# A Task Allocation Protocol For Real-time Financial Data Mining System

**LAM Lui-fuk**

A Thesis Submitted in Partial Fulfilment

of the Requirements for the Degree of

Master of Philosophy

in

Information Engineering

The Chinese University of Hong Kong

July 2003

# Abstract

By employing client/server architecture for developing a Real-time Financial Data Mining system, we can increase the usability, flexibility, interoperability, and scalability of the system when compare with centralized, mainframe, or file-sharing computing. To further improve the performance and capacity of the Real-time Financial Data Mining system, we can upgrade the hardware of the server or enable distributed computing in the system. Enabling distributed computing is a promising direction since it facilitates the use of client computers (in low CPU usage) as a new computing power of the system. To implement such distributed computing system, we need a task allocation protocol for allocation of tasks among client computers and servers.

In this thesis, we will explore the potential of the Contract Net Protocol (CNP) [1] [2] as a dynamic task allocation protocol for the Real-time Financial Data Mining system by focusing on modifying the CNP to make it most suitable for the Real-time Financial Data Mining system. And we will design a task allocation method using the modified CNP for the on-line data analysis service of the Real-time Financial Data Mining system.

# 摘要

憑著使用客戶機/伺服器系統結構來開發一個實時財務資料開採系統，我們能增加該系統與集中計算、電腦主機計算、或文件分享計算比較時的實用性、靈活性、共用，和可測量性。 爲進一步改進實時財務資料開採系統的表現和容量, 我們可以把伺服器的硬體升級或使用分配計算技術。 使用分配計算是一個有發爲的方向因爲它促進使用客戶電腦在低中央處理器用量時作爲系統的計算力量。 實施這樣的分配計算系統，我們需要一個任務分派協議爲協調任務在客戶電腦和伺服器之中的分派。

在這份論文, 我們將探索合同網協議 [ 1 ] [ 2 ]作爲一個動態任務分派協議爲實時財務資料開採系統所使用的潛力，我們將集中討論修改合同網協議使它最適當配合實時財務資料開採系統的運用。 與此同時我們將爲實時財務資料開採系統的網上分析服務設計一個任務分派方法,它將會使用修改過的合同網協議作爲分派協議。

# Acknowledgement

I would like to express my sincere gratitude to my supervisor, Professor Tony T. Lee for his continuous support, feasible research guidance and directions on this research. This thesis is completed with his professional advice on technical contents and thesis writing. I would also like to express my thanks to Professor Jack Y.B. Lee for his invaluable advice in designing the system architecture.

I would also like to thank all the members in broadband lab in department of Information Engineering: Mr. Wong Tse Chung and Mr. Li Chi Ming for their comments and continuous discussion on my research work. Mr. Y. Deng, Mr. S.W. Mui, Mr. W.H. Man, Mr. M.T. Choy and Mr. L. Zhang for their various kinds of help during these two years.

Finally, I am grateful to my family and my friends for their morally supporting throughout all these years.

# Table of Contents

# List of Figures

# List of Abbreviations

| | | |
|---|---|---|
| CPU | : | Central Processing Unit |
| CNP | : | Contract Net Protocol |
| PC | : | Personal Computer |
| MCNP | : | Modified Contract Net Protocol |
| FIPA | : | The Foundation For Intelligent Physical Agent |
| CFP | : | Call For Proposals |
| HTML | : | Hyper Text Mark Up Language |
| PDA | : | Personal Digital Assistant |
| RETSINA | : | The Reusable Environment for Task-Structured Intelligent Networked Agents Architecture |
| HTN | : | Hierarchical Task Network |
| SR | : | Services Request |
| GUI | : | Graphical User Interface |
| UDP | : | User Datagram Protocol |
| UML | : | Unified Modeling Language |
| LMQ | : | Local Message Queue |
| EMQ | : | External Message Queue |
| CASS | : | Client AS Server |
| M/M/1 | : | Single Server Exponential Queuing System |
| MBS | : | Mega Bits per Second |

# Chapter 1

# Introduction

## 1.1 Introduction

The current stock markets are more competitive and dynamic than ever. With the recent advances in the Internet and computer technologies, large amount of real-time stock market information can be obtained from the Internet. The Internet coupled with the tendency of globalization of world stock markets results in the present demand and production of more information accessible in a shorter amount of time. Since information is produced in greater quantity and more rapidly than it is consumed, the Real-time Financial Data Mining system that is capable of using user-designed algorithm to analyze and synthesize information in short time has attracted much attention from investors.

The success of a Real-time Financial Data Mining System is closely related to the sophistication and speed of the system, and its ability to analyze and synthesize information with simple operations by users. Therefore, a super-computing machine is required for the system. However, very few clients have machines that can meet the requirements of running such a sophisticated system at high speed. Accordingly, the number of potential users is restricted. In software engineering, client/server architecture is a common approach to deal with this problem, in that the server provides the same computing power to

analyze and synthesize information at a high speed through network connection for all of the clients with different computing competence. Therefore, without a super-computing machine, people can still enjoy the super-computing power. Client/server architecture also improves usability, flexibility, interoperability, and scalability as compared to centralized, mainframe, file-sharing computing.

On the other hand, however, the PC client software in a client/server based financial data mining system is, during most of the time, either interacting with user, or waiting for some conditions to trigger the data mining action, and the server does most of the computing tasks. Therefore, the PC client software will have low CPU usage (or in idle period) for a long period of time. Although client's PC cannot run the whole system in high performance throughout the whole time, the computing power provided by combining large number of clients PC can be even greater than that of a single server. Hence, a complete centralized system cannot fully utilize the resources of the system. To further improve the performance of the system, we can install distributed computing in the system. Enabling distributed computing has a great potential since it facilitates the use of client's computers (in low CPU usage) as a computing power of the system. Thus, when the number of PC clients increases in the system, not only the work loading of the system increases, but the computing power (resources) provided by clients also increases accordingly. Therefore, a Real-time Financial Data Mining system employing client/server architecture integrated with distributed computing is a promising system design.

## 1.2. Motivation and Research Objective

To enable distributed computing for the real-time financial data mining system, we need a task allocation protocol to facilitate tasks allocation among client computers and server(s). We decide to explore the potential of the Contract Net Protocol (CNP) [1] [2] as an efficient and simple task allocation protocol for the real-time financial data mining system. The CNP is modeled on the contracting mechanism used by business companies to govern the exchange of goods and services. It is designed for multi-agent system and has been widely used in multi-agent system. To make CNP possible for our system, we need to do some modifications on the protocol, and overcome some limitations of the CNP. Therefore, in the following sections we will focus on modifying the CNP to make it suitable for the real-time financial data mining system. And we will also design a task allocation method using the modified CNP for the on-line data analysis service system of the real-time financial data mining system.

## 1.3. Organization of the Dissertation

The remainder of this dissertation is organized as follows. In Chapter two, we will introduce the history of Contract Net Protocol and its common usage in computer science, the two-tier software architecture, and three-tier software architecture. Chapter three will be devoted to an overview of the design goal and system architecture advantages of the Real-time Financial Data mining system. We will discuss the software architecture of the server and client's application in the Real-time Financial Data Mining system, and the necessary

components for client's application to enable distributed computing through the use of a task allocation protocol in chapter four. The advantages of the software architecture will also be introduced.

Chapter five focuses on the Foundation For Intelligent Physical Agent (FIPA) Contract Net Interaction Protocol (IP) [3] -- its strengths and limitations. We will discuss the modifications of the FIPA contract net IP to suit our system, and its strengths and limitations. Outline the implementation of the Modified CNP.

Chapter six will design a system model enabled by the Modified Contract Net Protocol (MCNP) for the on-line data analysis service of the Real-time Financial Data Mining system. We compare the system capacity increased by the new system model with a single-server exponential queueing system through performance analysis and simulation. Thus showing a task allocation method combine with the task allocation protocol (MCNP) can enable the potential benefit of employing distributed computing.

Finally, chapter seven has the review, contribution and the future work.

# Chapter 2

# Background Studies

## 2.1 The Contract Net Protocol

The contract net protocol is an interaction protocol for cooperative problem solving among artificial intelligence agents [4]. It is modeled on the contracting mechanism used by businesses to govern the exchange of goods and services. The contract net provides a solution for finding an appropriate agent to work on a given task. Figure 2.1 illustrates the basic steps in this protocol.

In relation to an individual task, each agent may take on one of two roles:

- Manager
- Contractor

From a manager's view, the process is

1) To call for proposal for a task that needs to be performed
2) To receive and evaluate bids from contractors
3) To award a contract to a suitable contractor
4) To receive and synthesize results.

From a contractor's view, the process is

1) To receive call for proposals message
2) To evaluate my availability to respond

3) To respond (decline, bid)

4) To perform the task if my bid is accepted

5) To report the results.

There is no restriction on the role of agents. Any agent can be manager and contractor at the same time for different contracts. This flexibility allows a contractor for a specific task to become a manager by soliciting the help of other agents in solving parts of that task. Therefore, the agents form a control hierarchy of manager-contractor links [4] for task sharing and result synthesis.

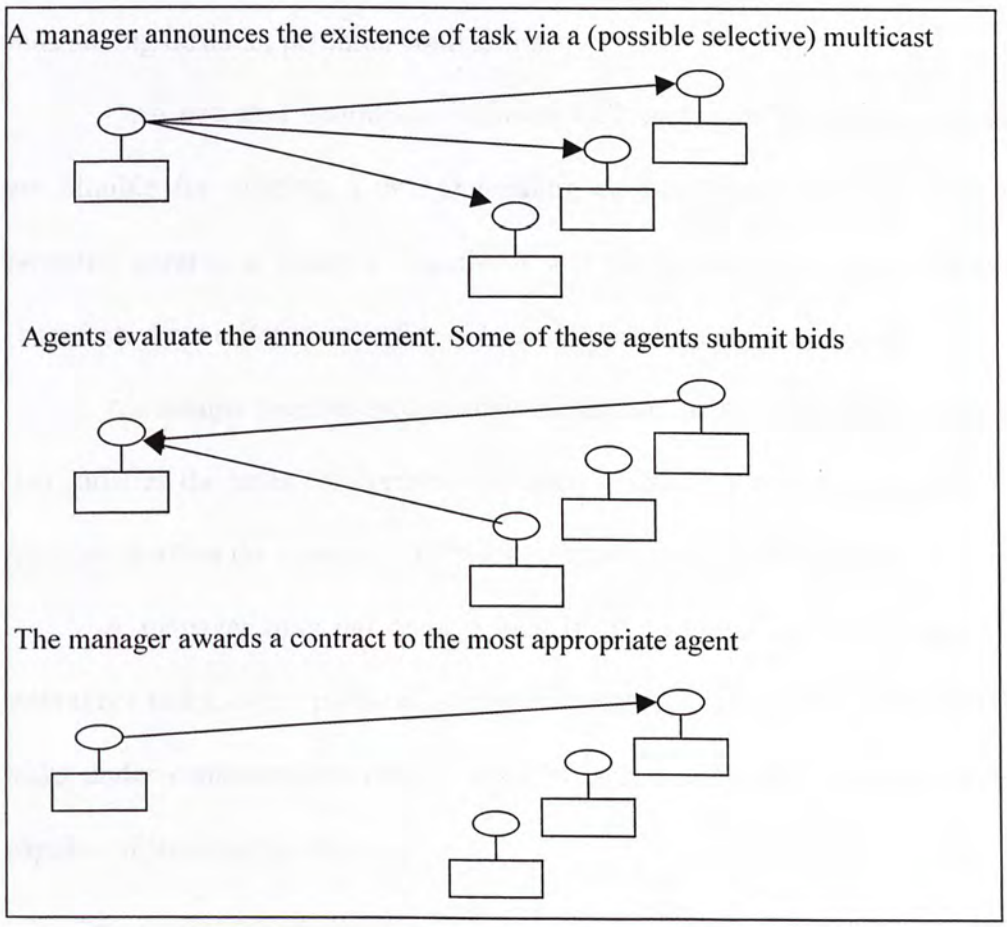


Figure 2.1: The basic steps in the contract net, an important generic protocol interactions among cooperative artificial intelligence agents. [4]

The manager can seek other contractors to perform the task, if the awarded contractor is unable to provide a satisfactory result.

The Call for Proposals (CFP) message includes the fields for *addressee, eligibility specification, task abstraction,* and *bid specification* [4]. The CFP message may be sent to one or more potential contractors who may meet the criteria of the eligibility specification. The task abstraction is a brief description of the task, and may be used by contractors to rank tasks from several CFP messages. The bid specification indicates what information must be provided with the bid; returned bid specification gives the manager a basis for comparing bids among different potential contractors.

Each potential contractor evaluates CFP messages to determine if they are eligible for offering a bid. Depending on how many external tasks the potential contractor wants to handle, it will choose the most attractive tasks (based on some criteria) and offer bids to the corresponding managers.

A manager receives and evaluates bids for each CFP message. Any bid that satisfies the basic requirement indicated in the CNP may be accepted. The manager notifies the contractor of bid acceptance with an award message.

A manager may not receive bids if all potential contractors are busy with other tasks, or the potential contractors rank the proposed task below other tasks under consideration. Even if some contractors are idle, they may not be capable of working on the task.

## 2.2 Two-tier software architectures

Two-tier software architectures were developed in the 1980s from the file server software architecture design. The two-tier architecture is intended to improve *usability* by supporting a forms-based, user-friendly interface. The two-tier architecture improves *scalability* by accommodating up to 100 users (file server architectures only accommodate a dozen users), and improves *flexibility* by allowing data to be shared, usually within a homogeneous environment [5]. The two-tier architecture requires minimal operator intervention, and is frequently used in non-complex, non-time critical information processing systems. Detailed readings on two-tier architectures can be found in Schussel and Edelstein [5] [6].

Two-tier architectures consist of three components distributed in two layers: client (services requester) and server (services provider). The three components are User System Interface, Processing Management and Database Management.

In two-tier architecture, the clients assume the responsibility for the application logic, while the server assumes the responsibility for data integrity checks, query capabilities, data extraction and most of the data intensive tasks, including sending the appropriate data to the appropriate clients. SQL is a standard language used on the clients to request appropriate subsets of data from the server. Data returned from the server to the clients is processed by the client software for reporting and business analysis.

The development time of two-tier systems is short, and many robust tools are available for fast prototyping.

## 2.3 Three-tier software architecture

The three-tier software architecture emerged in the 1990s to overcome the limitations of the two-tier architecture. The third-tier (middle tier server) is between the user interface (client) and the data management (server) components. This middle tier provides process management where business logic and rules are executed and can accommodate hundreds of users (as compared to only 100 users with the two-tier architecture) by providing functions such as queuing, application execution, and database staging. The three-tier architecture is used when an effective distributed client/server design is needed to provide (when compared to the two-tier) increased performance, flexibility, maintainability, reusability, and scalability, while hiding the complexity of distributed processing from the user. For detailed information on three-tier architectures, see Schussel and Eckerson. Schussel provides a graphical history of the evolution of client/server architectures [5] [8].

A three-tier distributed client/server architecture (as shown in Figure 2.2) includes a user system interface top tier where user services reside.

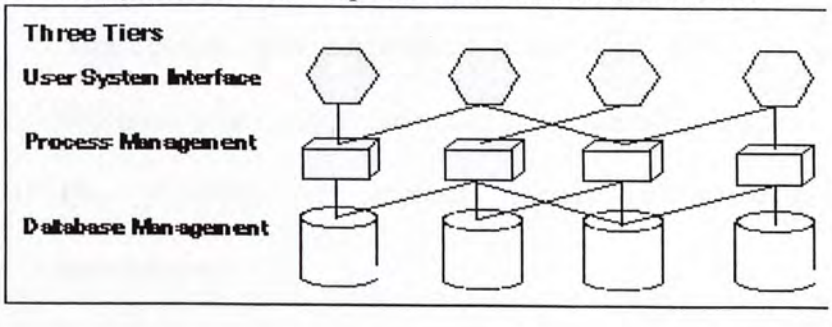


Figure 2.2: Three-tier distributed client/server architecture depiction [7]

Systems using this three-tier architecture are often called middleware systems. The definition of the term ``middleware" according to Webopedia [9] is the following:

1) Software that connects two otherwise separate applications. For example, there are a number of middleware products that link a database system to a Web server. This allows users to request data from the database using forms displayed on a Web browser, and it enables the Web server to return dynamic Web pages based on the user's requests and profile.
2) The term middleware is used to describe separate products that serve as the glue between two applications. It is, therefore, distinct from import and export features that may be built into one of the applications. Middleware is sometimes called plumbing because it connects two sides of an application and passes data between them.

According to Webopedia [9], the most important advantages of the middleware approach are the following:

1) The desire to use open services and protocols.
2) The wish to re-deploy logic at will and unconstrained by infrastructure; this necessitates using open APIs and protocols, which are widely supported across most infrastructure products.
3) The necessity to support cooperating mixed-architecture applications.

4) The urge to move network and service infrastructure decisions out of application space, so that system managers can make infrastructure decisions without being hampered by applications that depend on proprietary protocols or features.

Sometimes, the middle tier is divided in two or more units with different functions, in these cases the architecture is often referred to as multi layer. This is the case, for example, of some Internet applications. These applications typically have light clients written in HTML and application servers written in C++ or Java, the gap between these two layers is too big to link them together. Instead, there is an intermediate layer (web server) implemented in a scripting language. This layer receives requests from the Internet clients and generates html using the services provided by the business layer. This additional layer provides further isolation between the application layout and the application logic.

# Chapter 3

# System Architecture

## 3.1 Introduction

The objectives of our system design are to provide a flexible, comprehensive and real-time platform for the investors to carry out stock market analysis. By enabling distributed computing, we facilitate the use of client computers (in low CPU usage) as a computing power of the system for providing a super-computing environment for investors to perform complex data analysis. Investors can get more information about the market by custom query service, standard technical analysis and customer-define technical analysis and the financial news analysis. All these tools help them to obtain reference materials for their investments. Through the application, investors can acquire a more thorough understanding of the stock market by the simple and user-friendly interface. They can have better prediction of the market and thereby more risk control. In other words, they can have a more effective use of their time and money.

## 3.2 System Architecture Overview

The System Architecture is basically a three-tier structure. The front layer is the client while the middle layer is the Application Servers. The back-end layer

contains the database and data sources. Figure 3.1 shows the system architecture.

### 3.2.1 Client Layer

This layer is responsible for data presentation. It provides visualized access to all system services through user-friendly graphical interface. User running the client's application on personal computer will provide part of the on-line data analysis functions, which helps us to enable distributed computing for the whole system

### 3.2.2 Middle Layer

This layer encapsulates the business logic of the system. The on-line data analysis server performs data processing, data analysis and information syntheses; the user management server manages system connection security, user login verification and registration, and user's profile management; the real-time data streaming server streams the real-time data to all on-line clients; the data management server manages the access to the database that allows clients to use standard or user-define query to retrieve data from the database and provides real-time data alerting services. Since the middle layer is connected directly to the database and data sources, it acts as a bridge for the client to access to the valuable data, and also as the platform for client to perform complex data analysis. Thus, the system is highly manageable and scalable.

### 3.2.3 Back-end Layer

The database in this layer contains all the data of stocks, such as the real-time and historical stock prices, options information, companies information, financial news and user profiles. There are data sources in this layer providing the real-time stock data to our middle layer, and the real-time data will then be stored into our database for future usage.

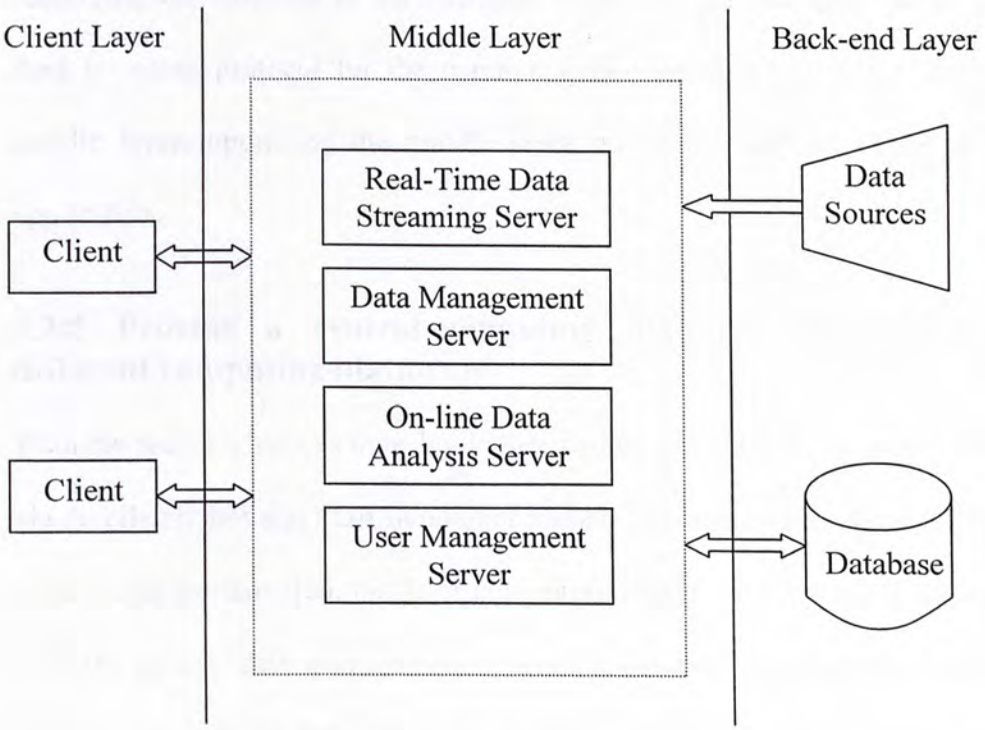


Figure 3.1: System Architecture

## 3.3 Advantages of the System Architecture

### 3.3.1 Separate the presentation components, business logic and data storage

The three-tier system separates the presentation components, business logic and data access into three distinct entities. Our system is suitable for three-tier

architecture, as the presentation layer is necessary to be separated for clients to gain access to our system in different places using different platforms. The business logic and data components are divided up as a client-server application. Hence, the number of clients being able to access the database will not be restricted by the database specification, though normally there are no more than 100 users at a time. In three-tier architecture, changing the data sources and modifying the database in the back-end layer will not affect the client layer. And by using protocol for the communication between the client layer and middle layer, upgrading the middle layer server will not affect the client's application.

### 3.3.2 Provide a central-computing platform for user using different computing platforms

With the recent advances in technologies, people can gain access to the Internet via mobile phones and PDA in outdoor places. These electronic devices provide good visual presentation, but their computing power is very limited. However, with the on-line data analysis server in the middle layer, users using different devices can still get the same data analysis services from the server, for the clients with more restricted computing platform will only need to forward the data analysis requests to the server and present the analysis result to the users.

### 3.3.3 Improve system capacity

By employing the three-tier architecture, the middle layer servers provide process management where business logic and rules are executed and can accommodate hundreds of users, as opposed to only 100 users with the two-tier

architecture, by providing functions such as queuing, application execution, and database staging.

### 3.3.4 Enable distributed computing

The client's application for personal computer (PC) is integrated with simple data analysis functions and task allocation protocol. Therefore, the PC client's application can choose to run data analysis by itself or use the task allocation protocol to allocate the data analysis tasks to other PC client's applications (in low CPU usage) or the on-line data analysis server, in which the data management server provides the necessary data for performing the analysis. Thus when a PC client's application gets a large amount of data analysis tasks from user within a short time, they can share the tasks with other PC client's application and the middle layer server, enabling distributed and parallel computing instead of using sequential computing.

# Chapter 4

# Software Architecture

## 4.1 Introduction

We found the Reusable Environment for Task-Structured Intelligent Networked Agents (RETSINA) agent architecture [10] developed by the Software Agents Lab at Carnegie Mellon University's Robotics Institute include the planning, scheduling and executing modules, which is quite suitable for our system software development. Therefore, we use the RETSINA agent architecture to develop our software architecture for the Real-time Financial Data Mining system.

## 4.2 Descriptions of Middle Layer Server Side Software Components

There are four components and two data storages in the middle layer server application as shown in figure 4.1. In our system, different kinds of servers have the same software components, but with different functions libraries that determine the type of the server. The two data storages are Data Cache and Functions Library. The four software components are Planner Module, Scheduler Module, Execution Module and Communicator [10]. We will give detailed descriptions of them in the following sub-sections.

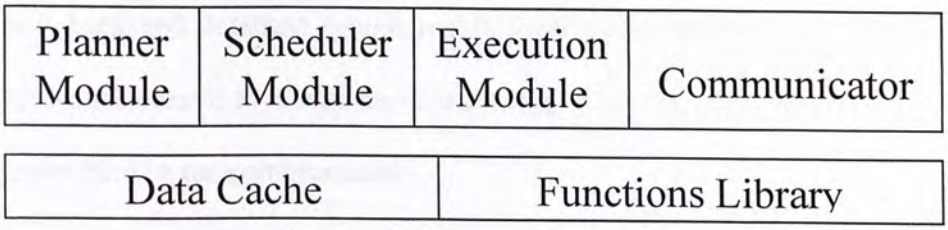


Figure 4.1: Server side software Architecture

### 4.2.1 Data Cache

Data Cache is the local data storage for the server application. It can store the task requests from client's applications, active clients list, real-time stock data, the search results from the database or the analysis results of the stock market.

### 4.2.2 Functions Library

Functions Library stores the specific primitive functions code for the server that determines the type and abilities of the server. For example, the on-line data analysis server gets the mathematical computation library for performing data analysis function; the user management server gets the security library for setting up secret connection between server and clients; the data management server gets the database manipulation library for managing the back-end database. And the Functions Library contains a task reduction schema [11] [12] presenting a way of carrying out a task by specifying a set of sub-tasks/actions and describing the information-flow [11] [12] relationships between them.

### 4.2.3 Communicator

The communicator is responsible for managing network communication for middle layer servers. The communicator will poll for client services requests

and back-end database search result, send query for back-end database, and return task result to clients asynchronously. The task allocation protocol will be embedded in the communicator.

### 4.2.4 Planner Module

The planner module queues the incoming service requests (task) from client's applications and plans the way to carry them out. The planning process is based on a hierarchical task network (HTN) planning formalism [14] .It takes in the server's current tasks and a library of task reduction schemas [11] [12]. A task reduction schema presents a way of carrying out a task by specifying a set of sub-tasks/actions and describing the information-flow relationships between them. That is, the reduction may specify that the result of one sub-task (e.g. query return) be provided as an input to another sub-task (e.g. sending a result). Actions may require that certain information be provided before they can be executed, and may also produce information upon execution.

### 4.2.5 Scheduler module

The scheduling process takes as input the server's current tasks structures, and the set of all primitive actions, and decides which primitive action to be executed next. The scheduler attempts to maximize some predefined utility function [13] defined on the set of task structures. For all servers, we use a very simple notion of utility—every action needs to be executed in order to achieve a task, and every task has an equal utility value. And we use first come first serve scheduling in our system.

### 4.2.6 Execution Module

The execution module takes in the server's next intended action (task) and prepares, monitors, and completes its execution. The execution module prepares an action for execution by setting up parameters (including the results of previous actions (tasks) etc.) for the action [13]. Then it performs the action execution.

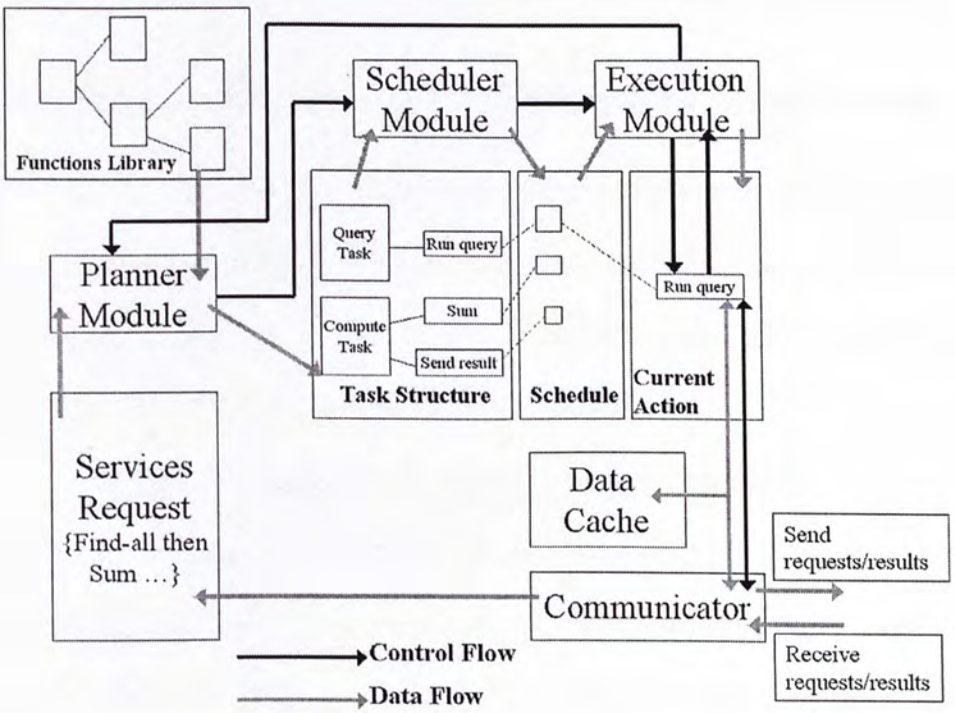


Figure 4.2: Control and data flow inside server

## 4.3 Overview the Execution of Service Request inside Server

Service request (*SR*) come from clients are received by the communicator of server, and are stored in the data cache. The planner module of the server inputs the *SR* and the function libraries, and then uses the task reduction schema to

find a way for carrying out the *SR* by transforming the *SR* into hierarchical task network. Then the scheduler module inputs the task structure and schedules the execution of tasks. The execution module takes in the scheduled task and performs the execution and monitoring. In the task execution, server may first send query messages to back-end database for query data, then perform some computation, and finally return *SR*'s results to client through the communicator. Figure 4.2 shows the control and data flow of task execution inside server.

## 4.4 Descriptions of Client layer Software Components

There are two software architectures for client's application. Figure 4.3 shows the software architecture of client's application that can run in all graphical computing platforms, and figure 4.4 shows the client software architecture specific for operating in personal computer platform.

The first architecture has two data storages, Data Cache and Functions Library. It also has two components, the Graphical User Interface and the Communicator. They provide basic functions such as user input, data presentation and network connection for clients. That is suitable for all graphical computing platforms.

The second architecture provides on-line data analysis functions by integrating the planner module, scheduler module, execution module and data analysis function libraries to the client's application. So, the second architecture is suitable for personal computer platform with considerable computing power. With the data analysis library, the system can enable distributed data analysis: when a client's application gets a data analysis goal, it can share the tasks of

archiving the data analysis goal to other client's application and server (that has the data analysis ability). We can execute tasks in parallel, so the time to complete the goal can be reduced. To enable distributed computing, we also need to integrate a task allocation protocol to the communicator in the client's application.

Detailed description of the Graphical User Interface is presented in next section, and the description of remaining components can be found in previous sections.

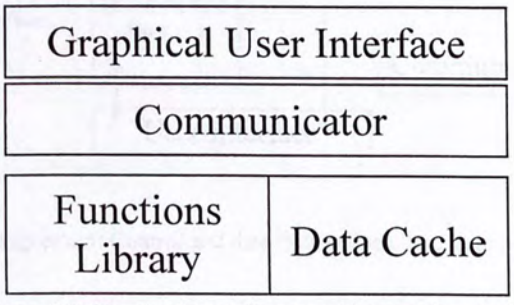


Figure 4.3: Simple client software architecture for all platforms

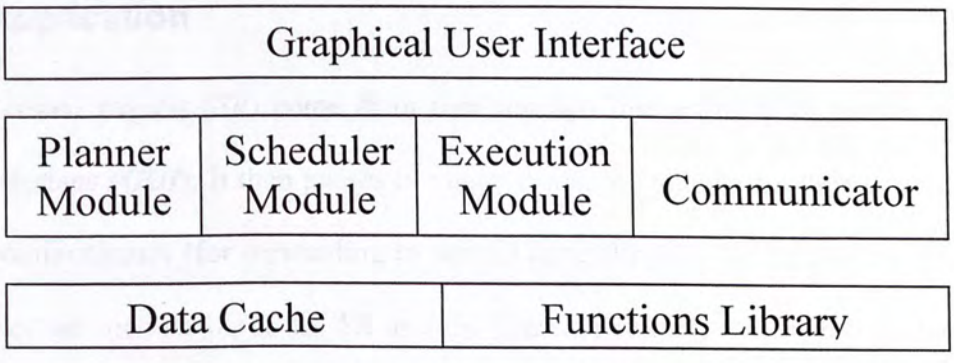


Figure 4.4: Advanced client software architecture specific for personal computer platform

### 4.4.1 Graphical User Interface

The graphical user interface is responsible for getting user input, data presentation such as chart plotting, and forwarding user's service requests to communicator or local computing module.

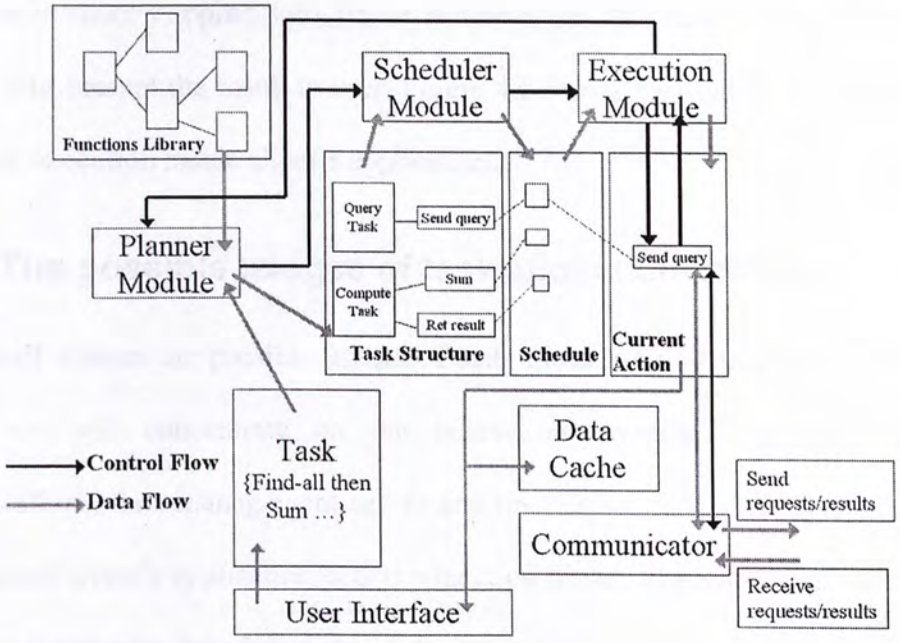


Figure 4.5: Control and data flow inside client's application

## 4.5 Overview of Task Execution in Advanced Client's Application

Service request (*SR*) come from user through interacting with graphical user interface (***GUI***). It then passes to queue inside the planner module or passes to communicator (for forwarding to server) depending on the type of the *SR*. The planner module input the *SR* and the function library, and then uses the task reduction schema to find a way for carrying out the *SR* by transforming *SR* into hierarchical task network. The scheduler module then inputs the task structure and schedules the execution of tasks. The execution module takes in the scheduled task and performs the execution and monitoring. In the task execution, client may first send query messages to middle layer server for query

data, then perform required computation (or forward the task to server and other advanced client's application), finally it may return *SR*'s results to ***GUI***, and the ***GUI*** will present the result to user. Figure 4.5 shows the control and data flow of task execution inside client's application.

## 4.6 The possible usages of task allocation protocol

We will discuss the possible usages of task allocation protocol in our system, and we will concentrate on the interaction between advanced client's applications, data management server and on-line data analysis server. As the advanced client's application gets the function library to perform data analysis, and as it can plan, schedule and execute some tasks by itself, it is possible for it to share the task to other client's application that has the same abilities. Therefore, distributed computing is possible.

The main purpose of distributed computing for our system is to make a path for client's application in a busy period to allocate tasks to other client's applications (in low CPU usage), and to facilitate the use of client computers (in low CPU usage) as a computing power of the system. Therefore, while upgrading system hardware can reduce the tasks completion time, developing an efficient task allocation method can help too. Thus, our distributed computing concept attempts to gather client's computing power for improving system capacity, performance and stability. And the task allocation method will determine how much the system can improve.

### 4.6.1 Chart Drawing

Stock Chart is the basic and widely used method to present the past and current stock prices. User can know about the past and current stock price quickly by looking at the chart. The chart presents not only stock prices but also pattern of the prices. The stock chart can show the technical analysis results as well. When user goes off-line, the on-going stock price will be missing in client's application. When user goes on-line again, if he wants to view stock chart, the client's application needs to request the data management server to retrieve the missing data so that to have a continuous chart.

If every advanced client's application caching real-time stock data for a fix period, the data missing within this period can also be retrieved from on-line client's applications. Hence, when client's application wants to retrieve the missing data, it can use the task allocation protocol to help it in allocating the data retrieval task to server or other clients according to their current loading or other information. Thus a simple data caching can help in share the task loading of the data management server.

### 4.6.2 Compute user-defined technical analysis indicator

To compute user-defined technical analysis indicator, the amount of data for computing is one of the most important factors in determining the time for the computation. With task allocation protocol, it is possible to compute user-defined technical analysis indicator on a large data set by dividing the large data set into a number of smaller data sets, and then distribute one data set to one

client's application, enable parallel computing, in order to reduce the time for the indicator computation.

### 4.6.3 Unbalance loading

Different people may be interested in different area of the stock markets. For example, people may be interested in the company that he has invested or he may be interested in a specific commercial sector. So, sometime when the price of a specific company or sector in stock market reaches a critical point, it may trigger the group of clients who are interested with it to perform data mining actions in order to find out the future investment direction. In this situation, a group of clients busy with the data mining actions, while the remaining clients may be waiting on their interested sectors of the markets to change and doing nothing. Hence, clients in busy period sharing the tasks loading to other relatively low loading clients, can help to reduce the average task completion time, and utilize the system resources.

### 4.6.4 Large number of small data mining V.S. small number of large data mining

When we got large number of small data mining tasks at the same time, allocate the tasks to a number of client's application, executing them at the same time, may reduce the time to complete all the tasks when compare to execute those tasks sequentially in a single server. When we got a small number of large and undividable data mining tasks, allocate the tasks to a single server, we can benefit from using the advance computing power provide by the server. So, a

dynamic task allocation protocol would be needed to perform a flexible task allocation by client's application.

## 4.7 Summary

From the above discussion, we know that with the advanced client software architecture plus a dynamic task allocation protocol, there is a possible way where we can benefit from enabling parallel, distributed computing and tasks sharing amount client's application and middle layer servers. But there is also an important assumption to make the benefit possible, which is client's applications need to be in low tasks loading on time average. If client's applications often in busy status, then they can provide a relatively less time for helping other clients applications to perform tasks execution or data mining actions. A dynamic task allocation protocol is preferred instead of a static one, because client's applications may go on-line and off-line in an unpredictable way, so static task allocation is not possible.

# Chapter 5

# The Contract Net Protocol for Task Allocation

## 5.1 Introduction

We have introduced the origin Contract Net Protocol (CNP) [1] [2] in section one of chapter 2. In this chapter we will introduce the Foundation For Intelligent Physical Agent (FIPA) Contract Net Interaction Protocol (IP) [3] that is a minor modification version of the original contract net protocol. Our Modified Contract Net Protocol (MCNP) is developed base on the FIPA contract net IP. In this chapter, we will have a detail look into the operating mechanisms of the FIPA contract net IP, and state its strengths and weakness for task allocation. As we know, the origin CNP and the FIPA contract net IP are designed for artificial intelligence agents, so we will also explore the necessary modifications on the FIPA contract net IP to make it suitable for our system (a client/server architecture software system).

## 5.2 The FIPA Contract Net Interaction Protocol

### 5.2.1 Introduction to the FIPA Contract Net Interaction Protocol

The FIPA Contract Net Interaction Protocol (IP) is a minor modification of the original contract net IP pattern in that it adds proposal rejection and confirmation communicative acts [3]. In the contract net IP, we view agents

who wishes to have some task performed by other agents as managers, and by allocating the task to other agents, the manager wishes to optimize a function that characterizes the task. This characteristic is commonly expressed as the price, in some domain specific way, but could also be soonest time to completion, fair distribution of tasks, etc [3]. Figure 5.1 show the UML interaction diagram for the FIPA contract net interaction protocol.

By sending a call for proposals (CFP) message, the manager solicits other agents to perform task. The call for proposals message specifies the task, the conditions that the manager is placing upon the execution of the task and the specification on the returned bid. The potential contractors should able to generate proposal and perform the task. After receiving CFP message, the potential contractor follows the bid specification to fill in the required information for biding the task, which may be the price, time when the task will be done, etc. Alternatively, the contractor may refuse to propose. Once the deadline passes, the manager evaluates any received bids and selects agents to perform the task; one, several or no agents may be chosen. The manager will send award messages to the potential contractors, informing them with accept-proposal or reject-proposal. Once the contractors have sent the proposals, they need to commit the execution of the task if the manager accepts the bid. The contractor will send the task execution result to the manager immediately after the task completion.

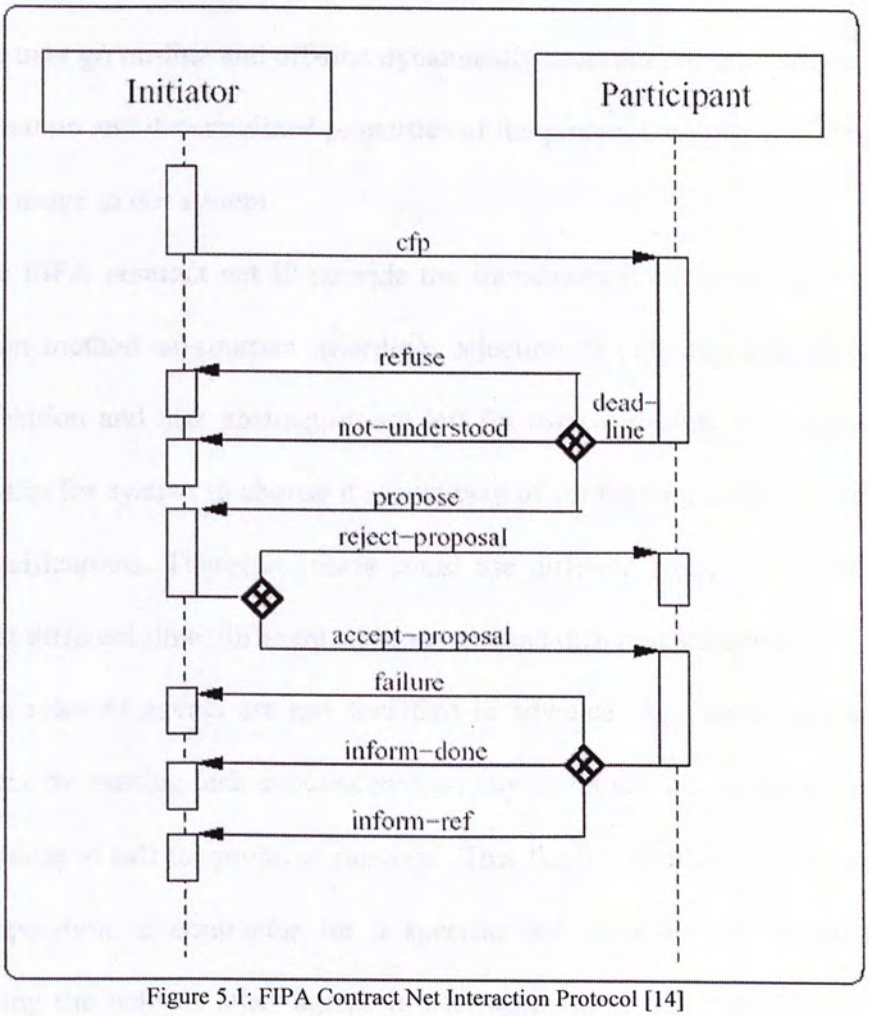

Figure 5.1: FIPA Contract Net Interaction Protocol [14]

### 5.2.2 Strengths of the FIPA Contract Net Interaction Protocol for our system

1) The protocol is simple; it's only need three messages to allocate a task, they are the call for proposals message, biding message, and awarding message.

2) The protocol is dynamic and decentralized, as manager make his own choice in allocating task according to the information provided in the biding message. Contractors can dynamically join or leave the system to provide services for manager before he has make any commitment to any task. In our system, clients

may act as contractors to provide on-line data analysis services for clients, and clients may go on-line and off-line dynamically according to their habits; hence the dynamic and decentralized properties of the protocol are required properties for the usage in our system.

3) The FIPA contract net IP provide the framework for contracting while the decision method of contract awarding, selection of potential contractors, bid specification and task abstraction are left for user to design, this give a great flexibility for system to choose it's own way of contracting in order to achieve its specifications. Therefore, users could use different strategies in allocating tasks at different time, different environment and different situation.

4) The roles of agents are not specified in advance. Any agent can act as a manager by making task announcements; any agent can act as a contractor by responding to call for proposal message. This flexibility allows for further task decomposition: a contractor for a specific task may act as a manager by soliciting the help of other agents in solving parts of that task. The resulting manager-contractor links form a control hierarchy for task sharing and result synthesis.

5) The FIPA contract net IP requires the manager to know when it has received all replies. In the case that a contractor fails to reply with bid indicating either *propose* or *refuse to propose*, the manager may potentially be left waiting indefinitely. To guard against this, the *call for proposal* includes a deadline [3] by which replies should be received by the manager. Bids received after the

deadline are automatically rejected with the given reason that the proposal was late.

7) While waiting for a task to be completed, a manager may take on the role of a contractor for another contract, rather than remaining idle.

### 5.2.3 Weakness of the FIPA Contractor Net Interaction Protocol for our system

1) The specification of the protocol are based on agent language and platform, it's difficult to use in our client/server architecture system.

2) The start of bids evaluation in manager side is fixed to be after the biding deadline. This is less flexible than if the manager can start the bids evaluation before the biding deadline, because some task allocation need not to receive all bids before manager can make the task allocation decision, and manager can start evaluate the received bids while waiting for biding deadline, thus the bids evaluation may complete earlier.

3) The contractor biding message doesn't contain the field to indicate the valid time of the biding message, which force the contractor to allocate resource for the bid after the biding message has been sent out no matter when the manager make the award reply. It's not flexible for the contractor who may concern about the resources that has been allocated for performing the task indicated in the call for proposals message. The contractor may potentially allocate the resources forever if the awarding message has lost in unreliable network transfer, and the biding message will store in the protocol message queue

forever. It is an important stability concern for our client's applications, as they will act as contractors in our system.

4) The contractor is under no obligation to send confirmation message when it receive the awarding message, so the manger will never know his awarding message hasn't been received by the contractor because it's lost in the unreliable network transfer or the contractor application has crashed. Thus the manager may not get the task completion result forever. For our system, the task completion is more important than whether the contractor can get the right to perform the task, so without the award confirmation message will create a great problem to our system.

6) This protocol was designed for distributing one task among a number of contractors, its may not perform well for distributing a number of task among a number of contractors.

## 5.3 The Modified Contract Net Protocol

1) We introduce the field for specifying starting time of bids evaluation in the call for proposals (CFP) message, which allow manager to set the time for start of bids evaluation earlier than the biding deadline. Figure 5.2 shows the only possible time sequence diagram of the FIPA contract net IP where the start of bids evaluation time is the same as the biding deadline.

By setting the start of bids evaluation equal to the biding deadline, our MCNP include the same time sequence as the FIPA contractor net IP.

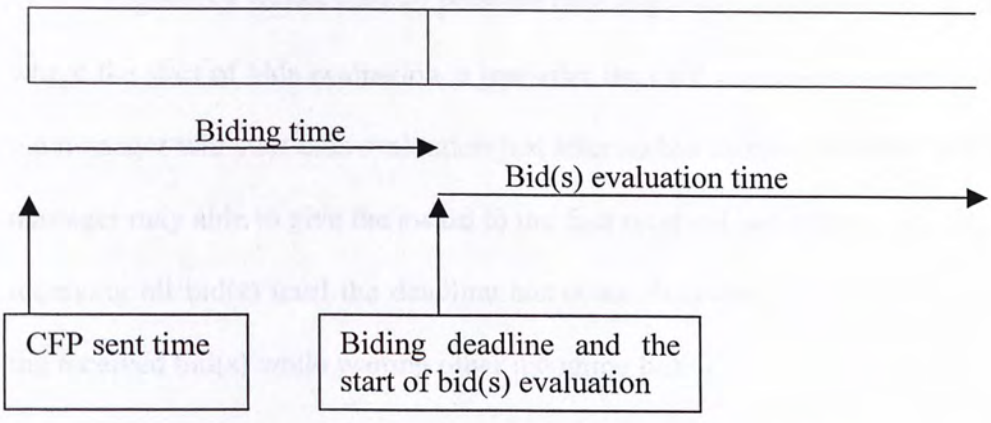


Figure 5.2: The time sequence diagram of the FIPA contract net IP

Figure 5.3 shows a possible time sequence diagram for the MCNP, in which the start of bids evaluation is in-between the CFP sent time and the biding deadline. So the manager can start evaluating the received bid(s) before biding deadline, and hence before the biding deadline has come, he may has already completed evaluating part of the received bid(s) and continuous the remaining part after the biding deadline, thus the protocol delay may probably reduced. And the manager can also award a bid before the biding deadline has come if he received and evaluated a bid that satisfied his requirement(s).

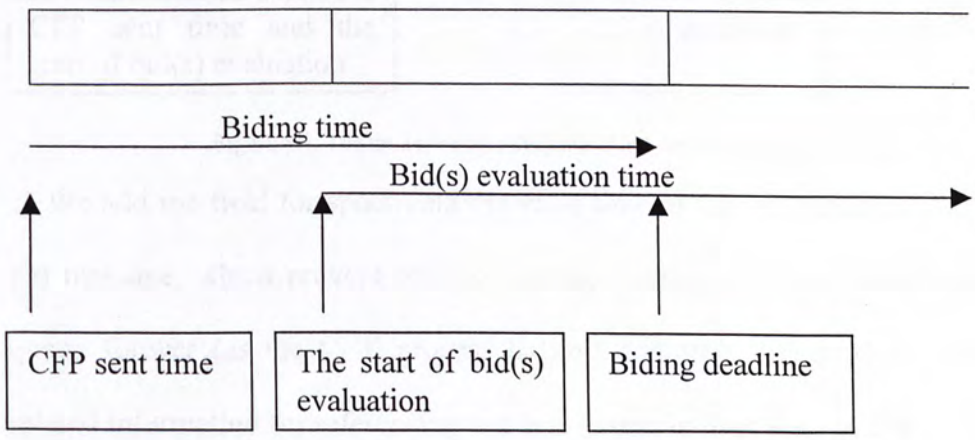


Figure 5.3: The time sequence diagram of the modified CNP

Figure 5.4 shows another possible time sequence diagram of the MCNP, where the start of bids evaluation is just after the CFP message has sent. Hence the manager will start bids evaluation just after he has received the first bid, and manager may able to give the award to the first received bid without waiting for receiving all bid(s) until the deadline has come. It is also possible to evaluate the received bid(s) while waiting other incoming bid(s).

The ability to set the starting time of bids evaluation is very flexible for our client's application, where it can choose different approach to start the bids evaluating process according to its requirement and the current system environment.

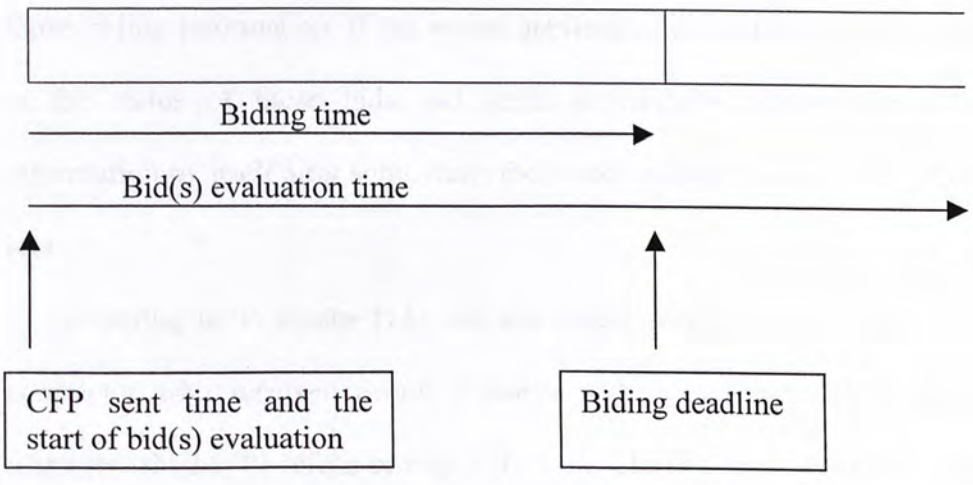


Figure 5.4: The time sequence diagram of the modified CNP

2) We add the field for specifying the valid time of the bid message inside the bid message, which prevent the bid message storing in the protocol message queue forever (as the CNP process haven't complete, we need to store the related information for referencing and processing in later state of the CNP) and prevent the contractor allocate resources for the biding content forever if the manager awarding message has lost in the unreliable network transfer. This

addition is very important for our middle layer servers, as the middle layer servers able to do any tasks offered by client's application, so these servers are potential contractors for all client's applications in any time, hence these servers will receive a large amount of CFP messages and making a lots of bids (as its only responsibility is to serve client's applications), so even only 0.001% of the award messages will loss in network transfer, but the number of bids will store in server side due to the lost of award messages will increase rapidly as time go by. The performance of the contract net protocol will degraded rapidly, the memory used for storing those biding information will increase rapidly, and the whole server system may crash easily due to using up the memory in storing those biding information. If the server application did constantly keep looking at the status of those bids and make decision to remove those biding information by itself after some time, the work loading of server will increase a lots.

3) According to T. Knabe [15], we add award confirmation message for the contractor who received award message with accept-proposal to reply the manager whether he refuse or accept the task. The award confirmation message has two usages: firstly, the manager who received the confirmation message can sure the contractor has received the award message. If the manager didn't receive the confirmation message after some time, he can decide to award the task to other contractor or restart the contracting process, hence the manager won't get any problem due the lost of award message in the unreliable network transfer. Secondly, for client's application using the MCNP, user can restricting

a maximum number of task offer acceptance at the same time, once the number of external tasks inside the local tasks queue has reach the maximum number allowed by the restriction, the task award will be rejected. With the above restriction, client's application will not suffer from accepting a large number of tasks in short time due to the abnormal behavior of other client's applications. By adding the confirmation state, we put the task commitment [16] at a later time, which is in the confirmation state not in the biding state when compare with the FIPA contract net IP. Putting the task commitment at a later time has advantage for contractor who decided to restrict the number of task acceptance. It is because if contractor need to commit the task acceptance in the biding stage, then he cannot bids more than the maximum number of task that he can accept at a time because all his bids may be accepted by managers and he need to commit all the bids even this situation happen in very low probability (as the number of competing contractors may be very large). So, once the contractor bids have reached the maximum number of task acceptance, contractor needs to refuse other incoming CFP, and at the final, contractor may received task award less than the maximum number allowed, thus the resources given by contractor may not be able to fully utilize in most of the time. Hence, by delaying the task commitment in the confirmation state, the contractor can make bid to all CFP, and the confirmation state will restrict the number of tasks accepted. So, the resources given by contractor will have a better utilization. We assume our client's applications are cooperative contractors, they will not reject task award

if they haven't reach their resources limitation. Figure 5.5 show the UML interaction diagram for the MCNP with the award confirmation state.

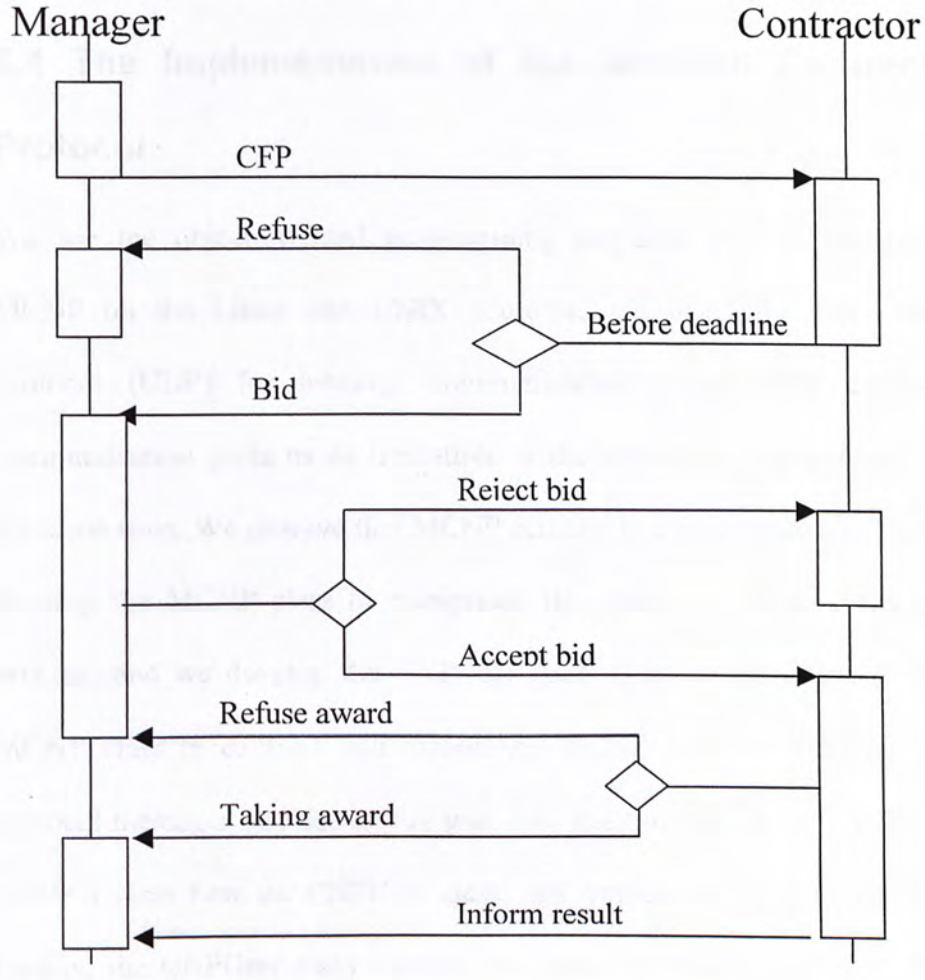


Figure 5.5: MCNP UML interaction diagram

4) We add the award confirmation deadline in the award message, so that the MCNP can inform the manager to decide awarding the task to other contractor or restart the contracting process if the manager didn't get any confirmation message before the confirmation deadline. This addition set a time out time for manager to waiting the confirmation message. By setting a relative long confirmation deadline can reduce the number of unnecessary restart of contracting process due to network transfer delay. How long the confirmation

deadline is needed will depend on the real-time network environment, and it will leave for MCNP user to determine.

## 5.4 The Implementation of the Modified Contract Net Protocol

We use the object-oriented programming language C++ to implement the MCNP on the Linux and UNIX platform, and using the User Datagram Protocol (UDP) for message communication. Using UDP for message communication gives us no limitation on the number of connections exists at the same time. We observe that MCNP actually is a finite state machine, so we develop the MCNP class to manipulate the states transition of the protocol process, and we develop the CNPUser class to act as an interface class for MCNP class to callback and inform the MCNP user for handling different protocol messages and status. For user who want to use the MCNP, he need to create a class base on CNPUser class, and implement the callback functions base on the CNPUser class interface for handling different protocol messages and status in his own way (for example, he will need to implement the function to make his way on handling the call for proposals message if he want to be a contractor). Figure 5.6 show how the user can connect MCNP class through the CNPUser class interfacing.

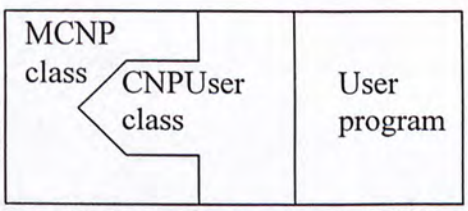


Figure 5.6: The connection diagram for user program, CNPUser class and MCNP class.

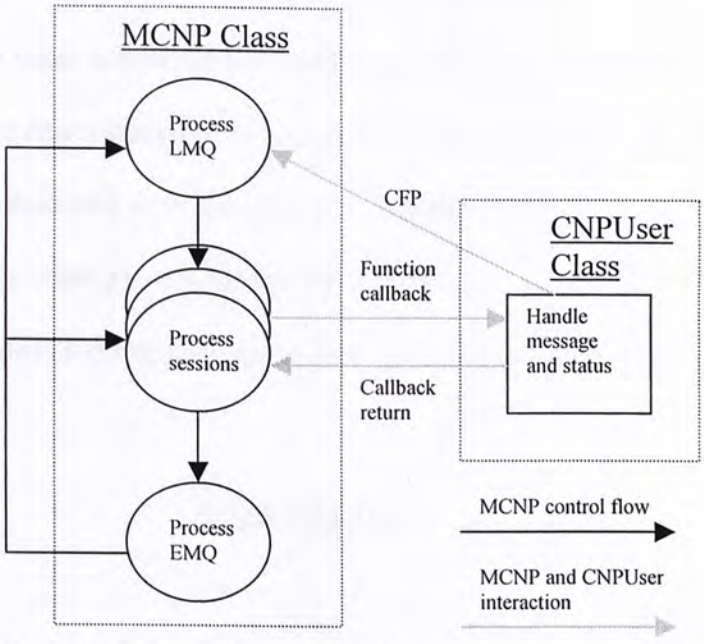


Figure 5.7: The control flow and interactions of the MCNP class and CNPUser class.

Figure 5.7 show the control flow and interactions of the MCNP class and CNPUser class. First, the CNPUser class passes the call for proposals (CFP) message to the MCNP class local message queue (LMQ), which informs the MCNP class to process the LMQ. In processing the message in the LMQ, the MCNP class creates a session for the CFP message, then it go to process the sessions in the current MCNP class, if the session need the CNPUser class to handle some received message or change of status, the MCNP will callback the CNPUser class functions, and pass the necessary information for the CNPUser class to process. After CNPUser class has handled the callback, it returns instructions to the MCNP class for further processing on the session. Then it goes to process one message from the external message queue (EMQ) that stores the message received from network. By processing the EMQ, received messages may be attach to it related sessions, or it will initiate the MCNP class

to process it related session. After process one message in the EMQ, the MCNP class will go back to process the LMQ, thus the whole control flow go back to the beginning and restarted. It is important to note that in each processing cycle, the MCNP class will only process one message from LMQ and one message from EMQ in order prevent blocking the process on a single message queue for very long time if there are a large number of messages pump into the queue in a short time.

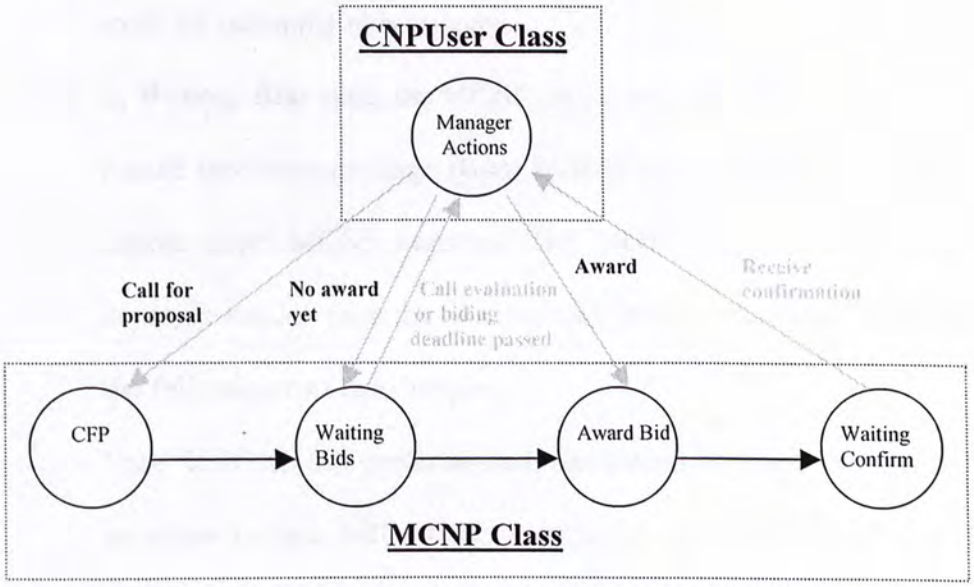


Figure 5.8: State transition diagram of MCNP in manager side actions.

From figure 5.8 we could know the state transition of the MCNP class and the CNPUser class for manager making task allocation. Manager uses the CNPUser class interface to control the contract net process operated by the MCNP class. Now, let us look into the state transition for manager making a task allocation.

1) The manger first activate the CNPUser object to ***Manager Actions*** state and make the call for proposal (CFP) action, then the CFP message will pass to the MCNP object and store in the MCNP object local message

queue (LMQ), then the MCNP object will informed to have message in the LMQ. So, it will read in the message from the LMQ, after classified the message as CFP message, the MCNP object will create a new *manager session* and the new *manager session* will go to the ***CFP*** state, it' will store a copy of the CFP message for future referencing and then sent it to the potential contractors as indicated in the CFP message. Then the *manager session* will go to the (2) ***Waiting Bids*** state to wait and store the incoming bids message.

2) In ***Waiting Bids*** state, the MCNP object will idle this session, and go to handle incoming message (from local program or external network) or handle other MCNP sessions. The MCNP object will activate this *manager session* once the time for start of bids evaluation has come, and the following two case happen:

   Case 1: if no bid received and the biding deadline has passed, the *manager session* will inform manager about the situation through the CNPUser object, and CNPUser object will go to ***Manager Actions*** state to execute manager decision to restart the task allocation process or cancel the task allocation.

   Case 2: if no bid received but the biding deadline hasn't pass, the *manager session* will stay in the ***Waiting Bids*** state and it will idle by the MCNP object.

a. Once a bid received before deadline, the MCNP object will pass the bid to the CNPUser object, and the CNPUser object will go to ***Manager Action*** state.

   i) Firstly, if the manager needs time to evaluate the bid, it will return ***No Award Yet*** to the *manager session*, thus release the *manager session* to (2) ***Waiting Bids*** state

   ii) Secondly, if the manager complete the bid evaluation, but will not accept the bid, it will return ***No Award Yet*** to the *manager session*, thus release the *manager session* to (2) ***Waiting Bids*** state. If the manager accepts the bid, it will pass an award message to the *manager session*, and the process will go to (3) ***Award Bid*** state.

b. When biding deadline has passed, the MCNP object will go back to the *manager session* and inform the manager, then the CNPUser object will go to ***Manager Action*** state.

   i) If the bid evaluation process has completed with no award, the manager will need decide restart the task allocation process (1) or cancel the task allocation.

   ii) If the bid evaluation process has completed with award, the manager will pass the award message to the *manager session*, and the *manager session* will go to the (3) Award Bid state.

iii) If the bid evaluation process hasn't completed, the manager will return ***No Award Yet*** to the *manager session*, and the *manager session* will go to the (3) Award Bid state to wait for award.

3) In ***Award Bid*** state, if the *manager session* didn't get the award message, it will stay in ***Award Bid*** state, wait for the manager to give the award or terminate the session. At this time the MCNP object will idle this session to handle other process, once the manager pass the award to the LMQ, the MCNP object will be informed to activate the *manager session*, it will sent the message to the awarded contractor, and then the *manager session* will go to (4) ***Waiting Confirm*** state.

4) In Waiting Confirm state, the *manager session* will wait for the contractor confirmation message and hence it will idle by the MCNP object again. If the confirmation message didn't receive before confirmation deadline, the MCNP object will activate the *manager session* to inform the manager, then the CNPUser object will go to ***Manager Action*** state to execute the manager decision on whether give the award to other contractor (manager session go back to (3) ***Award Bid*** state), restart or terminate the contracting process. If the confirmation message is received before confirmation deadline, the MCNP object will activate the *manager session* to inform the manager, and the whole task allocating process in manager side has completed.

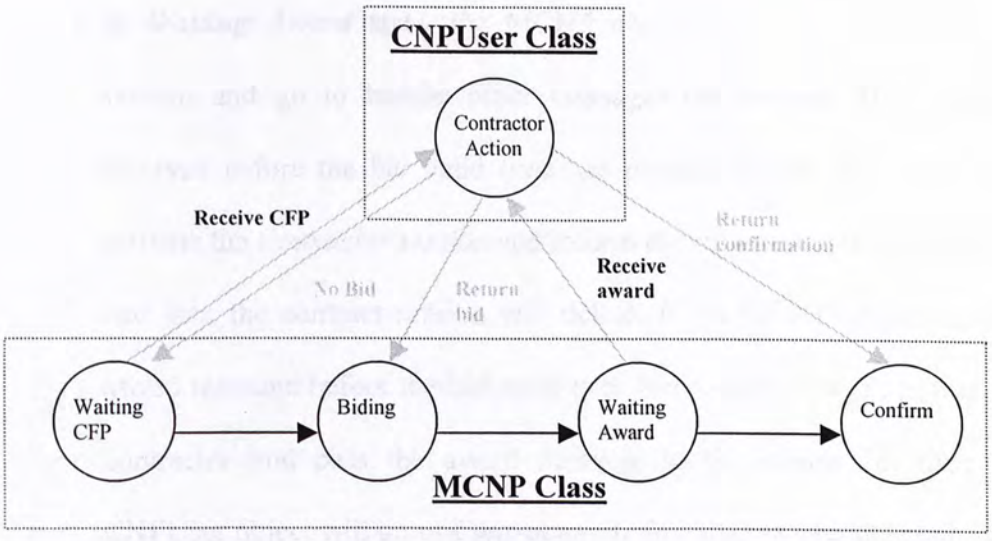


Figure 5.9: State transition diagram of MCNP in contractor side actions

From figure 5.9 we could know the state transition of the MCNP class and the CNPUser class for contractor waiting task allocation. Contractor uses the CNPUser class interface to control the contract net process operated by the MCNP class. Now, let us look into the state transition for contractor waiting task allocation.

1) The MCNP object after receiving CFP for network, it will create a *contractor session* and the *contractor session* will go to ***Waiting CFP*** state, then it pass the CFP to contractor through CNPUser object, and the CNPUser object will go to the ***Contractor Action*** state. If contractor doesn't want to bid, the CNPUser object will return ***No Bid*** to the MCNP object, and the MCNP object will delete the *contractor session*. If contractor bid, the CNPUser object will pass the proposal to the MCNP object, the *contractor session* will go to Biding state.
2) In ***Biding state***, the MCNP object will sends the bid message to the manager and then the *contractor session* will go to ***Waiting Award*** state.

3) In ***Waiting Award*** state, the MCNP object will idle the *contractor session* and go to handle other messages or sessions. If no award received before the bid valid time has passed, the MCNP object will activate the *contractor session* and inform the situation to the contractor, and then the contract session will delete. If the MCNP object receive award message before the bid valid time has passed, it will activate the contractor and pass the award message to the contractor, then the CNPUser object will go to ***Contractor Action*** state to execute contractor decision on whether confirmation the award (if the award indicate the bid is accepted) or terminate the *contractor session* (if the award indicate the bid is rejected). If contractor confirm the award, the CNPUser object will return the confirmation message, otherwise it will return the instruction of terminating the *contractor session* to the MCNP object.

4) If the MCNP object gets the termination instruction, the *contractor session* will be deleted. If it gets the confirmation message, the *contractor session* will go to ***Confirm*** state, and it will send the confirmation message to the manager. After the confirmation message has sent out, the *contractor session* will be deleted and the whole task allocation process in the contractor side has completed.

## 5.5 Summary

In the implementation, the MCNP class is run on a different thread from the CNPUser class and the user program. If user program will not block on the

MCNP callback, the bid evaluation process can keep ongoing while the MCNP keep receiving incoming messages and instructions in the same time. Also, with the above design and implementation of the MCNP control flow, it's able to handle multiple contracting sessions and send out multiple CFP without waiting the contracting process to complete before starting a new one. By using UDP for network communication, the number of contracting process will not restricted by the number of connections can be make at the same time, the communication channel is centralized for a much more easy control. And the award confirmation message ensures the contractor received the award.

# Chapter 6

# A Client as Server Model using MCNP for Task Allocation

## 6.1 Introduction

We will design a Client AS Server (CASS) system model enabled by the Modified Contract Net Protocol (MCNP) for the on-line data analysis service of the Real-time Financial Data Mining system. We compare the system capacity increased by the new system model with a single-server exponential queueing system (M/M/1) through performance analysis and simulation. Thus showing a task allocation method combine with the task allocation protocol (MCNP) can enable the potential benefit of employing distributed computing.

## 6.2 The CASS System Model

In our CASS system model, we assume:

- The system only contain clients using personal computer with advance client's application, therefore every client's application can handle the task if it got the corresponding library to perform it.
- All complex tasks have reduced into primitive tasks before starting the task allocation process if not specify externally, that mean tasks go into allocation process could map directly into each functions library.

In the CASS system model, there is an on-line data analysis server in the middle layer to provide services for all clients. We group the commonly used functions of the on-line data analysis server into 5 libraries, which are price/volume analysis library, fundamental and risk analysis library[1], pattern matching library, technical indicators analysis library and financial news analysis library. Then we give one library to one client's application, and five client's applications with distinct library can group together to act like an on-line data analysis server to provide those commonly used functions to each other inside the group as show in figure 6.1.

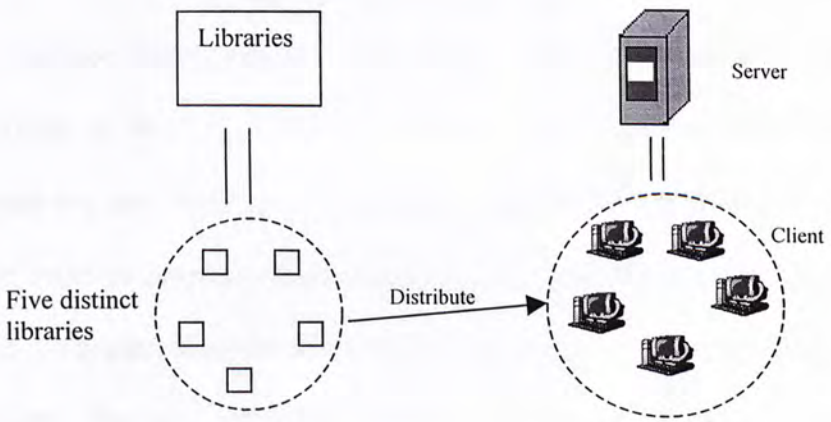


Figure 6.1: Five client's applications with distinct library can group together to act like a server

Hence, in our CASS system model, there is an on-line data analysis server in the middle layer serving all clients and at the same time every five client's applications group together in client layer, acting like the server to serve each other inside the group. Figure 6.2 shows the CASS system model,

[1] Fundamental analysis includes the analysis of economic environment, the stock market trend, business sectors, company financial information, etc.

where client's application can send its service request to the on-line data analysis server or other clients in the same group to get the requested services.

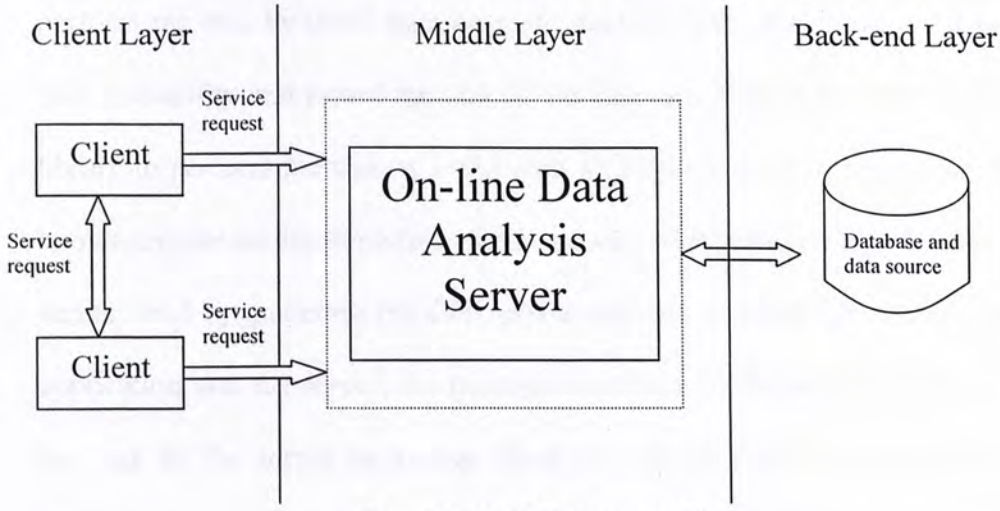


Figure 6.2: The CASS system model

Because clients can get services from client's applications and server, we decided to develop a task allocation method combine with MCNP for balancing the task loading among client's applications and server, such that average services response time (queueing time plus servicing time) of server and client's applications are the same. Which means all tasks will treat fairly in the system. The task allocation method will use the MCNP to gather tasks loading information in the CASS system in real-time environment.

Here, we describe the general idea of the task allocation process. For a new primitive task generated by a client's application (manager), the manager will first classify which functions library is needed to perform the task, if the manager has that functions library, it will use the MCNP to send a call for proposals (CFP) message to the on-line data analysis server, and the on-line data analysis server will return it services rate and the current CFP arrival rate (that is the maximum possible task arrival rate) received by the server, then the

manager will base on the these information with its current task loading and services rate to decide the probability of allocating the task to the server or perform the task by itself, then generate the task allocation result according to that probability and award the task. If the manager doesn't have the functions library to perform the task, it's will send a CFP to a client's application that it known has the ability to perform the task and a CFP to the on-line data analysis server. And by gathering the CFP arrival rate and services rate of that client's application and the server, the manager calculate the probability of allocating the task to the server or to that client's application, then generate the task allocation result according to that probability, and award the task by using the MCNP.

## 6.3 The analytical model of the CASS system

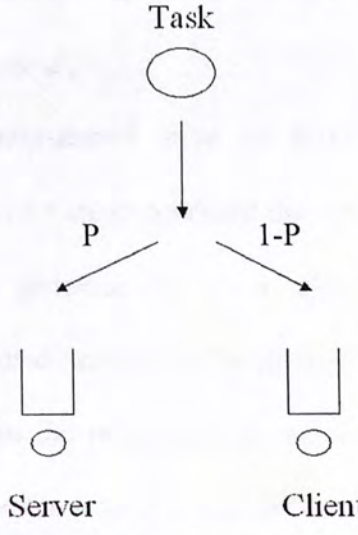


Figure 6.3: Task allocation among client's application and server

Figure 6.3 shows the generic view of task allocation among client's application and server in the CASS system model. A task generated in the CASS system will allocate between a client's application and the server. The task will allocate

to the server with probability ***P*** and allocate to a client's application with probability ***1-P***, where this probability is determine by the services rate and the CFP arrival rate of the server and the client's application in order to archive load balancing between client's application and server. The server in figure 6.3 is the on-line data analysis server, while the client in figure 6.3 could be the client's application that generate the above task or other client's applications in the network that has the ability to perform the task execution.

In order to find out the probability ***P*** for archiving load balancing with the criteria of getting the same average services response time for client's applications and server in the CASS system, we need to get the tasks arrival rate that are allocating between server and a client's application. In our CASS system, we assume:

1. Client's applications are identical and independent in generating tasks arrival,
2. The inter-arrival time of tasks generate by each client's application are exponential distributed with rate equal $\lambda$,
3. Tasks generate by each client's application are equally distributed on the five functions libraries, i.e. 1/5 of tasks arrival will use the price/volume analysis library, 1/5 of tasks arrival will use the pattern match library, and so on.

Thus, for each client's application, 1/5 of the tasks arrival can allocate among itself and the server (as show in figure 6.4) and the remaining 4/5 of the

tasks arrival can allocated among other client's applications (inside its group) and the server (as show in figure 6.5).

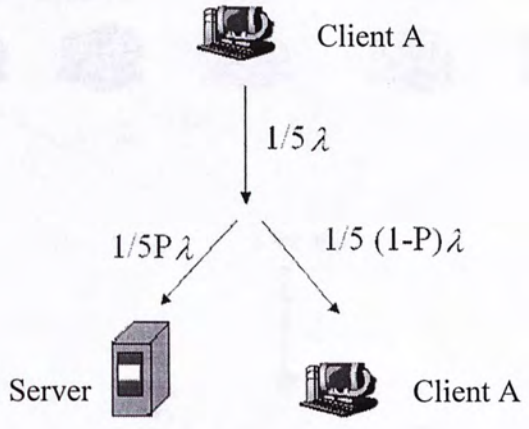


Figure 6.4: Each client's application generates tasks arrival rate of λ, with 1/5 tasks arrival can allocate between itself and the server.

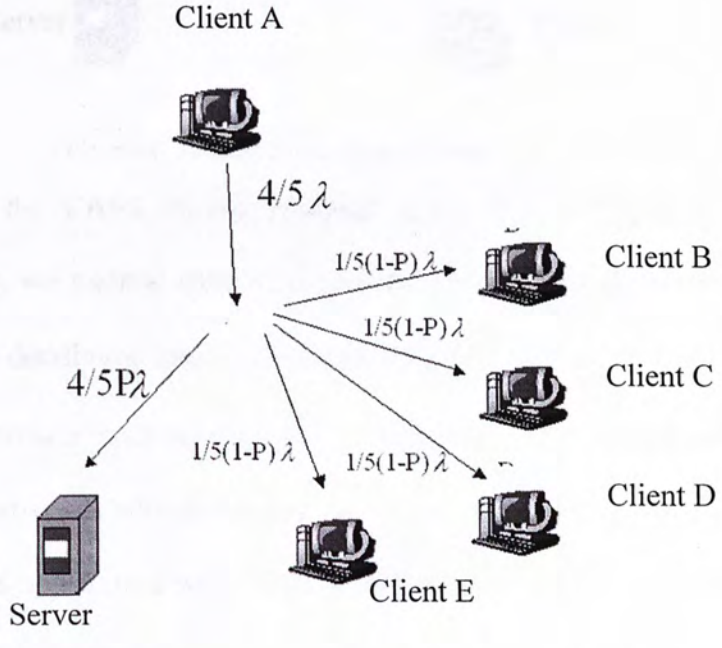


Figure 6.5: The remaining 4/5 tasks arrival will allocate between the server and other client's applications

Accordingly, inside a group of five client's applications (each with distinct functions library), the total task arrival will use the price/volume analysis library with rate λ, the total task arrival will use the pattern matching

library with rate λ, and so on. Therefore, each client's application will see a task arrival rate of λ that is allocating between itself and server as show in figure 6.6.

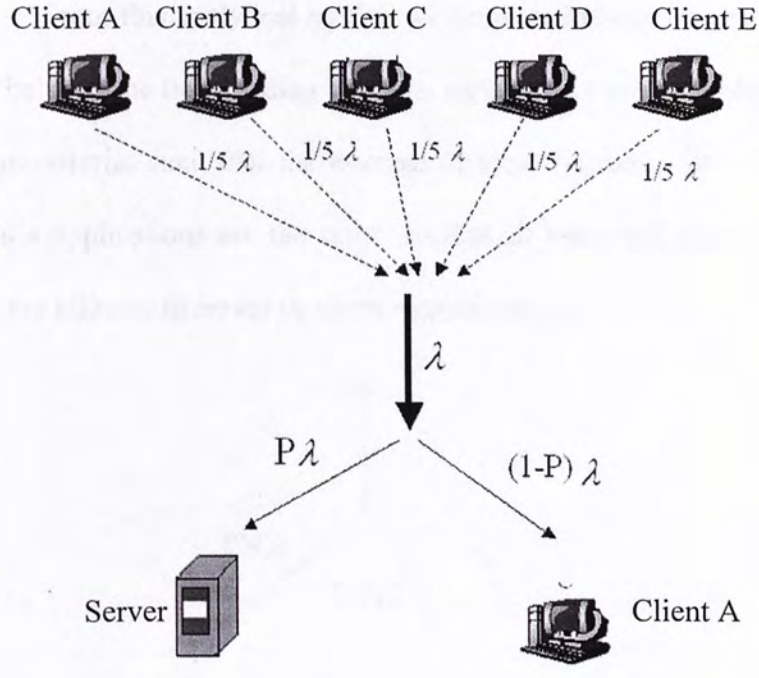


Figure 6.6: Task arrival rate allocate between a client and a server.

For the CASS system contains $N$ ($N$ is a multiple of 5) client's applications, we assume each of the client's application generates tasks with exponential distributed inter-arrival time and with rate equal to $\lambda$, hence there are $N$ task streams each generating tasks with rate $\lambda$. The tasks generated in the $N$ task streams will allocate among the server and the $N$ client's applications, each client's application will seen one task stream and the server will seen all the $N$ task streams as show in figure 6.7.

We further assume:

- All client's applications and server has queue with infinite queue length to queue the incoming tasks for processing,

- Each client's application with services rate equal to $\mu_c$, server with services rate $\mu_s$.

From this analytical model, we want to find out the probability $P$ which will balance the task loading between server and client's application according to the criteria: such that the average services response time of server and all client's applications are the same, so that all tasks will serve fairly no matter they are allocate to server or client's applications.

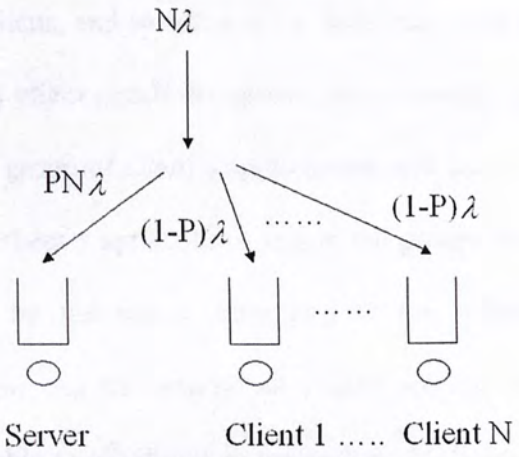


Figure 6.7: The analytical model of the CASS system.

## 6.4 Performance Analysis of the CASS System

In order to satisfy the load balancing criteria, we equate the average services response time of the server ($T_s$) with the client's application ($T_c$) to find out the server's task allocating probability $P$.

The average services response time of server is:

$$T_s = \frac{\frac{1}{\mu_s}}{1 - \frac{P\alpha N\lambda + kN\lambda}{\mu_s}} \qquad (6.1)$$

Where $\mu_s$ is server's services rate, $T_s$ is the server average services response time, $N$ is the numbers of clients, $\lambda$ is the arrival rate generated by each client's application, $k$ is the percentage of tasks produced by each client's application that only allocate to server and $\alpha = 1 - k$. We add factor $k$ into the equation as even five client's applications group together, they are not exactly ensemble all functions as server has, and sometime if the number of clients in the system are not the multiple of five, some of the client's group will have fewer than five clients, and so some of the tasks may only process in the server. Also, due to every client inside the system can go on-line and off-line randomly, we assume every group of client's applications will have some of the time that do not have five client's applications inside the group, therefore $k$ is measured in time average by the server according to the information given in the contracting process. As we assume all clients are the same which mean the factor $k$ is applicable to all clients in the system. If clients in the system go on-line and off-line frequently, the $k$ will become quite large, so we may make a group contains 10 client's application together, with 2 client's application for each functions library, and the percentage of tasks produced by each client's application that must executed by server can reduce. But certainly we need to collect information after the CASS system has been build up to determine the suitable group size for the system. For simplicity, we will use group size of 5 in following discussion.

The average service response time of client's application is:

$$T_c = \frac{\frac{1}{\mu_c}}{1 - \frac{(1-P)\alpha\lambda}{\mu_c}} \tag{6.2}$$

Where $T_c$ is client's average service response time and $\mu_c$ is client's services rate. Other symbols are the same as the pervious definition.

We set $T_s = T_c$ to find out the probability $P$,

$$\Rightarrow \frac{1}{\mu_s - (P\alpha N\lambda + kN\lambda)} = \frac{1}{\mu_c - (1-P)\alpha\lambda} \tag{6.3}$$

$$\Rightarrow \mu_s - \mu_c - kN\lambda + \alpha\lambda = P\alpha N\lambda + P\alpha\lambda$$

$$\Rightarrow \quad P = \frac{\mu_s - \mu_c}{(N+1)\alpha\lambda} + \frac{\alpha - kN}{(N+1)\alpha} \tag{6.4}$$

We note that equation (6.4) only valid for $\alpha > 0$, that mean it's only valid for there exists some of the tasks that can allocate to the client's applications. If all tasks must process in server, then the system become M/M/1 system, and equation (6.4) will be useless.

For $0 < P < 1$,

$$\Rightarrow \quad \frac{\mu_s - \mu_c}{(N+1)\alpha\lambda} + \frac{\alpha - kN}{(N+1)\alpha} > 0 \quad \text{and} \quad \frac{\mu_s - \mu_c}{(N+1)\alpha\lambda} + \frac{\alpha - kN}{(N+1)\alpha} < 1$$

$$\Rightarrow \quad \mu_s - \mu_c + \alpha\lambda - kN\lambda > 0 \quad \text{and} \quad \mu_s - \mu_c + \alpha\lambda - kN\lambda < (N+1)\alpha\lambda$$

$$\Rightarrow \quad \mu_s - \mu_c + \alpha\lambda > kN\lambda \qquad \text{and} \quad \mu_s - \mu_c < N\lambda$$

$$\Rightarrow \quad \frac{\mu_s - \mu_c}{\lambda} < N < \frac{\mu_s - \mu_c + \alpha\lambda}{k\lambda} \tag{6.5}$$

Let's look at equation (6.5), assume $\lambda$ is fixed and $\mu_s - \mu_c > 0$. When $N$ increase from zero to $(\mu_s - \mu_c)/\lambda$, $P = 1$, therefore all tasks will allocate to the

server as the average services response time of server ($T_s$) is smaller than the average servicing time of client's application ($1/\mu_c$). While $N$ increase from $(\mu_s-\mu_c)/\lambda$ to $(\mu_s-\mu_c+\alpha\lambda)/k\lambda$, $0 < P < 1$, and tasks will allocate between server and client's applications according to $P$ as $T_s$ is larger than the average servicing time of client's application. For $N$ larger than $(\mu_s-\mu_c+\alpha\lambda)/k\lambda$, $P$=0, no task will allocate to server because by receiving the $kN\lambda$ arrival of tasks (that must process by server) has already make the average services response time of server ($1/(\mu_s-kN\lambda)$) larger than the average services response time of the client's applications ($1/(\mu_c-\alpha\lambda)$). In the following discussion, we assume equation (6.5) is satisfied and so $0 < P < 1$.

Let's look at $\frac{\alpha - kN}{(N+1)\alpha}$ in equation (6.4). For simplicity, we first assume $\alpha$=1 and $k$=0, then it become $1/(N+1)$. This term means we first divide the task allocation probability by ($N$+1) as there are $N$ client's applications and one server, then how much more or less the probability of task will allocate to server depend on the term $\frac{\mu_s - \mu_c}{(N+1)\alpha\lambda}$ in equation (6.4), where $\frac{\mu_s - \mu_c}{(N+1)\alpha\lambda}$ determine how much the servicing rate provided by the server is superior to the client's applications in the CASS system. For $\alpha$ decrease from 1 and $k$ increase from 0, $\frac{\alpha - kN}{(N+1)\alpha}$ will decrease, which means tasks allocated to server will have smaller probability. Because when $\alpha$ decrease from 1 and $k$ increase from 0, tasks can be only executed by server increased, and hence server has already

receive some tasks, therefore the tasks that can allocate between server and client's applications should allocated to server with a smaller probability.

Let's look at $\frac{\mu_s - \mu_c}{(N+1)\alpha\lambda}$ from equation (6.4), we find probability $P$ depend on how much the server can execute tasks faster than client's application ($\mu_s$-$\mu_c$), if the server can execute tasks much faster than client's application ($\mu_s$-$\mu_c$ is large), then the probability $P$ increase. This is reasonable as this means server can execute more tasks than client's application within the same amount of time.

However, probability $P$ decreases as the total task arrival rate can be performed by client's applications [($N$+1)$\alpha\lambda$] increase. The total tasks arrival rate can be performed by client's applications increase with the increase of the number of client's applications in the system ($N$+1), the tasks arrival rate generate by each client's application ($\lambda$) and the percentage of tasks can perform by clients application ($\alpha$). With the number of client's applications increased in the system (increase of $N$+1), the computing power provided by the client's applications increase, therefore probability $P$ decrease and hence more tasks allocate to client's applications is reasonable. With the increase of tasks arrival rate can be performed by each client's application (increase of $\alpha\lambda$), $T_s$ will increase and at some point it will reach $T_c$, and so decrease the probability $P$ can balance $T_s$ with $T_c$, otherwise the $T_s$ will become larger than $T_c$, and thus tasks allocate to server will perform poor than those allocate to client's applications in the sense of having longer services response time.

Now we want to find out the services response time of the CASS system for $0< P < 1$.

Substitute equation (6.4) into equation (6.1),

$$\Rightarrow T_s = \frac{1}{\mu_s - \left( \frac{(\mu_s - \mu_c)N + \alpha N\lambda - kN^2\lambda}{(N+1)} + kN\lambda \right)}$$

$$\Rightarrow T_s = \frac{1}{\mu_s - \frac{(\mu_s - \mu_c)N + \alpha N\lambda - kN^2\lambda + kN^2\lambda + kN\lambda}{(N+1)}}$$

$$\Rightarrow T_s = \frac{1}{\mu_s - \frac{(\mu_s - \mu_c)N + N\lambda}{N+1}}$$

$$\Rightarrow T_s = \frac{N+1}{\mu_s + (\mu_c - \lambda)N} \qquad (6.6)$$

From equation (6.6), we see that when the number of client's applications in the system and the task arrival rate generated by each client's application increase, the average services response time of the system will increase ($T$), and if the services rate of the server or client's application increase, then $T$ decrease. Also, we observe that if $\mu_c$ and $\lambda$ are fixed and $(\mu_c\text{-}\lambda) > 0$, $T_s$ will approach and bounded by a limit as $N$ approach infinity for $k$=0.

For $\mu_c - \lambda > 0$ and $k = 0$, when $N \to \infty$, the average services response time of the system is:

$$T_{N\to\infty} = \frac{1}{(\mu_c - \lambda)} \qquad (6.7)$$

If $k > 0$, when n increase, $P \to 0$, equation (6.3) must satisfy the following conditions:

$$\mu_s - kN\lambda > 0 \text{ and } \frac{1}{\mu_s - kN\lambda} \leq \frac{1}{\mu_c - \alpha\lambda}$$

$$\Rightarrow \quad N < \frac{\mu_s}{k\lambda} \text{ and } N \leq \frac{\mu_s - \mu_c + \alpha\lambda}{k\lambda} \tag{6.8}$$

Condition (6.8) imply if $k > 0$, for $N \leq \frac{\mu_s - \mu_c + \alpha\lambda}{k\lambda}$, the system services response time will bound by (6.7) with fixed $\mu_c$ and $\lambda$, if $N$ larger than $(\mu_s - \mu_c + \alpha\lambda)/k\lambda$, equation (6.3) will not satisfy, and hence the probability equation in (6.4) will be useless. And the services response time of server will larger than client's applications as mention in pervious discussion.

We define $N_{max}$ as the maximum number of clients that can be serve by the CASS system, where $N_{max} = \min\{\frac{\mu_s}{k\lambda}, \frac{\mu_s - \mu_c + \alpha\lambda}{k\lambda}\}$ for $\mu_c - \lambda > 0$. If $\mu_s >> \alpha\lambda - \mu_c$, then $N_{max} \cong \frac{\mu_s}{k\lambda}$. When compare $N_{max}$ with the maximum number of clients that can be served by M/M/1 system ($\frac{\mu_s}{\lambda}$), we observe that the system capacity of CASS system is $1/k$ times larger than that of M/M/1 system. Therefore if the percentage of tasks must be executed by server is small, then the capacity of CASS system is much larger than that of M/M/1 system because most of the tasks can be executed by client's applications in CASS system, hence when the amount of tasks increase with the number of clients in the system, it's will allocate more tasks to client's applications, keeping the server

from over loading. For $\mu_c - \lambda < 0$, the average services response time of the system will approach infinity as $N$ increase.

If $\mu_c - \lambda < 0$ and $k > 0$, when n increase, $P \to 0$, equation (6.6) and (6.3) must satisfy the following conditions:

$$\mu_s - (\mu_c - \lambda)N > 0 \text{ and } \frac{1}{\mu_s - kN\lambda} \leq \frac{1}{\mu_c - \alpha\lambda}$$

$$\Rightarrow N < \frac{\mu_s}{\lambda - \mu_c} \text{ and } N \leq \frac{\mu_s - \mu_c + \alpha\lambda}{k\lambda} \qquad (6.9)$$

Equation (6.9) is quit similar with (6.8), which show the maximum number of clients can be support by the CASS system for $\mu_c - \lambda < 0$.

$N_{max} = \min\{\frac{\mu_s}{\lambda - \mu_c}, \frac{\mu_s - \mu_c + \alpha\lambda}{k\lambda}\}$, for $\mu_c - \lambda < 0$. If $\mu_s >> \alpha\lambda$-$\mu_c$, $\frac{\mu_s - \mu_c + \alpha\lambda}{k\lambda}$ is approximately equal $\frac{\mu_s}{k\lambda}$, hence $N_{max} = \min\{\frac{\mu_s}{\lambda - \mu_c}, \frac{\mu_s}{k\lambda}\}$. So, the system capacity of CASS is min $\{\frac{\lambda}{\lambda - \mu_c}, 1/k\}$ times of the M/M/1 system. Therefore, the CASS system will have a larger system capacity than M/M/1 system, as some of the tasks can be performed by client's applications.

## 6.5 Performance Simulation

We have developed a testbed system with simulation server and client using the MCNP for task allocation in UNIX operating system. We assume:

- The tasks inter-arrival time generated by each client is exponential distributed,

- The services rate of simulation server and client are exponential distributed.

In the first simulation, we will compare the performance of CASS system with M/M/1 system for $\mu_c - \lambda > 0$ as the number of clients in the system increase. And then we will compare the performance of CASS system with M/M/1 system for $\mu_c - \lambda < 0$ as the number of clients in the system increase.

We have used up to 11 machines connected by a 100 MBS Ethernet network to perform the simulation, one machine for server and ten machines for clients. In the simulation, when a client logon in the system, it will search the active clients list inside the server to find a client group to join. We use group size of 5, and we assume no client will logoff during simulation. For simplicity, we assume tasks that must execute in server are only generated when there are not enough clients in the system to form client group with size of 5. Other tasks will allocate between a client and server by the MCNP, the server's bid will contain it's services rate, the percentage of tasks that only allocated to itself and the average CFP arrival rate seen by itself; the bids send by contractor client will contain it's services rate and the average tasks arrival rate seen by itself. The manager client uses these information and equation (6.4) to find out the probability for task allocation, and then generate the task allocation result according to that probability. And we assume tasks are equally distributed on the 5 different functions libraries.

In the first simulation $\mu_c - \lambda > 0$, the simulation client's services rate is 3.5 tasks per second; the tasks arrival generated by each simulation client is 1.0

task per second. The simulation server's services rate is 10.0 tasks per second. Table 6-1 show the percentage of total tasks that must executed by server with different number of clients in the CASS system. This factor $k$ is not applicable to all clients, as clients will not go off-line in our simulation, hence the forming of groups are fixed, and only those clients in the group with size smaller than five will generate that must executed by server. Thus induce the estimated probability $\boldsymbol{P}$ in different clients will be different, which cause small variation in the average services response time of clients. But when $N$ increases, $k$ will decrease, and the variation of average service response time of client will also decrease.

| N | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| $k$ (%) | 80 | 60 | 40 | 20 | 0 | 13.3 | 17.1 | 15 | 8.8 | 0 |
| N | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| $k$ (%) | 7.2 | 10 | 9.2 | 5.7 | 0 | 5 | 7 | 6.6 | 4.2 | 0 |

Table 6-1: Percentage of tasks ($k$) that must executed by server with different number of clients in the CASS system

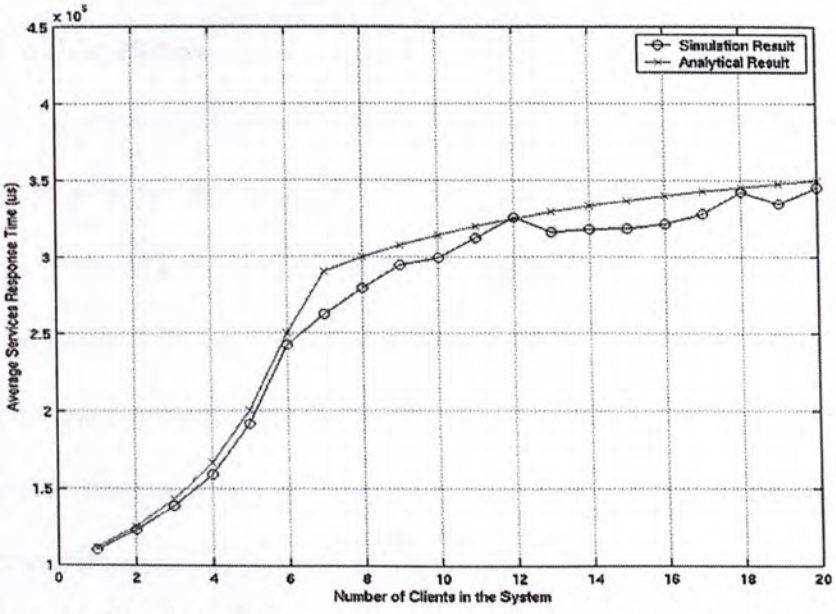


Figure 6.8: The analytical and simulation result of the average services response time (in micro-second) with different number of clients in the CASS system for $\mu_c$-$\lambda$>0.

In figure 6.8 the line with circle show the simulation result of the system average services response time (in micro-second) vs. the number of clients in the system for $\mu_c - \lambda > 0$. It is according to our analysis result (the line with cross).

Since bids evaluation can only start after receiving all bids, so we start bids evaluation after biding deadline. We give 20 msec for contractors to bid. The average protocol delay of MCNP is about 23 msec in the simulation. And about 0.8% of the contracting process needs to restart due to cannot receive all bids before biding deadline has come. For $N < 7$, $P=1$ in most of the time. For N > 7, the average services response time of each client is about ±20 msec different from the average services response time of the server. This is due to the factor $k$ in the simulation are not applicable to all clients as some fixed clients cannot form a group of five clients, hence only these clients will generate tasks that must process by server and their average service response time will be smaller.

Comparing figure 6.8 with figure 6.9, it show our CASS system can support at least one time more clients than M/M/1 system with the same server's services rate in the simulation. As we doesn't have and actually value on the factor $k$ in true life, we could not get the exact number that the CASS system can support in this setting. For less than 20 clients, the average services response time of our CASS system is bounded below 400 msec, but it has reaches 500 msec for 8 clients in the M/M/1 system.

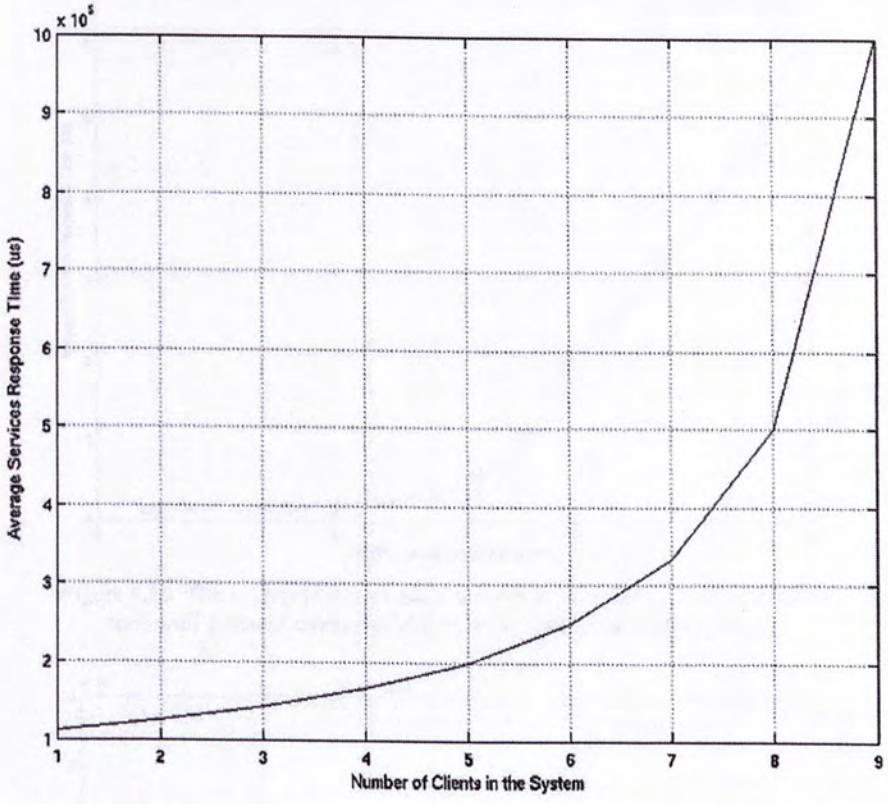


Figure 6.9: The average services response time (in micro-second) with different number of clients in the M/M/1 system.

In the second simulation $\mu_c - \lambda < 0$, the simulation client's services rate is 0.5 tasks per second; the task arrival generated by each simulation client is 1.0 task per second. The simulation server's services rate is 10.0 tasks per second.

In figure 6.10 the line with circle show the simulation result of the average services response time (in micro-second) vs. number of clients in the system for $\mu_c - \lambda < 0$. It is according to our analysis result (the line with cross).

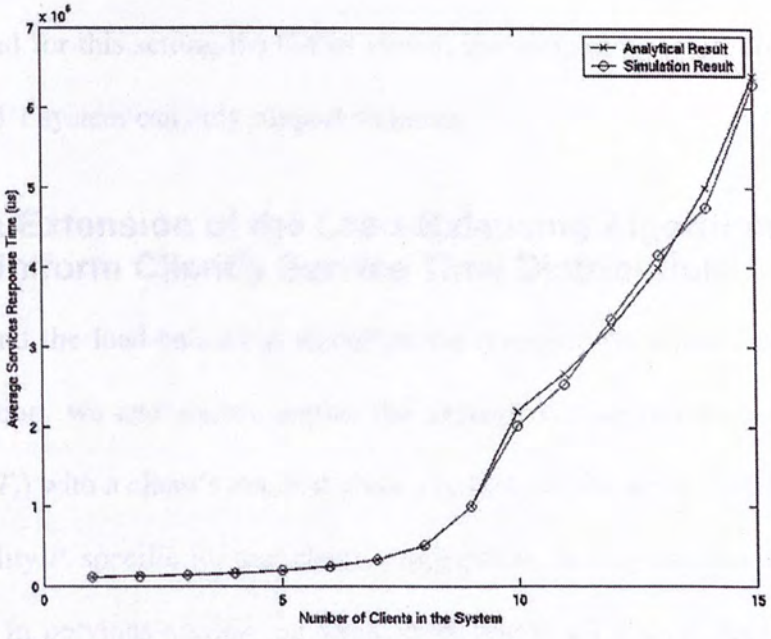


Figure 6.10: The analytical and simulation result of the average services response time with different number of clients in the CASS system for $\mu_c$-$\lambda$<0.

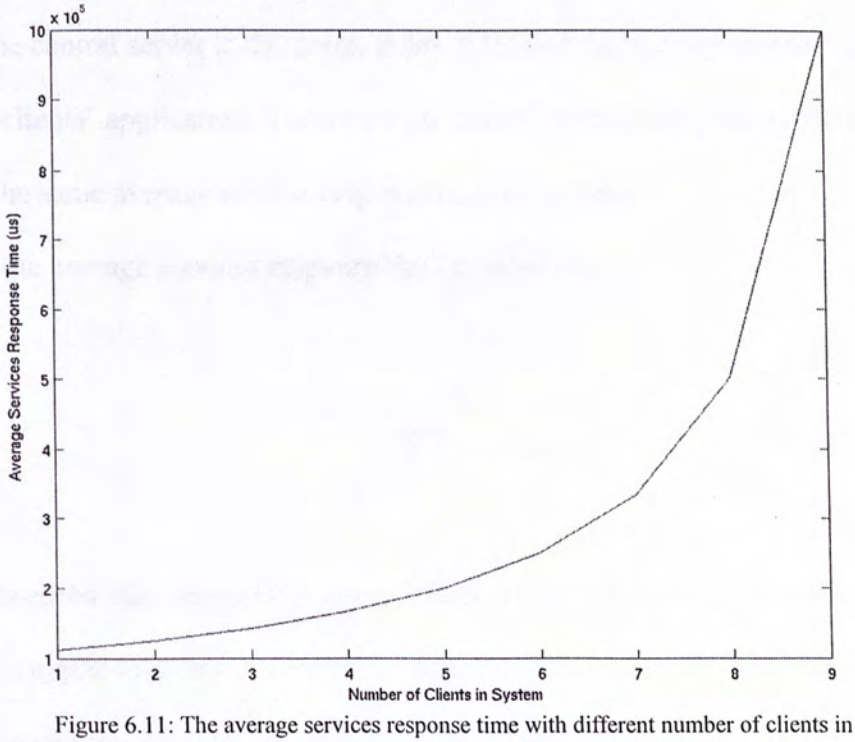


Figure 6.11: The average services response time with different number of clients in the M/M/1 system.

Comparing figure 6.10 with figure 6.11, it show our CASS system can support more clients than M/M/1 system with the same server's services rate in the simulation. But the system services response time is not bounded in this

case. And for this setting the CASS system can support up to 19 clients, while the M/M/1 system can only support 9 clients.

## 6.6 An Extension of the Load-Balancing Algorithm for Non-Uniform Client's Service Time Distribution

To extend the load-balancing algorithm for non-uniform client's service time distribution, we can simply equate the average service response time of the server ($T_s$) with a client's application ($T_r$) to find out the server's task allocating probability $P_r$ specific for that client's application, as long as other assumptions defined in pervious section are valid. And, due to all clients' application will equate their average service response time with the one of the central server, thus the central server is the center point to link up the average service response of all clients' application. Therefore, all clients' application and the server will have the same average service response time as expected.

Now, the average services response time of server is:

$$T_s = \frac{\frac{1}{\mu_s}}{1 - \frac{\sum_{i=1}^{N} P_i \alpha\lambda + kN\lambda}{\mu_s}} \tag{6.10}$$

We observed that equation (6.10) is different from equation (6.1), because each client's application has it's own task allocating probability $P_i$. For $i$ = 1,2, ..., $N$. The average service response time of client's application $r$ is:

$$T_r = \frac{\frac{1}{\mu_r}}{1 - \frac{(1-P_r)\alpha\lambda}{\mu_r}} \tag{6.11}$$

Where $\mu_r$ is the servicing rate of client $r$.

We set $T_s = T_r$ to find out the probability $P_r$,

$$\Rightarrow \frac{1}{\mu_s - (G_r\alpha\lambda + P_r\alpha\lambda + kN\lambda)} = \frac{1}{\mu_r - (1-P_r)\alpha\lambda} \qquad \text{Where } G_r = \sum_{i \neq r}^{N} P_i$$

$$\Rightarrow \mu_s - \mu_r - G_r\alpha\lambda - kN\lambda + \alpha\lambda = 2P_r\alpha\lambda$$

$$\Rightarrow P_r = \frac{\mu_s - \mu_r}{2\alpha\lambda} + \frac{\alpha(1-G_r) + kN}{2\alpha} \qquad (6.12)$$

As the central server and each client's application can measure the incoming task arrival rate in real-time, therefore we could find out $G_r$ in real-time easily. Hence, we could use equation (6.12) to find out the task allocation probability of each client's application for the system with non-uniform client's service time distribution.

## 6.7 Summary

We have design a Client AS Server (CASS) system model enabled by the Modified Contract Net Protocol (MCNP) for the on-line data analysis service of the Real-time Financial Data Mining system in this chapter. The task allocation process is simple, as a task will only allocate between two contractors. The CASS system capacity is $\frac{\lambda}{\lambda - \mu_c}$ or $1/k$ times of the M/M/1 system for server's services rate: $\mu_s >> \alpha\lambda$-$\mu_c$, where $\mu_c$ is the services rate of client's application, $\lambda$ is the tasks arrival rate generated by each client, and $k$ is the percentage of tasks that must execute in server. Thus showing a task allocation method combine with the task allocation protocol (MCNP) can enable the potential benefit of

employing distributed computing. The CASS system certainly showing the power of dynamic task allocation: when there are few clients in the system, most tasks will allocated to server to get a fast execution, but when the number of clients in the system increased, allocate some tasks to clients for execution will better than allocating all tasks to the server. And we have design a extend algorithm for non-uniform client's service time distribution. If tasks are not equally distributed on the five function libraries or the task arrival rate generated by each client's application is different, we will need to make an extended load-balancing algorithm for these changes using the same way as we did in section 6.6 to meet the load-balancing criteria. But we would expect that we could not find the system capacity using the extended load-balancing algorithm. Also, from the CASS system we cannot show the power of parallel processing for increasing the speed of executing a number of tasks, as our implementation of Real-time Financial Data Mining system haven't complete, thus we don't have real tasks structure to design a model for parallel processing.

# Chapter 7

# Conclusion and Future Work

## 7.1 Conclusion

The success of a Real-time Financial Data Mining system will depend on the sophistication and speed of the system, and its ability to analyze and synthesize information with simple operations by users. Therefore, a super-computing machine is required for the system, but very few clients have machines that can meet the requirements of running such a sophisticated system at high speed. By employing client/server architecture, providing a supercomputing machine in the server-side, clients with different computing competence can enjoy the sophisticated system. But due to computing process is highly centralized in the server-side, clients computing power will be wasted. Therefore, we modify the contract net protocol and design a task allocation method to enable distributed computing for tapping the computing power of idle client's PCs in the system.

In this thesis we have outline the system and software architecture for the Real-time Financial Data Mining system. With the design of the system and software architecture plus a dynamic task allocation protocol, it is possible to get benefit from tapping the computing power of idle client's PCs in system and turning them into a "poor man's supercomputer," reducing the time it takes to do calculations and simulations for financial forecasting. But there is also an

important assumption to make the benefit possible, which is client's applications need to be in low tasks loading on time average. If client's applications often in busy status, then they can provide a relatively less time for helping other clients applications to perform tasks execution or data mining actions. A dynamic task allocation protocol is preferred instead of a static one, because client's applications may go on-line and off-line in an unpredictable way, and thus static task allocation is not possible in our system.

Then we have explored the potential of the Contract Net Protocol (CNP) as a dynamic task allocation protocol. We observe that CNP is a simple, dynamic and decentralize task allocation protocol. The CNP provide a flexible framework for task allocation. We base on the FIPA contract net Interaction Protocol (IP) to develop our Modified Contract Net Protocol (MCNP). We introduce the starting time of bid(s) evaluation in the call for proposal (CFP) message, which allow manager to set the time for start of bid(s) evaluation earlier than the bidding deadline. And we add the valid time of the bid message inside the bid message, which prevent the bid message storing in the protocol message queue forever. Finally, according to T. Knabe [15], we add the award confirmation message ensures the contractor received the award and delay the time of contract commitment. Through our design and implementation of the MCNP, it's able to handle multiple contracting sessions and send out multiple CFPs without waiting a contracting process to complete before starting another one.

Further more, we have design a MCNP enabled Client As Server (CASS) system model. It is shifting the task computing between clients and server according to the real-time task loading information. The CASS system tapping the computing power of client's PCs in system and turning them to work with the middle layer server, increasing the system capacity when compare with using single-server queueing (M/M/1) system. We have show the CASS system capacity is $\frac{\lambda}{\lambda - \mu_c}$ or $1/k$ times of the M/M/1 system for server's services rate: $\mu_s >> \alpha\lambda$-$\mu_c$, where $\mu_c$ is the services rate of client's application, $\lambda$ is the tasks arrival rate generated by each client, and $k$ is the percentage of tasks that must execute in server. And the simulation result of the CASS system certainly showing the power of dynamic task allocation: when there are few clients in the system, most tasks will allocated to server to get a fast execution, but when the number of clients in the system increased, allocate some tasks to clients for execution will better than allocating all tasks to the server. Thus showing a task allocation method combine with the task allocation protocol (MCNP) can enable the potential benefit of employing distributed computing.

## 7.2 Future Work

We have introduced a statistical task allocation method enabled by MCNP in chapter 6. But certainly we need a new task allocation method using the MCNP to make parallel computing possible in the Real-time Financial Data Mining system, in order to show the power of tapping the computing power of idle

client's PCs in the system and turning them into a "poor man's supercomputer," reducing the time it takes to do calculations and simulations for financial forecasting. And the continuous of the implementation of the Real-time Financial Data Mining system will benefit in giving a concrete system and task structure for the development of the task allocation method for parallel computing.

# Bibliography

[1] Smith, R.G., "The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver". *IEEE Trans. on Computers* C-29(12):1104-1113, 1980.

[2] Smith, R.G., and Davis, R., "Frameworks for Cooperation in Distributed Problem Solving." *IEEE Trans. on Systems, Man, and Cybernetics* 11(1):61-70, 1981.

[3] FIPA Contract Net Interaction Protocol Specification. Available WWW <URL: http://www.fipa.org/specs/fipa00029/ > (1995).

[4] Gerhard Weiss. "Multiagent Systems, A Modern Approach to Distributed Artificial Intelligence". *The MIT Press*, 100-103 & 233-239, 1999.

[5] Schussel, George. "*Client/Server Past, Present, and Future.*" 1995.

[6] Edelstein, Herb. "Unraveling Client/Server Architecture." *DBMS 7,* 5, May, 1994.

[7] Louis [online]. Available WWW <URL: http://www.softis.is> (1995).

[8] Eckerson, Wayne W. "Three Tier Client/Server Architecture: Achieving Scalability, Performance, and Efficiency in Client Server Applications." *Open Information Systems 10,* 1, January, 1995.

[9] Webopedia, "PC Webopedia Definitions and Links". Available WWW <URL: http://webopedia.internet.com> 1999

[10] Sycara, K., Decker, K., Pannu, A., Williamson, M. and Zeng, D., "Distributed intelligent agents." *IEEE Expert, Intelligent Systems and their Application*, 11(6):36-45, 1996.

[11] Paolucci, M., Onn Shehory and Sycara, K., "Interleaving Planning and Execution in a Multiagent Team Planning Environment." In *Electronic Transactions of Artificial Intelligence*, 2001.

[12] Paolucci, M., Shehory, O., Sycara, K., Kalp, D. and Pannu, A. "A Planning Component for RETSINA Agents." Lecture Notes in Artificial Intelligence, Intelligent Agents VI. M. Wooldridge and Y. Lesperance (Eds.), forthcoming.

[13] K. Decker, A.S. Pannu, K. Sycara, and M. Williamson, "Designing Behaviors for Information Agents". *Proceedings of the First International Conference on Autonomous Agents*, February, 1997.

[14] Sycara, K., Williamson, M., and Decker, K. "Unified Information and Control Flow in Hierarchical Task Networks." *Working Notes of the AAAI-96 Workshop, "Theories of Action, Planning, and Control."* Aug., 1996.

[15] T. Knabe, M. Schillo, and K. Fischer. "Improvements to the FIPA contract net protocol for performance increase and cascading applications." In *International Workshop for Multi-Agent Interoperability at the German Conference on AI (KI-2002)*, 2002.

[16] Sandholm, T. W. and Lesser, V. R., "Advantages of a Leveled Commitment Contracting Protocol". *Proceedings of AAAI-96*, Portland, OR, 126-133, 1996.

[17] Hluchý L., Dobrucký M., Astalos' J.: "Hybrid Approach to Task Allocation in Distributed Systems." *Computers and Artificial Intelligence*, Vol.17, No.5, 1998, pp. 469-480.

[18] A. Takefusa, H. Casanova, S. Matsuoka, and F. Berman. "A Study of Deadline Scheduling for Client-Server Systems on the Computational Grid." *Proceedings of 10th IEEE International Symposium on High Performance Distributed Computing (HPDC-10)*, pp. 406-415, 2001.08.