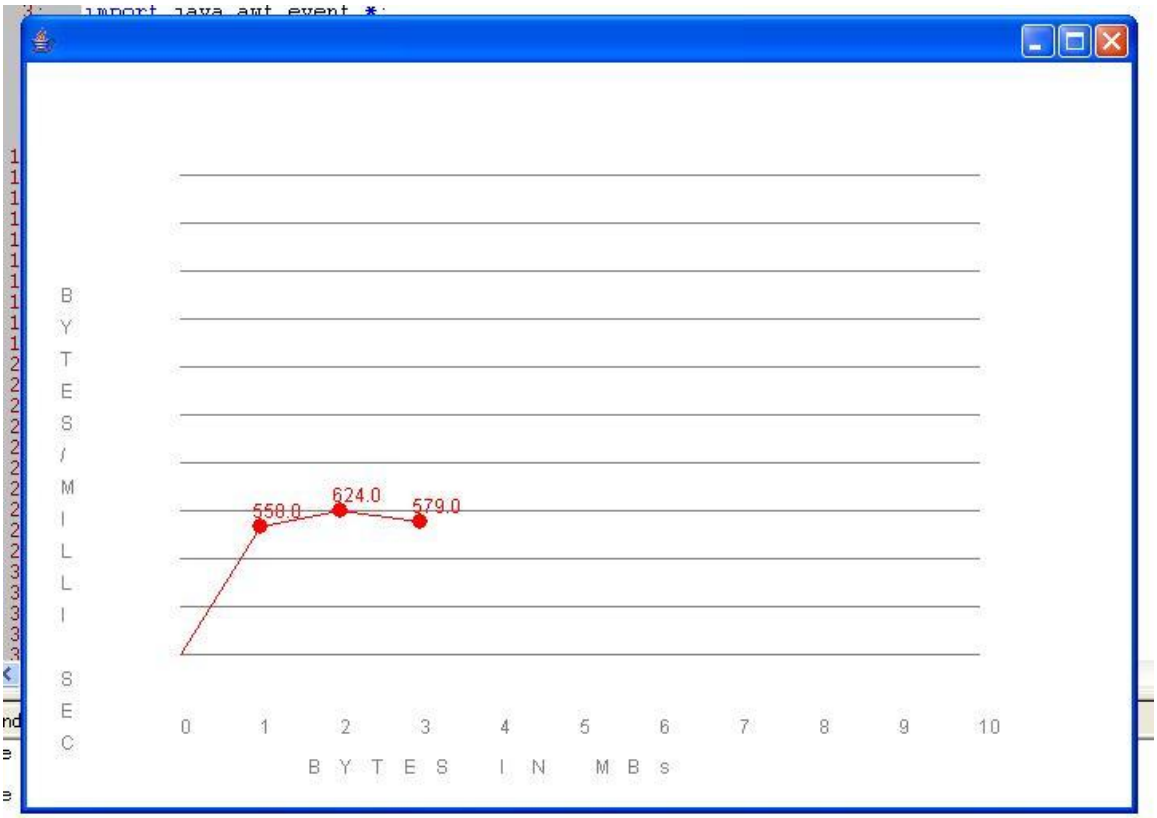


Figure 9.4 Data performance monitor results

Chapter 10

FUTURE ENHANCMENTS

10.1 DATABASE FOR PERSISTENT STORAGE

Currently all persistent data storage is done through files. The stored data includes user information such as passwords and keys, client job requests and their results. This job requests and results are added and deleted all the time as old jobs get processed and returned and new jobs come into the system. After implementing this storage through files I have realized that the same job can be done through use of a database. The advantages of this approach include more reliability and efficiency. Implementation will also be easier and more manageable if done through a database.

10.2 CONFIGURATION GUIs

Currently all the configuration of the system is hard coded and the files have to be recompiled after every change. Configurable values in the whole system are more than 50 at the moment and will increase if the system is further developed so a proper GUI becomes even more important.

10.3 ADVANCED JOB DEPENDENCY OPTIMIZATIONS

As already explained the current system uses turn by turn scheduling to implement the job dependency optimizations which results in reduction of network message but further optimizations can be done. The basic mechanisms for implementing these advanced optimizations have already been mentioned in the report.

Chapter 11

CONCLUSION

Java dynamic clustering platform provides opportunities for research and development of clustering systems that use technologies that are going to be widely used in the near future. Java technology allows platform independence and control over the resources which is essential in today's heterogeneous environment. A new programming model for client applications can also be explored and developed further as a research project. The system also supports real-time jobs which are not supported by many other clustering systems hence the platform can be used in specific situations especially where GUI based applications have to run faster.

After the completion of the project JDCP system has evolved from a simple distributed computing system to a complete clustering platform. Now the system provides not only distributed computing but also allows management of resources, fault tolerance, security and information services in addition to the programming specific enhancements like job dependency optimizations and batch job support. Schedulers have been added into the system as plug-ins. This allows future developers to extend the system and test new scheduling algorithms.

Testing under various workloads and resources has to be done to properly determine the efficiency of the system and scheduling algorithms designed during the project. Database usage is necessary so convert the system into a stable environment. Large data loads are not currently handled by the system which can be corrected by changing the message passing architecture of the system.

REFERRENCES

[1] URL: http://wwws.sun.com/software/gridware/

[2] URL: http://www.globus.org/

[3] URL: http://www.lri.fr/~fedak/XtremWeb/introduction.php3

[4] URL: http://icl.cs.utk.edu/netsolve/

[5] URL: http://java.sun.com/products/jce/

APPENDIX A – USER MANUAL

INSTALLATION

The only software requirement is to have JRE 1.4 or later installed on the PC running the JDCP system. The OS supported include Windows 2000 or later or Linux 9.0 or later. No installation needed for client and information server. During worker installation change server IP in the file WorkerController.java (line 66) and WorkerThread.java (line 208) from local host to actual server IP and recompile. During server installation change information server IP in Advertiser.java and recompile.

RUNNING THE SYSTEM

Put server folder at any location on the hard disk and run it through java.exe with current directory included in the class path. Do the same with information server and worker. Multiple workers can be connected to the server. The server screen will now show the workers connected to the server and ready for processing. After this is complete run any valid client program. Client program can be compiled and run with the distribution handler folder included in the class path.

CLIENT API

The JDCP client programs have to use a library provided with the system in order to connect to a server and get its jobs processed. The library includes all the functionalities that a client programs needs to use the JDCP system. The library

provides an API for the client programs. The following methods are present in the client API

Method	*Description*
void setServer(String sIP,int port)	Sets server connectivity information
void connect(int timeout) throws Exception	Called after the setServer method to actually connect to the JDCP server
int getID()	Returns the ID that the server assigned to this client instance
void sendFile(String fileName,String filePath)	Sends a file to the server. This includes class files and other file.
int executeMethod(MethodRequest mr)	Executes one job asynchronously
Object returnResult(int token)	Returns the result of the job with this token
Object executeMethodBlocking (MethodRequest mr) throws Exception	Executes a blocking job call
int[] executeMethods (MethodRequest[] requests)	Executes multiple methods asynchronously
boolean executeMethodsBlocking (Object[] tbr,MethodRequest[] requests) throws Exception	Executes multiple methods. This method blocks till all the results have been received
int executeMethodBatch (MethodRequest mr)	Executes a batch job at the server

void getBatchResult (int token)	Gets the result of a batch job from the server
int[] executeBundle (JobRequestBundle bundle)	Executes a bundle of jobs at the server asynchronously
void setSecurityLevel (SecurityInfo si)	Sets the security level that the user program wants to connect at. The actual connection is at the discretion of the server.

STEPS IN CREATING A SIMPLE CLIENT PROGRAM

Following steps have to be taken by the client program to call one method request remotely on the JDCP server assuming that the system is running.

- Create a DistributionHandler() object
- Pass server info containing IP and port to the distribution handler by calling the method setServer(String IP, int port)
- Connect to the JDCP server by calling the connect() method
- Send the class file of the class whose method you want to call remotely. This is done through calling the distribution handler function void sendFile(String fileName,String filePath).
- Create a MethodRequest object and pass the constructor with the required information
- Call executeMethodBlocking(MethodRequest request) of the connected distribution handler to pass the request to the server and get the result back as an object returned by the method

SAMPLE CLIENT PROGRAM

The following client program runs one method remotely on the JDCP server and returns the result. All the components of the platform except the information server have to ve3 running for this program to run correctly.

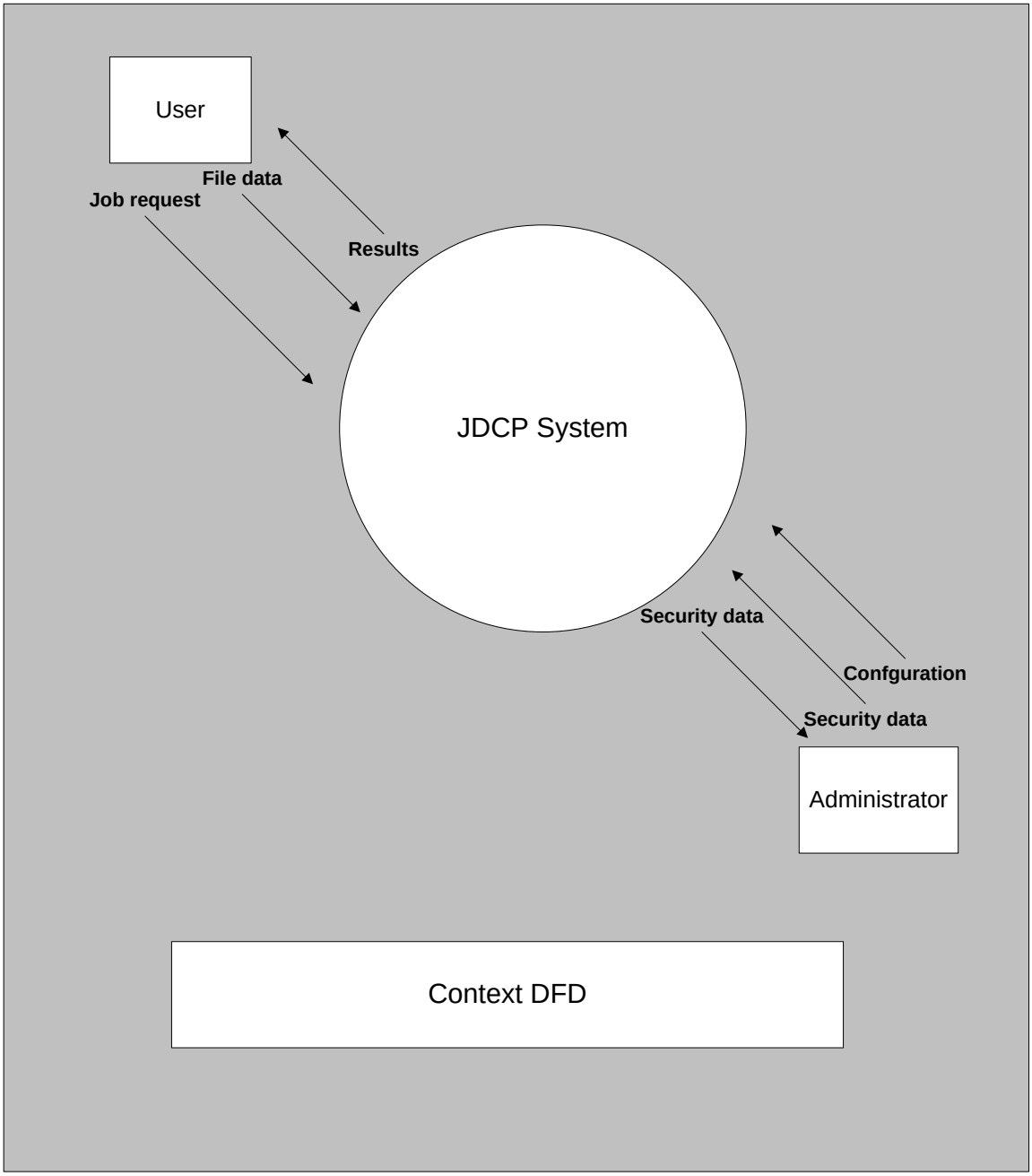
```
Class SimpleClient
{
    public static void main(String a[])
    {
            DistriubutionHandler dh=new DistributionHandler();
            dh.setServer("localhost",5001);
            dh.connect(10000);
            dh.sendFile("ABC.class","h:/Ali/ABC.class");
            MethodRequest mr=new
            MethodRequest(1,"ABC","AMethod",null,null);
            Object result=Dh.executeMethodBlocking(mr);
    }
}
```

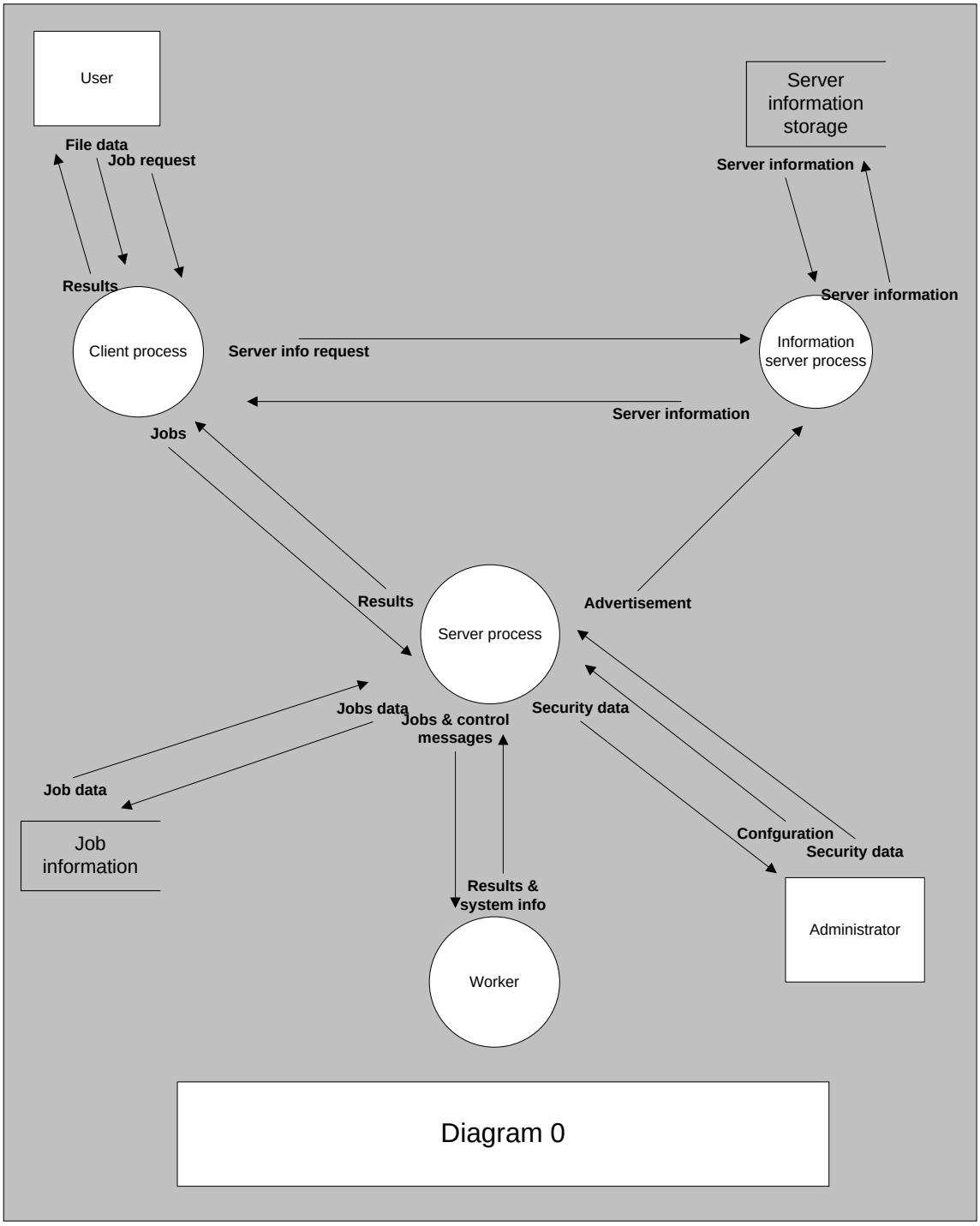
USING INFORMATION SERVER

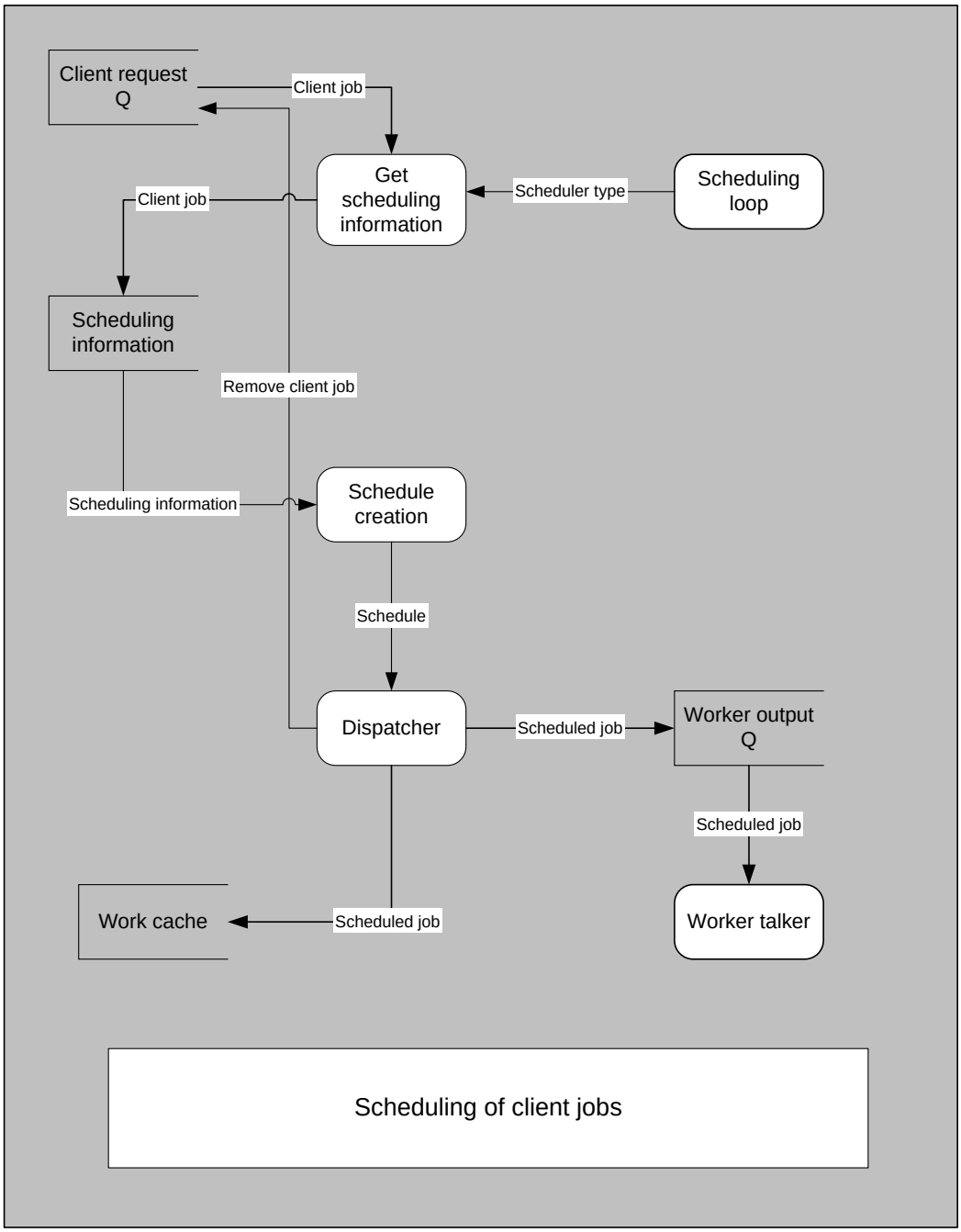
The client can connect to the information server and get information about currently available server. This is done all through an object of the DiscoveryHandler class. The following API can be used by the client to use the information server.

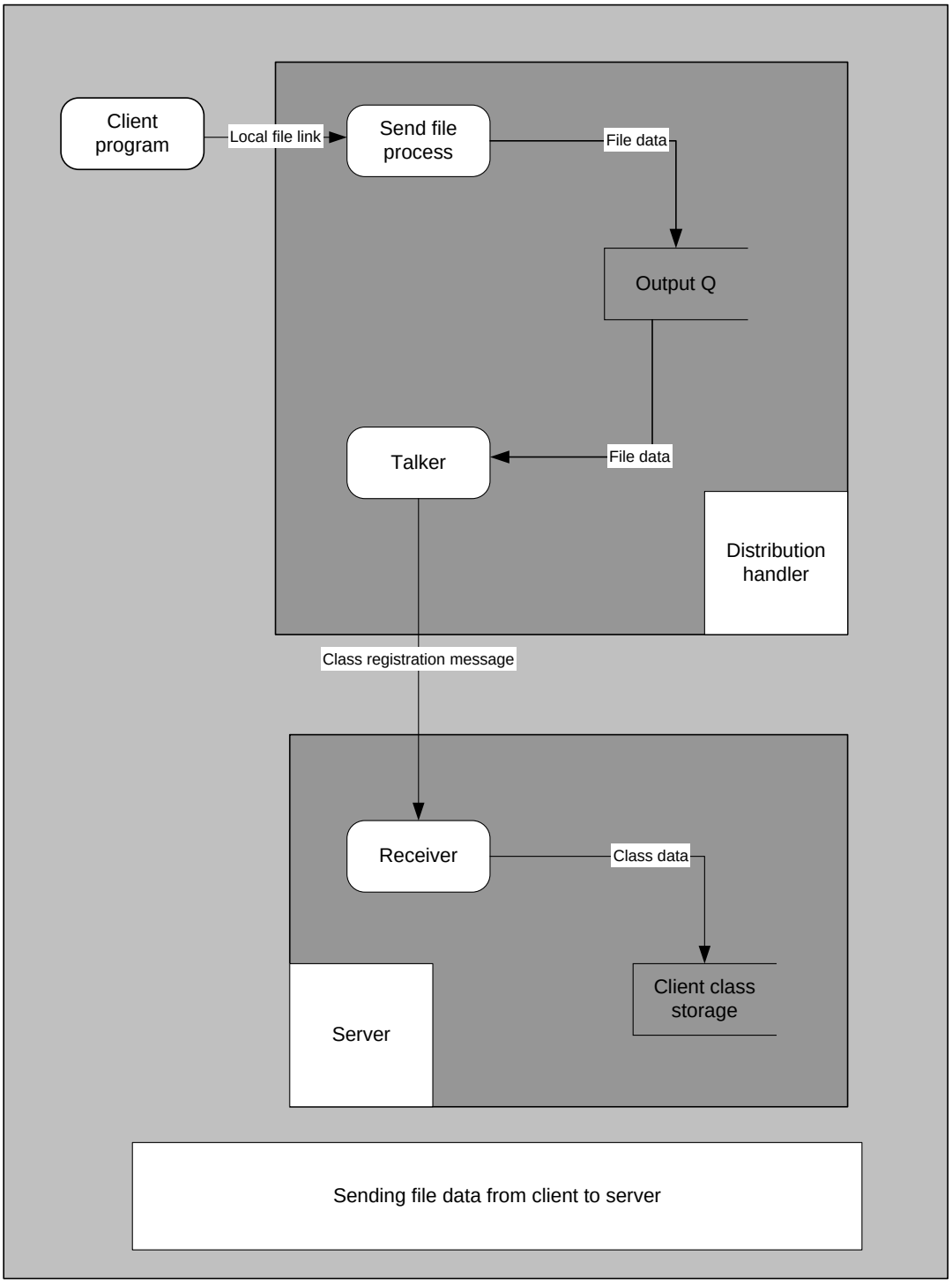
API Method	*Description*
DiscoveryHandler()	Creates a new discovery handler
void setInformationServer(String ip,int p)	Sets the connectivity information to the information server
boolean connect()	Actually connects to the information server
String[] getAvailableServers()	Gets names of all the available servers
ServerInfo getServerInfo(String serverName)	Gets complete information about the server whose name is passed as argument
ServerInfo[] getAllServerInfo()	Gets complete information about all servers

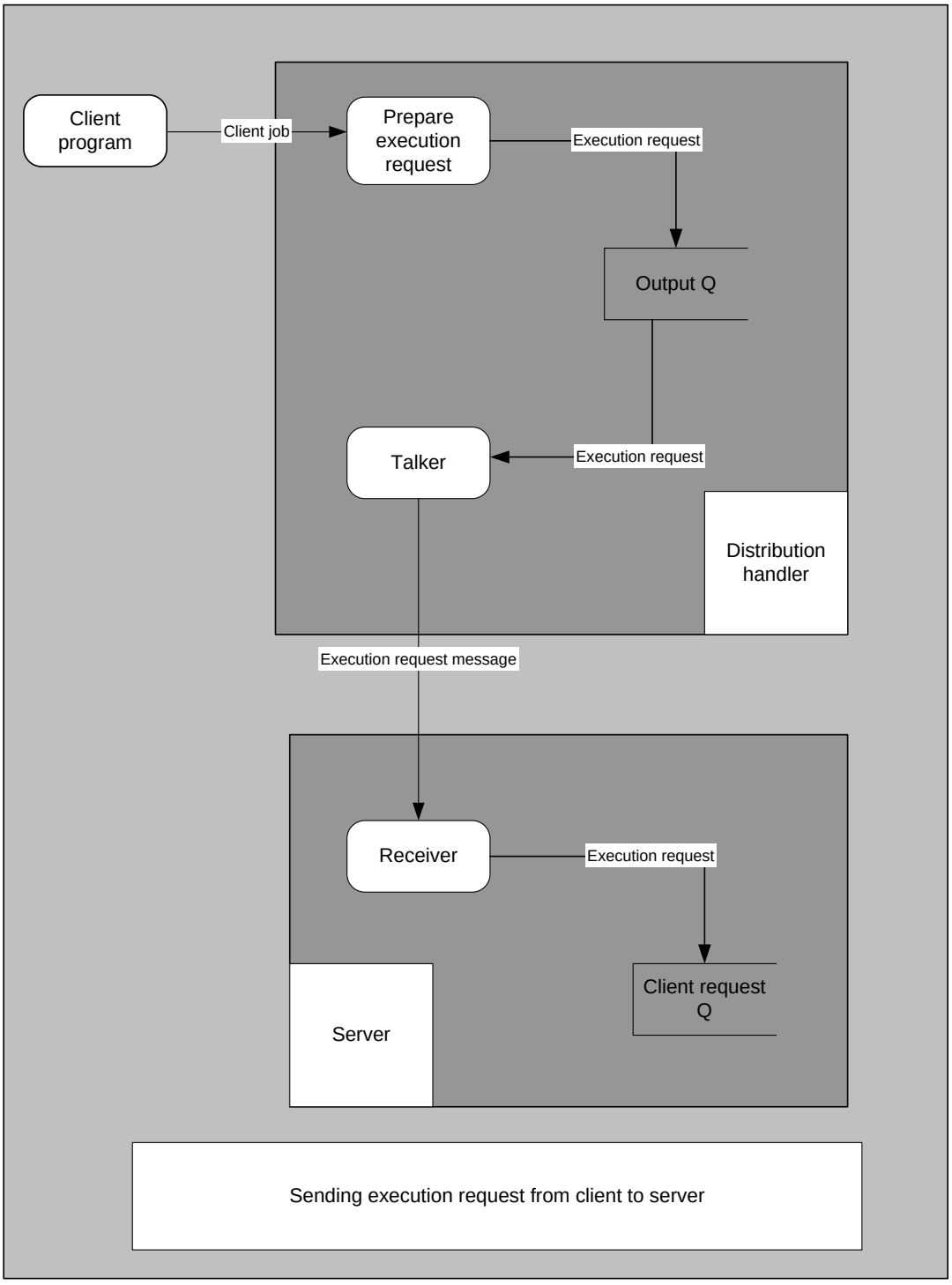
APPENDIX B – ARCHITECTURAL DIAGRAMS

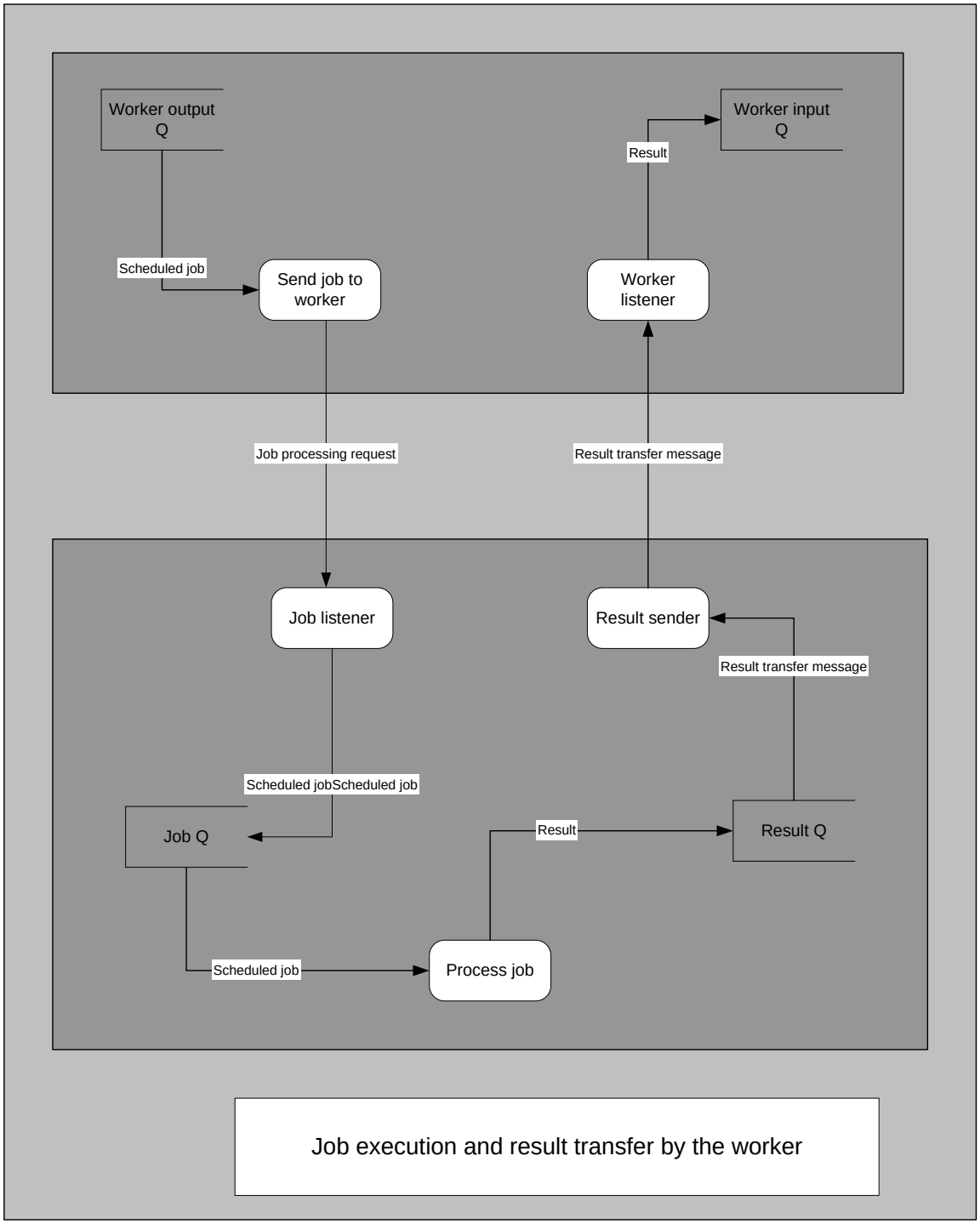


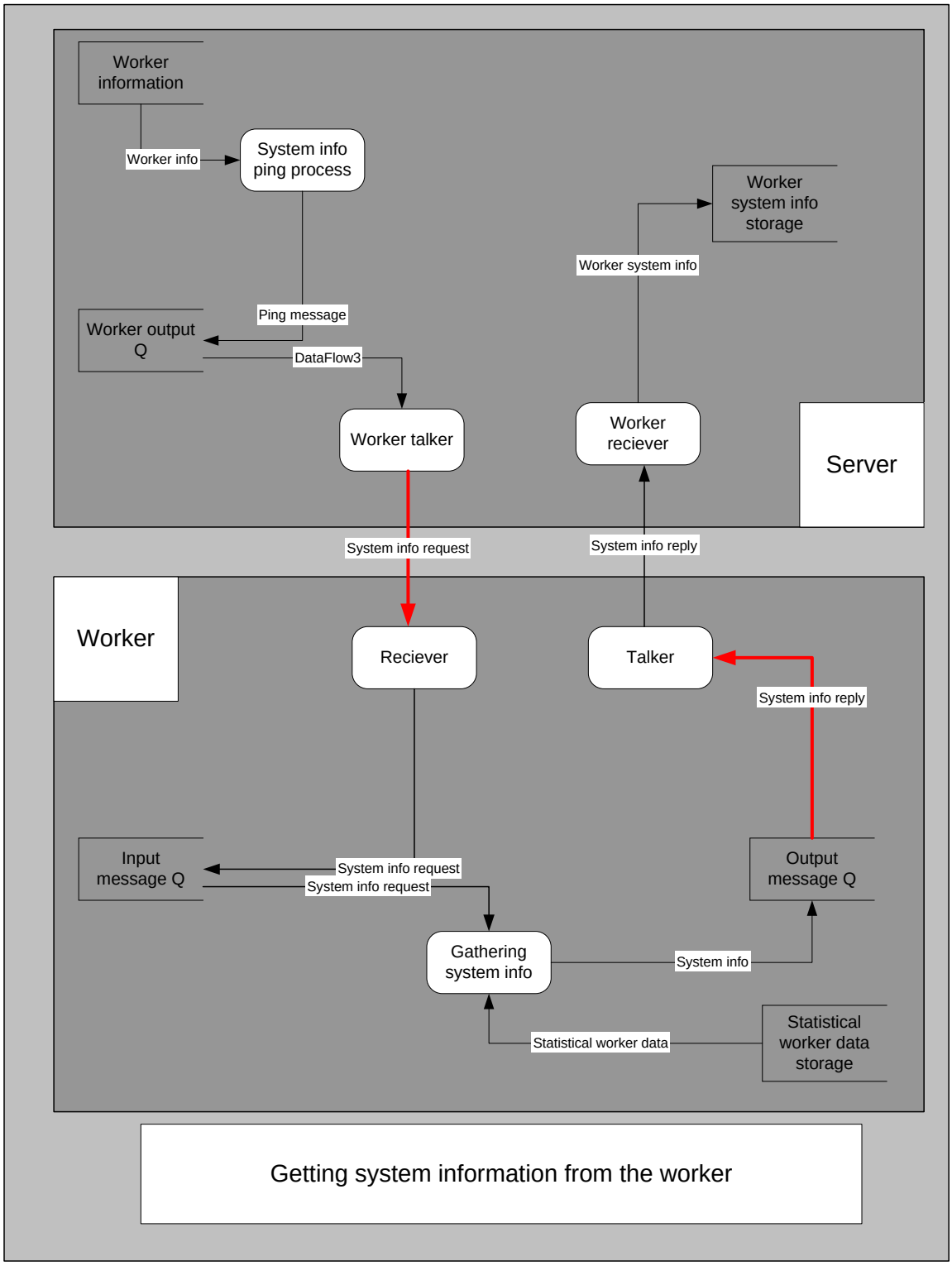

User
Server information storage
File data
Job request
Server information
Results
Server information
Client process
Server info request
Information server process
Server information
Jobs
Results
Server process
Advertisement
Jobs data
Jobs & control messages
Security data
Job data
Job information
Confguration
Security data
Results & system info
Administrator
Worker
Diagram 0

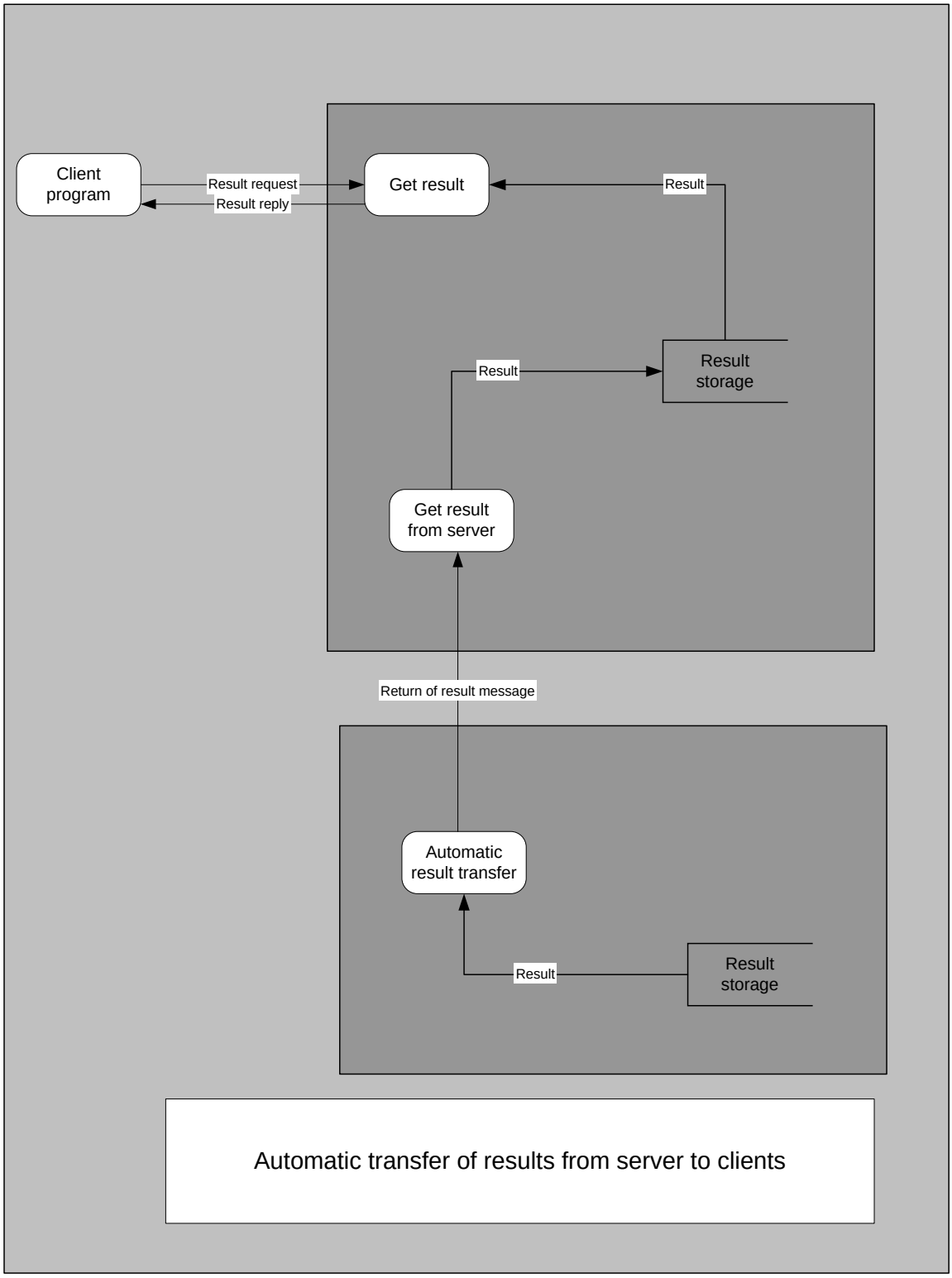

Client request Q
Client job
Get scheduling information
Scheduler type
Scheduling loop
Client job
Scheduling information
Remove client job
Scheduling information
Schedule creation
Schedule
Dispatcher
Scheduled job
Worker output Q
Scheduled job
Work cache
Scheduled job
Worker talker
Scheduling of client jobs

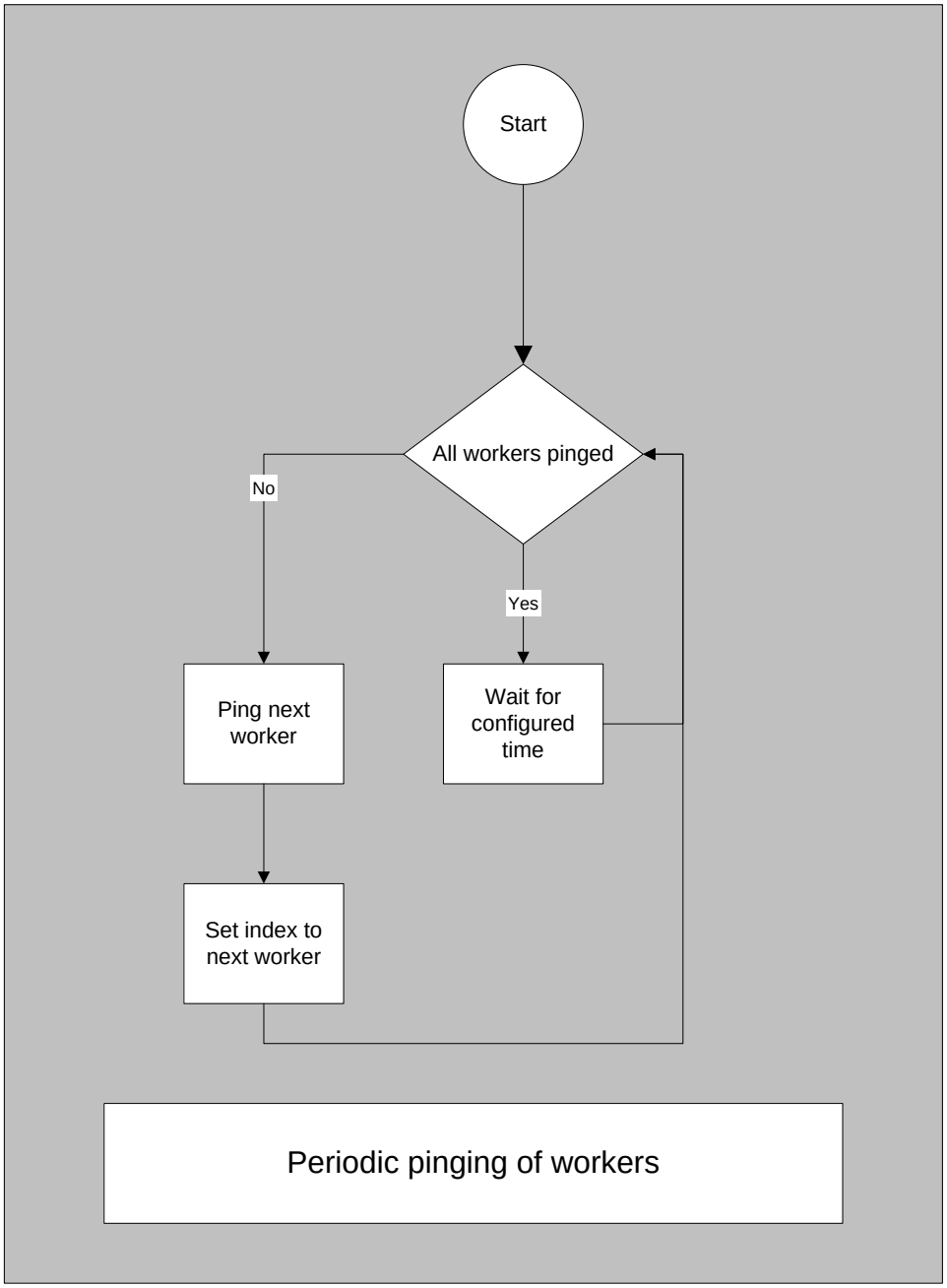

Client program
Local file link
Send file process
File data
Output Q
Talker
File data
Distribution handler
Class registration message
Receiver
Class data
Client class storage
Server
Sending file data from client to server


Client program
Client job
Prepare execution request
Execution request
Output Q
Talker
Execution request
Distribution handler
Execution request message
Receiver
Execution request
Client request Q
Server
Sending execution request from client to server


Worker output Q
Worker input Q
Result
Scheduled job
Send job to worker
Worker listener
Job processing request
Result transfer message
Job listener
Result sender
Result transfer message
Scheduled jobScheduled job
Job Q
Result
Result Q
Scheduled job
Process job
Job execution and result transfer by the worker


Worker information
Worker info
System info ping process
Worker system info storage
Worker system info
Ping message
Worker output Q
DataFlow3
Worker talker
Worker reciever
Server
System info request
System info reply
Worker
Reciever
Talker
System info reply
Input message Q
System info request
System info request
Output message Q
Gathering system info
System info
Statistical worker data
Statistical worker data storage
Getting system information from the worker


Client program
Result request
Result reply
Get result
Result
Result storage
Result
Get result from server
Return of result message
Automatic result transfer
Result
Result storage
Automatic transfer of results from server to clients


Start
All workers pinged
No
Yes
Ping next worker
Wait for configured time
Set index to next worker
Periodic pinging of workers