USING ECONOMIC MODELS TO TUNE RESOURCE ALLOCATIONS IN DATABASE MANAGEMENT SYSTEMS

by

Mingyi Zhang

A thesis submitted to the School of Computing

In conformity with the requirements for

the degree of Master of Science

Queen's University

Kingston, Ontario, Canada

(November, 2008)

Abstract

Resource allocation in a database management system (DBMS) is a performance management process in which an autonomic DBMS makes resource allocation decisions based on properties like workload business importance. We propose the use of economic models in a DBMS to guide the resource allocation decisions. An economic model is described in terms of business trades and concepts, and it has been successfully applied in some computer system resource allocation problems.

In this thesis, we present approaches that use economic models to allocate single and multiple DBMS resources, such as main memory buffer pool space and system CPU shares, to workloads running concurrently on a DBMS based on the workloads' business importance policies. We first illustrate how economic models can be used to allocate single DBMS resources, namely system CPU shares, to competing workloads on a DBMS. We then extend this approach to using economic models to simultaneously allocate multiple DBMS resources, namely buffer pool memory space and system CPU shares, to competing workloads on a DBMS based on the workload business importance policy in order to achieve their service level agreements. Experiments are conducted using IBM® DB2® databases to verify the effectiveness of our approach.

Acknowledgements

I would like to extend my sincerest thanks to my supervisor, Dr. Patrick Martin, for bringing me into the wonderful Autonomic DBMS world. I would like to thank Dr. Martin for all his great guidance, advice, and support throughout these two years of my studies in School of Computing at Queen's University.

I would like to heartily thank Wendy Powley for her great help during the years when I was pursuing my research at Database Systems Laboratory in School of Computing at Queen's University. Without Dr. Martin and Wendy's help, this research would be impossible. I would also like to thank to Paul Bird for all his help with this research. Additional thanks to the members of Database Systems Laboratory in School of Computing at Queen's University for their opinions and friendship.

I would like to thank Queen's University, IBM Canada Ltd., Toronto, Ontario, Canada, Natural Science and Engineering Research Council of Canada, and the Centre for Communication and Information Technology, a division of Ontario Centers of Excellence Inc for the financial support.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

With the rapid development of information technology (IT) over the past 20 years, IT infrastructures, such as the Internet, computer systems, and software applications, have significantly accelerated the pace, reach, and agility of business [8]. The fast growing IT infrastructures, however, have also introduced a huge amount of complexity into today's business and IT industry itself. The consequence of the growing complexity is the damage to an enterprise caused by more system downtime, higher costs, and inferior customer service [37]. Since the trend of IT infrastructures appears to be towards more complex, thus more difficult to manage systems, the complexity crisis is becoming the main obstacle to further progress in the IT industry [18].

In particular, database management systems (DBMSs), as the core component of a computing system in many organizations, are often integrated with other computing systems. A DBMS may have hundreds of parameters which can be tuned for the system to perform optimally, and performance tuning on one large subsystem may lead to unanticipated effects on the entire system [18]. As computing systems are becoming increasingly complex, business organizations require more and more skilled IT professionals to install, configure, tune, and maintain their systems. It is estimated that one-third or one-half of a company's total IT budget is spent on maintenance and development of existing enterprise computing systems [11]. In addition, the emerging trend to server consolidation and service-oriented architectures are making the management of DBMSs even more complicated. In order to minimize cost and maximize investment return, business organizations consolidate their databases onto one database server.

This trend, for a database server, means that a single server needs to be able to simultaneously handle multiple workloads with diverse characteristics and dynamic resource demands while at the same time, satisfy the workloads' service level agreements (SLAs).

An option to address this complexity issue is to apply IBM's autonomic computing technology on the computing systems. Autonomic computing is a term used to describe a broad range of standards, architecture, products, services and tools that enable technology systems to be self-managing [17]. An autonomic computing system, in other words, can be simply described as a system that manages itself with the high-level objectives given by the system administrators. The goal of an autonomic computing system is to shift tasks such as system configuration, maintenance, and fault management from people to technology. An autonomic computing system is capable of taking over manually intensive IT operations, many of which are routine and repetitive [17]. One of the efforts towards self-managing computing systems involves automatic resource allocation in autonomic DBMSs to ensure that competing workloads running on the systems meet their performance requirements. Such requirements may be based on a business-level workload management policy such as workload business importance policies.

1.2 Problem

As introduced above, with the emerging trend for enterprises to consolidate workloads onto a single database server, the management of the diverse variety of workloads running on a DBMS is becoming increasingly complex and costly. Workloads submitted by different applications, or from different business units, may have unique performance requirements, such as response time and throughput requirements, which are normally expressed in terms of SLAs that need to be strictly satisfied [20]. Workloads concurrently running on a database server inevitably compete

for shared system resources, such as CPU, main memory, and disk I/O. As a result, it is possible that the workloads may not acquire sufficient system resources and fail to meet their SLAs.

To meet the performance requirements of database workloads, database administrators (DBAs) may need the workload management policies and DBMS system-level techniques to implement and satisfy these policies. Workload business importance policies, a type of workload management policies, apply business metrics to classify database workloads into multiple classes based on the workloads' business importance. A workload business importance policy can be defined either by DBAs or by IT service providers. A workload may be considered important if it is generated by an organization's CEO or if it is directly revenue producing. Less important workloads might be those pertaining to functions such as human resources or business report generation. It is a challenge for DBAs to tune DBMSs according to a given workload business importance policy as system-level metrics must be defined to measure customers' expectations and the business-level importance policies must be translated into system-level tuning policies in order to impact these metrics.

Autonomic computing suggests that DBMSs are capable of becoming self-configuring, self-tuning, self-protecting, and self-healing [18]. A key feature of autonomic DBMSs is policy-driven management, which includes making resource allocation decisions based on properties such as workload business importance. This feature enables the high-level business policies to be automatically translated into low-level system resource tuning actions. In addition, the feedback loop of autonomic DBMSs verifies that workload SLAs are met and initiates system reconfiguration if the system fails to meet the SLAs. With autonomic DBMSs, system administrators may concern themselves only with the definition of the high-level business policies and let the system take care of the rest.

Resource allocation, as a workload management technique, manages individual DBMS resources and assigns the resource shares to workloads based on business-level administrative objectives such as a workload business importance policy. With the allocated resources, the workloads can be managed to achieve their SLAs when they are concurrently running on one database server. We propose the use of economic models in a DBMS to implement this workload management technique. Economic models have been introduced and successfully applied in some computer system resource management problems [10][19][31][33]. The models incorporate an implicit sense of business trades and concepts, and therefore the workload business importance policies can be easily implemented by economic models and a DBMS can in turn be indirectly managed by these importance policies.

To manage the shared resources in a DBMS and make system-level resource allocation decisions for workloads using an economic model, there are some critical issues inherently existing with the resource allocation problem. These issues include how to partition the individual shared DBMS resources among the competing workloads, how to determine the optimal amounts of each resource for each workload that will not only allow the individual workload to achieve its performance objectives, but also maximize overall system performance, and how to choose a reasonable total amount of resources to allocate, as well as how to predict the workload performance with a given resource allocation so that the model can verify that the workload SLAs will be met.

1.3 Research Statement

The objective of this research is to investigate resource allocation in DBMSs by applying autonomic computing technology and economic concepts to reduce resource management complexity. We introduce an approach that uses economic models to allocate DBMS resources

based on business-level management objectives such as workload business importance policies. A set of methods are proposed for using the economic model to allocate single and multiple DBMS resources for multiple classes of competing workloads. These methods include resource modeling, performance modeling, and a resource allocation method used for partitioning individual resources, predicting system performance and determining bottleneck resources. By applying the economic model, a DBMS is able to automatically translate a high-level workload business importance policy to the system-level resource tuning actions as a step towards a self-managing system.

This research makes several contributions to database workload management through automatically conducting resource allocations in a DBMS. The first contribution of the work is an investigation of the use of the resource allocation framework originally proposed by Boughton [7] for system CPU resource allocations. Boughton [7] tuned buffer pool memory resource allocations by using the resource allocation framework in an autonomic DBMS for multi-class workloads with unique performance requirements. In this study, we extend the framework by defining a method of CPU resource modeling and a CPU resource utility function for the resource allocation framework to allocate system CPU resources.

The second contribution is the specification of a resource allocation framework that extends the functionality of single-resource allocations to the case of multiple-resource allocations for multiple workloads with unique performance requirements. The resource modeling methods are applied to partition the multiple resources, namely buffer pool memory space and system CPU shares, and resource allocation and performance modeling methods are proposed in the framework for the multiple-resource allocations. A multiple-resource utility function is constructed using the performance modeling techniques.

The third contribution of this work is the extension of the economic model simulator created by Boughton [7] to validate the resource allocation framework. We simulate a DBMS environment with several workloads of differing business importance on a single DB2® database server. The workloads compete for the server's limited system resources. We consider the cases of single resource, namely CPU resources, and multiple resources, namely buffer pool memory space and system CPU shares respectively. The simulator suggests resource allocations for the workloads as produced by the economic model. The performances of the multi-class workloads are verified by running the workloads on the DB2 database server.

1.4 Thesis Organization

The remainder of the thesis is organized as follows. Chapter 2 discusses related work found in the literature, as well as a brief description of autonomic computing concepts, a discussion of proposed approaches of resource management in DBMSs, and a short review of economic models for computer system's resource management. Chapter 3 explains our general framework of using economic models for resource allocation in a DBMS. Chapter 4 describes an approach that uses our resource allocation framework to allocate single DBMS resources, namely system CPU shares, to workloads running concurrently on a DBMS based on their business importance. A validation of the approach is then provided. Chapter 5 shows the use of the extended resource allocation framework to simultaneously allocate multiple DBMS resources, namely buffer pool memory space and system CPU shares, to workloads running concurrently on a DBMS. The validation of the resource allocation framework for multi-resource allocation is then presented. We conclude and suggest future work in Chapter 6.

Chapter 2
Related Work

Generally, resource allocation in an autonomic DBMS is a performance management process undertaken by the system to provide timely and correct allocations of limited system resources to workloads based on business-level administrative objectives to optimize workload and system-wide performance [33]. Our study presents an approach that applies autonomic computing technology to automate resource allocation in DBMSs and to automatically translate the business-level workload management objectives into the DBMS system-level resource tuning actions. Our approach is tightly related to two areas of research, namely autonomic computing systems and resource management in DBMSs, and employs a resource management model. In Section 2.1, we review the concepts and evolutionary processes of IBM's autonomic computing. Resource management is a fundamental concept in computer system design and management, and a number of approaches have been proposed for resource management in DBMSs. In Section 2.2, we discuss some proposed techniques of resource management that are specifically relevant to policy-based or goal-oriented workload management by controlling resource allocations in DBMSs. In Section 2.3, we review economic models and their application for resource allocation in a computer system.

2.1 Autonomic Computing

The concept of autonomic computing was first presented by Paul Horn, IBM Senior Vice President and Director of Research, in 2001 [11]. The autonomic computing initiative aims to provide the foundation for systems to manage themselves without direct human intervention in order to reduce the complexity of the computing environment and infrastructure. By choosing the

word *autonomic*, IBM makes an analogy with the human autonomic nervous system which controls activities, such as heartbeat, blood pressure and breathing that enable human body to self-regulate and adapt to changing conditions [18][11]. In particular, autonomic computing is necessary for realizing workload management technology through employing resource allocation in a DBMS as there is no way for DBAs to manually and consistently conduct resource allocations to workloads in order to efficiently manage the dynamic and diversified workloads.

An autonomic computing system, as a self-managing system, has four fundamental properties: *self-configuring, self-optimizing, self-healing*, and *self-protecting* [18]. Self-configuring means computing systems are able to automatically configure components to adapt to dynamically changing environments. The functionality of the property allows the addition and removal of system components or resources without system service disruptions. Self-optimizing means that systems automatically monitor and control the resources to ensure optimal functioning with respect to the defined requirements. Self-healing means that systems are able to recognize and diagnose deviations from normal conditions and take action to normalize them. This property enables a computing system to proactively circumvent issues which could cause service disruptions. Self-Protecting means computing systems are able to proactively identify and protect from arbitrary attacks. This property enables a computing system to recognize and circumvent security threats, and facilitates the system to protect itself from physical harm, such as excessive heat or motion [17].

To implement autonomic computing, the industry must take an evolutionary approach and deliver improvements to current systems that will provide significant self-managing value to customers without requiring them to completely replace their current IT environments [11]. The path to autonomic computing is described in the five levels shown in Figure 1. These levels start at *basic* and continue through *managed, predictive, adaptive* and finally *autonomic* [12].

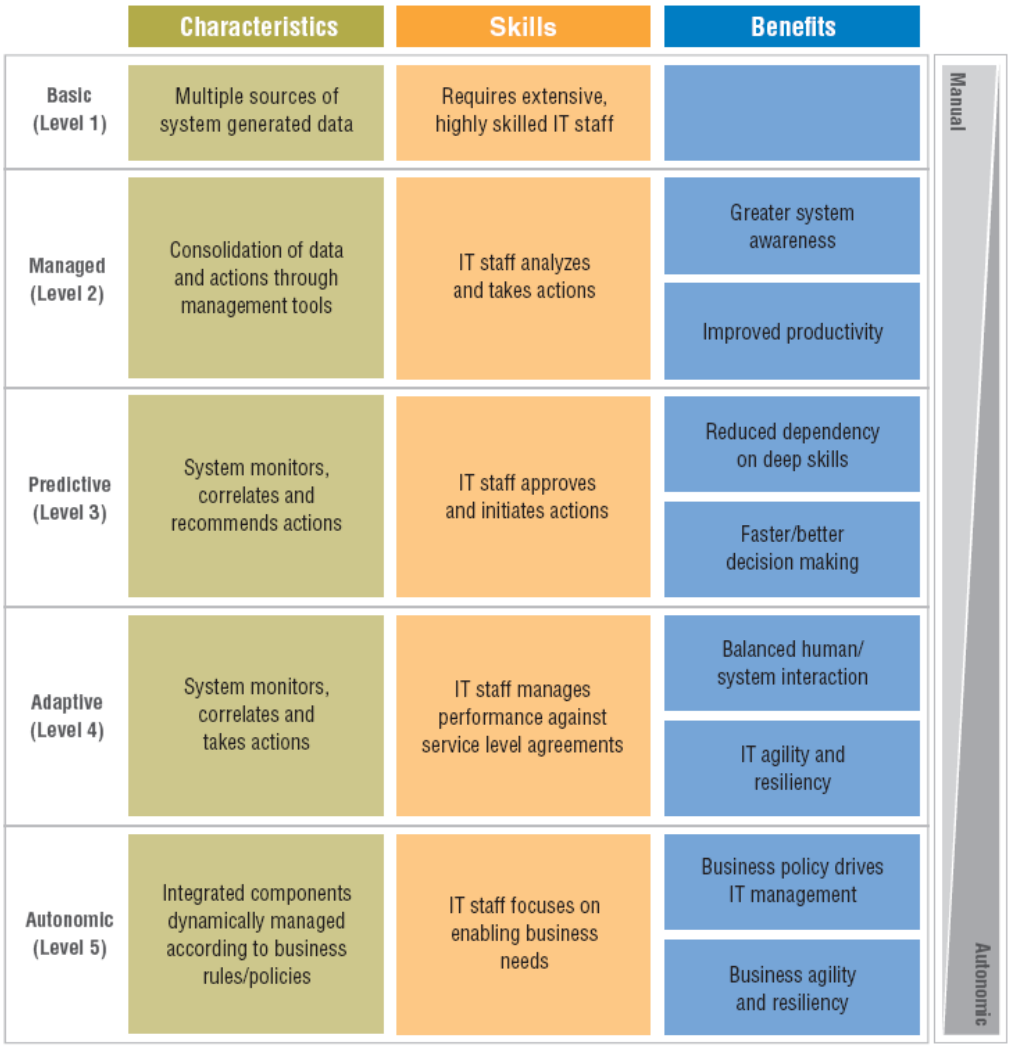

Figure 1: Evolutionary Path of Autonomic Computing [13]

At the basic level, each infrastructure element is manually managed by IT professionals who set it up, monitor and manage it, and eventually replace it. At the managed level, system management technologies can be used to collect and consolidate information from disparate systems and present it on a single console, thus reducing the time it takes for the administrator to collect and synthesize information. At the predictive level, since new technologies are introduced to provide correlation among several infrastructure elements, the system itself can begin to

recognize patterns, predict the optimal configuration and provide advice on what course of action the administrator should take. At the adaptive level, the systems themselves can automatically take action based on the information that is available to them and the knowledge of what is occurring in the system. Finally, at the autonomic level, the IT infrastructure operation is governed by business policies and objectives. Users interact with the autonomic technology to monitor the business processes, alter the objectives, or both [12].

Autonomic computing systems are self-managing systems, and they can regulate and maintain themselves without human intervention. Such systems are able to adapt to changing environments, such as changes in the workload or failures, in a way that preserves given operational goals (e.g., performance goals) [3]. Menasce et al [4][22][23][24] propose a set of techniques to design self-organizing and self-tuning computer systems. The techniques they proposed are based on the combined use of combinatorial search techniques and analytic queuing network models. In their work, Menasce at el defined the cost function to be optimized as a weighted average of the deviation of response time, throughput, and probability of rejection metrics relative to their service level agreements. Bennani at el [3] show that the techniques Menasce at el developed can be employed to solve resource allocation problems in an autonomic data center. The data center hosts several application environments (AEs) and has a fixed number of servers that are dynamically allocated to the various AEs in a way that maximizes a certain utility function. The utility functions described by Walsh et al [39] depend on performance metrics, such as throughput and response time, for different workload intensity levels and different number of servers allocated to an AE. Bennani at el [3] propose the use of predictive multi-class queuing network models to replace the table-driven approach proposed by Walsh et al [39]. The table-driven approach has the limitations of not being scalable with respect to the number of transaction classes in an application environment, not being scalable with respect to

the number of AEs, and not being scalable with respect to the number of resources and resource types. Moreover, building a table from experimental data is time consuming and has to be repeated if resources are replaced within the data center.

Dynamic resource allocation provides the feature for an automatic computing system to self-optimize its resource usage and avoid the inefficiencies of over provisioning of resources. In order to dynamically allocate resources among multiple applications, it is necessary to accurately and automatically compute the value of resources for each application [34]. Tesauro et al [34] studied autonomic resource allocation among multiple applications based on optimizing the sum of utility for each application. In their work, a data center was used, which consisted of multiple, logically separated application environments, each providing a distinct application service using a dedicated, but dynamically allocated, pool of servers. Each environment had its own service-level utility function specifying the utility to the data center from the environment as a function of service metrics. Tesauro et al [34] compared two methodologies, a queuing-theoretic performance model and model-free reinforcement learning, for estimating the utility of resources. They evaluated the two methodologies empirically in the distributed prototype data center and highlighted tradeoffs between the two methods.

2.2 Resource Management in DBMSs

In this section, we discuss previous work from the literature on resource management and allocation in DBMSs for workloads with widely varying resource requirements and performance objectives.

Davison et al. [9] propose a framework for resource allocation in DBMSs based on concepts from microeconomics. The framework aims at reducing the response time of queries and improving resource utilization in a multi-user environment. The central element of the framework

is a resource broker that schedules queries and allocates resources among executing operators to achieve system-wide performance objectives. The guiding principle for brokering resources is profit maximization. By selling limited system resources to competing operators, the resource broker maximizes the profit and the system-wide performance objectives are therefore achieved. In their work, Davison et al. present a prototype broker that manages memory and disk bandwidth for a multi-user query workload.

Boughton et al. [6] present a framework for resource allocation based on the use of economic models. In this framework, a limited number of database buffer pool memory resources are allocated among competing workloads with different SLAs and levels of business importance. A utility function for buffer pool memory resources is introduced to determine the maximum bids that resource consumers are willing to provide for the shares of the resource in order to achieve their SLAs. The framework automatically translates high-level business workload importance policies to the low-level buffer pool tuning actions.

The frameworks of Davison and Boughton are both based on business concepts in order to reduce the complexity of resource allocation problems and to provide stability. In both frameworks, consumers are assigned wealth to reflect their performance objectives and resource brokers are guided by the principle of profit maximization, which in turn maximizes the system-wide performance and achieves workload objectives. Davison et al. use an admission policy to control resource contention by scheduling new queries for execution and use an allocation policy to control a query's bid for resources when the query is scheduled [9]. Boughton et al. use a high-level business importance policy to determine the workloads' wealth and the workloads have equal chances to bid for the shared resources. The amount of wealth assigned to a workload is based on the workload business importance level. Wealthy workloads, therefore, achieve better performance than poor workloads.

Brown et al. [5] propose an algorithm to achieve a set of per-class response time goals for a multi-class workload through automatically adjusting DBMS multiprogramming levels (MPL) and memory allocations. They assume that the system is configured such that it is possible to satisfy the goals for all classes in steady state. Since it is difficult to predict response times as a function of MPL and memory, they avoid exhaustively searching the entire solution space to find optimal $<mpl_c, mem_c>$ pairs for the workloads to meet response time goals by proposing heuristics and a feedback algorithm to search the $<mpl_c, mem_c>$ pairs. In our study, we use a queuing network model [21] to predict the performance of a workload with a certain amount of resources, and apply a greedy algorithm to search for optimal resource pairs in the case of multiple-resource allocation.

Niu et al. [27] propose a framework for workload adaptation in a DBMS. The framework manages multiple-importance classes of queries to meet their performance goals through allocating DBMS resources. Their approach uses an adaptable admission control mechanism to ensure that multiple workloads with different importance levels meet their performance goals. The importance and performance goals of a workload are expressed by a utility function.

Schroeder et al. [32] present a similar framework as Niu et al. to meet a set of online transaction processing (OLTP) workload quality of service (QoS) targets. The framework has two main components, the scheduler and the MPL advisor. The core idea of the framework is to maintain an upper limit on the number of transactions executing simultaneously within the DBMS. In their framework, Schroeder et al. divide transactions into different classes based on their business importance, and obtain QoS targets and overall performance of the DBMS through choosing MPL levels.

The approaches of both Niu and Schroeder perform workload management in DBMSs based on the business importance policies and schedule queries outside the DBMS. They do not directly deal with resource allocations, and therefore, they do not require low-level resource management plans. To partition individual DBMS resources and make system-level resource allocation plans for competing workloads, more sophisticated resource allocation techniques are necessary.

2.3 Economic Models for Resource Management

Applying economic models to allocate resources in computer systems and networks has been studied for many years as economic models can provide interesting contributions to resource sharing algorithms. Economic models potentially reduce the resource allocation complexity by decentralizing the control of resources and partition large complex allocation problems into smaller and disjoint allocation problems.

In an economy, resource decentralization is generated by the economic model agents. Generally, there are two types of agents, suppliers and consumers, and they selfishly attempt to achieve their goals. A consumer attempts to optimize its individual performance by obtaining the resources it requires, and a supplier attempts to optimize its individual profit by allocating its individual resources to consumers. Most economic models introduce currency and pricing as the technique to coordinating the behavior of agents [10].

An economy contains a set of resources. An agent may prefer some resource allocations over others, and the preferences are typically represented by utility functions. In economic models, there are two main ways to allocate resources among the competing agents. One of them is the exchange based economy and the other is the price based economy. In the exchange based economy, each agent is initially endowed with some amount of the resources. They exchange

resources until the marginal rate of substitution of the resources is the same for all the agents. In a price based system, the resources are based on the demand, supply, and the wealth in the economic system. Each agent computes the demand from the utility function and the budget constraint. Bidding and auctioning resources is a form of resource allocation based on prices [10]. In this study, we use the price based economy to coordinate the agents' behaviors in an economic model since the utility functions required in the price based economic models are relatively easy to be defined and implemented.

There are many studies on using economic models in resource allocations. Stratford et al. [33] present an architecture based on an economic model of resource management to achieve the global (system) and local (application) performance optimizations on a computer system. In their approach, system applications are responsible for their own local optimization through negotiating resource allocations with resource managers. Stratford et al. present that by correctly setting up the local optimization problems, it should be possible to achieve global optimization as a side-effect of the local optimizations. The basic mechanism of their approach is the resource pricing. Resource managers attempt to maximize their revenues, which are generated by selling resource contracts to applications, and applications attempt to maximize their utility by purchasing and trading resource contracts. Applications are provided with credits from a user agent, renewable over a given time-scale. User agents are responsible for implementing the policy of a particular user of the system. In the architecture, Stratford et al. use dynamic pricing as a congestion feedback mechanism to enable applications to make system policy controlled adaptation decisions.

Stonebraker et al. [31] propose an economic paradigm to solve the complex query execution and storage management issues in the Mariposa database system at Berkeley. There are three kinds of entities in the economic system, namely clients, brokers, and servers. Clients

represent queries submitted by user applications at a client site. A query starts with a budget which pays for executing the query. The broker's job is to get the query performed. The goal of a broker is to expend the client's budget to balance resource usage with query response time. Mariposa objects can be stored over a number of autonomous sites, thus each server site responds to queries issued by a broker for data or metadata. A server site attempts to maximize income by buying and selling storage objects, and processing queries for locally stored objects. In the Mariposa database system, brokers manage the bidding and query execution on behalf of clients, and clients direct brokers using budgets. Mariposa uses an economic bidding process to regulate name service, storage management, and query execution.

Kurose et al. [19] consider the problem of optimal file (data objects) allocation in a complex distributed computer system. In the approach, they present decentralized algorithms to allocate file resources among agents (computer systems). The algorithms are based on the economic models in a cooperative and non-competitive manner [10]. The file access optimization is to minimize the overall communication cost required to satisfy file accesses and to minimize the average time delay associated with file access. They define a utility function to reflect the utility of the file allocation in the distributed system. In their approach, a global utility function is used for all the users (N nodes). Each user is given an initial allocation, and they then compute the optimal allocation by solving the global optimization problem by utilizing the utility function. The process is done by trading appropriate amounts of resources iteratively (marginal rate of substitution of resources) among the users themselves until they reach a point where the marginal rate of substitution of resources is equal, and the allocation at this point is optimal.

Defining a utility function is a key procedure in designing an economic based computer system as the utility function is the fundamental mechanism in the resource allocation processes.

In our study, we define utility functions in our resource allocation framework based on our proposed performance models.

Chapter 3
Resource Allocation Framework

As described earlier, resource allocation in an autonomic DBMS is a process of automatically optimizing resource assignment among competing workloads to achieve the workloads' performance objectives and economic concepts can be used in the resource allocation process in DBMSs in order to map a high-level business policy to system-level resource tuning actions to potentially reduce the operational complexity. In previous work, our research group proposed a resource allocation framework that uses an economic model to implement resource allocation in an autonomic DBMS [7]. The work focused on managing a single resource, namely buffer pool memory space. The experimental results indicated that the resource allocation framework could successfully make buffer pool memory allocation plans for competing database workloads to meet their performance requirements.

To further investigate this resource allocation approach in DBMSs, in this research, we first study the application of system CPU resource allocations and then extend the single-resource allocation study to simultaneously allocate multiple resources, namely buffer pool memory space and system CPU shares, in an autonomic DBMS. The economic model, as shown in Figure 2, consists of a set of shared resources, resource brokers, a trade mechanism, and resource consumers. It represents a price based economy with a market mechanism of auctioning and bidding for the shared resources.

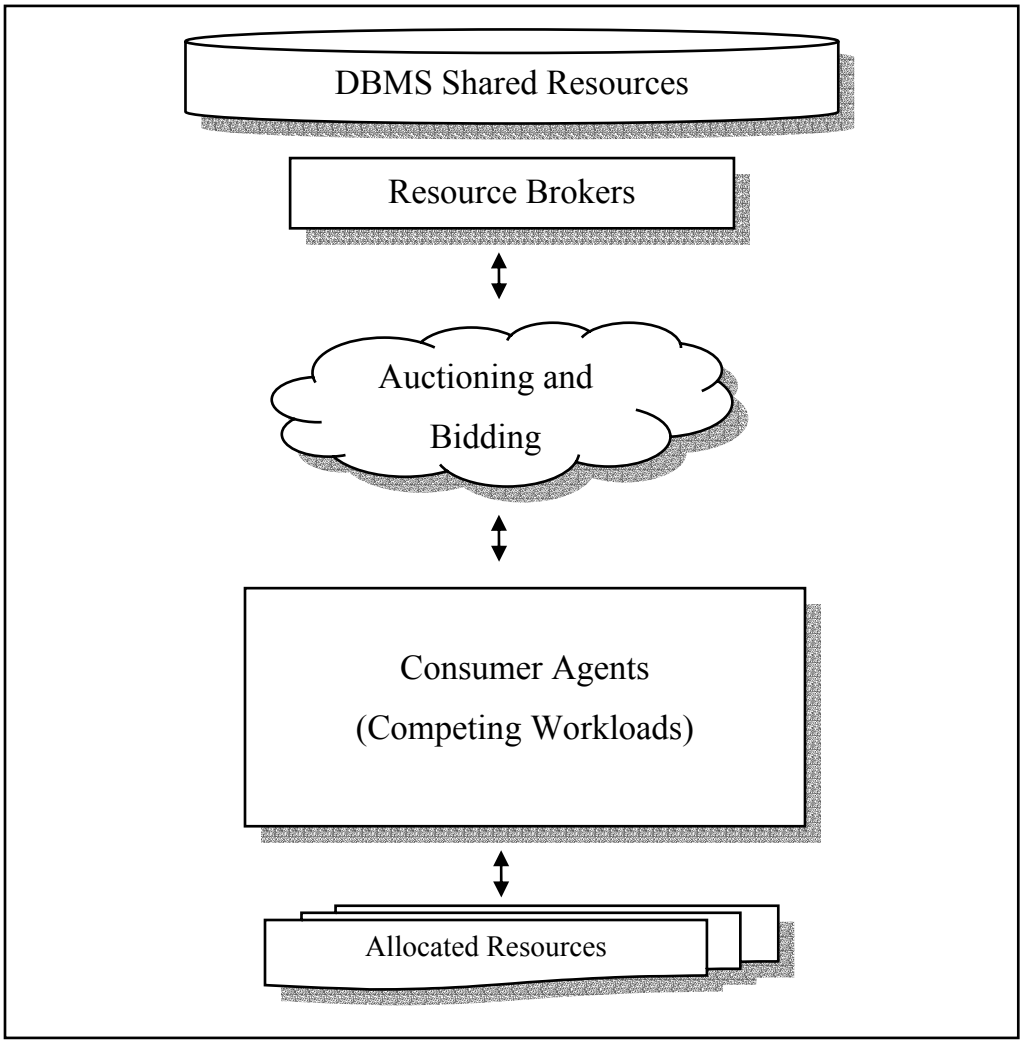

Figure 2: Economic Model Framework [7]

3.1 Shared Resources

Theoretically, an economic model can manage an arbitrary number of shared resources for allocation. We study the allocations of two DBMS shared resources, namely buffer pool memory space and system CPU shares. We choose these two resources since they are key factors in database performance management.

The buffer pool of a database system is an area of main memory in which data and index pages are cached to reduce disk access. The purpose of the buffer pool is to improve performance of a database system by attempting to reduce the number of physical reads from its I/O system. For many workloads, a larger buffer pool size means that a database can achieve better performance, and a smaller buffer pool size, on the other hand, means poorer system performance. System CPU share is another main resource that contributes significantly to database system performance. A request in a database system needs a certain amount of CPU service to complete its work. If a database system obtains high CPU utilization, then the database can process more requests at a time and therefore achieves better performance.

In this study, we consider the case where a DBMS concurrently runs several workloads which are classified into multiple business-level importance classes and have unique performance requirements. The workloads compete for these two shared resources and the economic model is applied to make the resource allocation plans and determine appropriate allocations.

3.2 Resource Brokers and the Trade Mechanism

Resource brokers allocate resources to consumer agents that represent the workloads via an auctioning and bidding based economy. Each broker owns a type of resource and charges consumer agents for the use of the resources. Consumer agents are assigned some amount of wealth to bid for the shared resources. The amount of wealth reflects a workload's business importance.

The principle behind the auctioning and bidding mechanism is that the resource brokers and consumer agents selfishly attempt to achieve their goals [10]. Resource brokers attempt to maximize their profits by conducting auctions to sell the resources they own and take the highest bids submitted by consumer agents. Consumer agents, on the other hand, attempt to maximize

their workloads' performance by submitting the maximum bids they can afford to win auctions and therefore gather more resources. With resource brokers maximizing the profits, the optimal resource access among competing workloads and the SLAs can be maximally achieved.

Several types of auctions, such as English auction, Dutch auction, Hybrid auction, and Sealed-Bid auction, can be used in the auctioning and bidding based economy. An English auction is where the price of the resource gradually increases with the bidding. The highest bidder obtains the resource. A Dutch auction means prices are gradually lowered by the seller until one of the buyers claims the resource. In a Hybrid auction, the asking price of a resource is increased if a bid is submitted, and decreased if no bid is submitted. In a Sealed-Bid auction, sealed bids are submitted by the agents and the highest bid is given access to the resource [10].

A Sealed-Bid auction is chosen as the trade mechanism in our economic model since it is more efficient and easily implemented than other auctions. In Sealed-Bid auctions, consumer agents submit bids to resource brokers and are not aware of the amounts bid by others. Resource brokers collect the sealed bids submitted by consumer agents, select the highest bid as the winner and allocate the resources to the consumer agent. To maximize profits, resource brokers conduct auctions until there are no remaining resources, or until there are no further bids, indicating that the consumer agents have depleted their wealth, or their performance requirements have been satisfied using the current allocation. The price of a resource share in an auction is set by the highest bidder among the consumer agents. Resource prices, therefore, might vary through the process of all the auctions.

3.3 Consumer Agents

A consumer agent represents a workload running on the DBMS and competes for the shared resources with other consumer agents. Each consumer agent executes queries in its workload and

strives to meet the workload performance requirements. We consider several competing workloads as described earlier, thus there are the same number of consumer agents and workloads in the economic model. A consumer agent typically consists of a workload, a certain amount of wealth, and a resource utility function. The interactions between the consumer agents and the resource brokers are through the Sealed-Bid auctions.

3.3.1 Amount of Wealth

The workload of a consumer agent is conceptually divided into a series of *resource allocation intervals* each with approximately an equal number of requests [6]. By using the resource allocation intervals, an economic model could easily adapt to competing workloads' dynamic resource demands, providing more opportunity for competitions for appropriate resource allocations. At the beginning of each interval, all the resources are returned to resource brokers, and the consumer agent is assigned some amount of wealth. This initial amount of wealth is determined by two factors, namely the workload business importance level, and the cost of running the workload for the resource allocation interval. In our case, we use the total number of I/O operations of the workload queries in the current resource allocation interval to represent the cost. If a consumer agent represents a workload W with K queries in a resource allocation interval, then the wealth assigned to the consumer agent at the beginning of the resource allocation interval is calculated using equation (3.1):

$$Wealth(W) = imp_W * \sum_{i=1}^{K} cost(q_i) \tag{3.1}$$

where, $cost(q_i)$ is the estimated cost of the *i-th* query of the workload W in the current resource allocation interval, and imp_w is the workload business importance level called the *importance multiplier* which is defined in Section 3.4. We use batch workloads in our studies, where all

queries are known in advance, so we are able to estimate each workload's query I/O operations using the DB2 Explain utility [15].

Workload business importance levels can significantly affect the amount of wealth of consumer agents if the costs of the competing workloads are similar. A consumer agent might have much more wealth than others if the workload that it represents has the highest business importance, while a consumer agent might have much less wealth if its workload has less business importance. A wealthy consumer agent is more likely to win resource auctions, and therefore, its workload will be allocated sufficient resources to achieve high performance. The less important workloads, associated with less wealthy consumer agents, on the other hand, might experience low performance in the resource constrained situation. In a resource allocation interval, if a consumer agent wins an auction and gains the resource shares, then it will lose wealth, and therefore it will be less likely to win resource shares on the next bid. This ensures that all consumer agents can have a chance to win auctions [6].

3.3.2 Utility Function

The utility function [39] of a consumer agent, in the economic model, is a monotonically non-decreasing function, and it maps performance achieved by a workload with certain amount of allocated resources into a real number u [29]. There is no single way to define a utility function in the model. The guidelines for choosing or constructing a utility function are the following [29][26]:

- The value of u should increase as the performance experienced by a workload increases, and decreases otherwise.

- The amount of utility increase should be getting larger as the performance of a workload increases quickly, while, the amount of utility increase should be getting smaller as the performance of a workload increases slowly.
- The return value, *u*, of a utility function should be determined by the amount of allocated resources of a workload, and the value of $u \in [0,\cdots,1]$.

If the performance of a workload is high, then the utility of resources allocated to the workload is close to 1, while, if performance of the workload is low, then the utility of resources allocated to the workload is close to 0. The resource utility function helps consumer agents to determine the workloads' resource preferences and their maximum bidding values.

3.3.3 Maximum Bid

The maximum bidding value of a consumer agent is determined by its wealth and marginal utility, that is, the difference in utility between two consecutive resource allocations [6]. If an economic model allocates *M* shared resources, and a consumer agent in the model, representing the workload *W*, has the allocated resources $\vec{x} = \{x_1, x_2, \cdots, x_m\}$, and it is to bid on $\vec{y} = \{y_1, y_2, \cdots, y_m\}$ additional resources, then the marginal utility can be calculated with equation (3.2) [6]:

$$Mgl(W) = U((x_1 + y_1), (x_2 + y_2), \cdots, (x_m + y_m)) - U(x_1, x_2, \cdots, x_m) \quad (3.2)$$

where, $U(x_1, x_2, \cdots, x_m)$ is the resource utility function that is defined in the cases of single-resource and multiple-resource allocations respectively. The maximum bid of the consumer agent for the additional resources, $\vec{y} = \{y_1, y_2, \cdots, y_m\}$, is determined by:

$$Bid(W) = Mgl(W) * Wealth(W) \tag{3.3}$$

where, $Mgl(W)$ can be calculated with equation (3.2), and $Wealth(W)$ is determined by equation (3.1).

As the utility function is non-decreasing, and its return value is in the real number range $[0,\cdots,1]$, based on the equation (3.2), the value of a marginal utility is also in range $[0,\cdots,1]$. The marginal utility reflects potential performance improvement of a workload. For some resources, if the calculated marginal utility of a consumer agent is close to 1, then it means these additional resources can significantly benefit the consumer agent's workload performance, while, if the calculated marginal utility is close to 0, then the additional resources will not help the workload's performance. By examining the marginal utility value, a consumer agent can determine the preferred resources for its workload. The maximum bid, as shown in equation (3.3), is the marginal utility multiplied by current wealth of a consumer agent, and says that a consumer agent is willing to spend the marginal-utility percentage of its current wealth as a sealed-bid to purchase the resources.

3.4 Workload Importance and Importance Multiplier

Workloads running concurrently on a database server are diverse and complex. They may have unique performance requirements and dynamic resource demands. As an example, a transactional OLTP workload may have a high throughput requirement, while a decision support system (DSS) workload may have a short response time goal. By applying a business importance policy, these workloads can be classified into different important classes.

3.4.1 Workload Importance

Normally, there are two interpretations to express that one workload is more important than another, namely absolute importance and relative importance. Absolute importance means that

competing workloads are assigned priorities according to their business importance levels [1][32]. The resource requirements of high priority workloads should be satisfied before others, thus an important workload may be allocated all resources and less important workloads must wait for the resources to be released. Relative importance is that all workloads simultaneously share the limited system resources, and the amount of resources allocated to one workload is commensurate with the workload's importance relative to others [26].

3.4.2 Importance Multiplier

We use the concept of an *importance multiplier* [7] to capture the differences in the relative importance of competing workloads. For example, we may wish to classify multiple workloads into three business importance classes, which can be labeled as *high importance*, *normal importance*, and *best effort*. We may assign a value of 1 to the importance multiplier of the best effort class and then express the degree of importance of the remaining classes relative to that value. For instance, if the high importance class has its importance multiplier as 10, then it is 10 times more important than the best effort class. If the normal importance class has its importance multiplier as 5 then the class is 5 times more important than the best effort class, and in turn, the high importance class is 2 times more important than the normal importance class.

As described in equation (3.1), a workload importance multiplier can significantly affect the amount of wealth that is assigned to a consumer agent, and the wealth, then, determines the amount of resources that could be assigned to the consumer agent. By using the importance multipliers, an economic model enforces competing workloads' degree of importance in a resource allocation process. In our approach, the domain of the importance multipliers is positive integer, thus the amount of wealth assigned to a consumer agent can be very large. However, since we use the relative importance to describe a workload business importance level and assign

the value of 1 to the importance multiplier of the best effort class, and the business importance levels of competing workloads are also defined in a given business importance policy, as a result, the amount of wealth assigned to a consumer agent is reasonable.

3.5 Summary

In this chapter, we present a general framework of an economic model for resource allocation in a DBMS. It consists of four components, namely a set of shared resources, resource brokers, a trade mechanism, and consumer agents. Resource brokers conduct auctions to sell the shared resources, and consumer agents submit bids to buy the shared resources from the resource brokers via an auctioning and biding based trade mechanism.

In the economic model, a consumer agent represents a workload to bid for the shared resources in resource auctions. A consumer agent typically consists of a workload, a certain amount of wealth, and a utility function. The wealth of a consumer agent is determined by its workload cost and the workload business importance level. Based on the utility function, a consumer agent can calculate the marginal utility of the resource shares for bidding. The bidding value of a consumer agent is determined by the marginal utility and the amount of wealth that the consumer agent currently has. The wealthy consumer agents are able to win more resource auctions than the poor agents, and, therefore, the workloads of wealthy consumer agents can achieve better performance than the workloads of poor ones.

In Chapter 4, we illustrate the use of the economic model to allocate system CPU resources to workloads in multiple importance classes, and in Chapter 5, we extend our resource allocation framework using the economic model to allocate multiple resources in a DBMS.

Chapter 4
Allocating CPU Resources

The economic model, as described in Chapter 3, directly deals with resource allocations for competing database workloads based on their business-level importance policies. One application of using the model to allocate resources is buffer pool memory space management in an autonomic DBMS [7]. To manage a DBMS resource using the economic model, the resource modeling methods and the resource utility function must first be defined in the model as they are key mechanisms in the resource allocation process. Resource modeling provides methods to partition the shared resource among competing workloads and to determine the reasonable total amount of the resource for allocation. In our work, we determine a reasonable total amount of a resource by selecting the minimum amount of the resource where maximum system performance can be achieved. A utility function maps usefulness of resources to performance of a workload achieved, so it helps consumer agents to determine their maximum bidding values with respect to the workload potential performance improvement and the workload business importance levels. Different DBMS resources could have different modeling methods and utility functions since the resource characteristics are different.

In this chapter, we demonstrate the use of the economic model to allocate system CPU resources to competing database workloads in order to achieve the workload performance objectives, where the competing workloads are classified into different business importance classes and have unique performance objectives. The economic model for CPU resource allocation shown in Figure 3 consists of system CPU shares, a CPU resource broker, and consumer agents. The consumer agents represent competing workloads running on a DBMS,

which are in different business importance classes, namely high importance, normal importance and best effort, as described in Section 3.4. The CPU broker conducts auctions to sell the CPU resources, and the consumer agents bid for the resource shares. The total amount of CPU resources for allocation in the economic model is determined by the CPU resource modeling method and the CPU resource utility function is defined in the consumer agents.

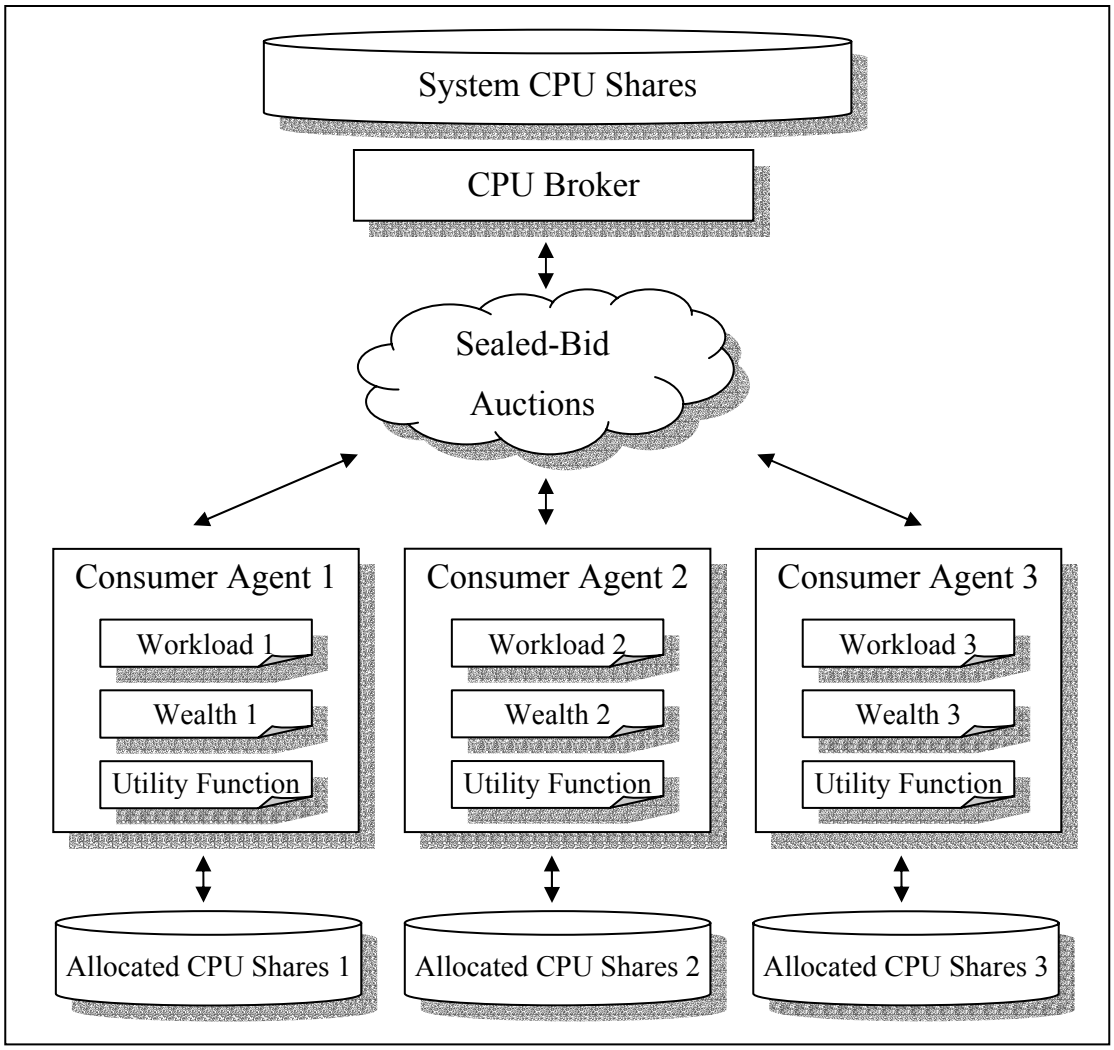

Figure 3: Economic Model for CPU Resource Allocation

The validation of our approach is presented at the end of this chapter. We extend the economic model simulator created by Boughton [7] to simulate CPU resource allocation processes and run the database workloads with the allocated CPU resources on a DB2 DBMS to verify that their performances match a given business importance policy.

4.1 CPU Resource Model

As it is not sufficient to simply assign different OS priorities to competing workloads based on their business importance, as described in Section 3.4, we use the relative importance to capture differences among competing workloads in our resource allocation framework. In a resource allocation process, we assume shared DBMS resources could be partitioned, and the resource shares could be directly assigned to individual competing workloads, and hence the database workloads use the assigned resource shares to achieve their performance objectives at the same time.

4.1.1 CPU Resource Partition

Given the limitations of current operating systems, CPU cycles cannot be directly assigned to a workload. Instead, we partition system CPU resources through controlling the number of database agents that are available to service requests on the database server. Database agents, in terms of IBM's definition in DB2 databases, can be simply described as processes, or threads, of a database manager. They facilitate communication between a database manager and clients and local applications, and they do the work that clients and local applications request in the database manager instance. In UNIX® environments, database agents run as processes, while, in Intel-based operating systems such as Windows®, the agents run as threads [14][15].

In a DB2 database server, the DBMS can be configured such that one client has one active database agent to coordinate its work, and therefore the number of database agents

available determines the number of requests of a database workload that can be admitted into the system at a time. A greater number of database agents result in more requests being processed at a time and higher system CPU utilization achieved.

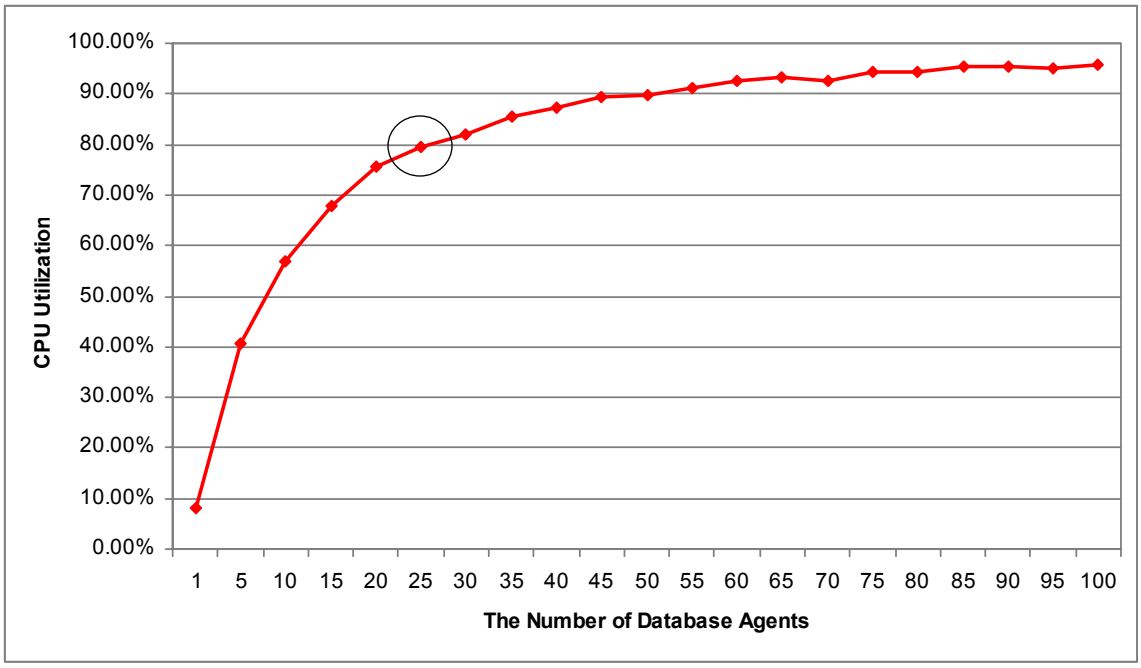

Figure 4: CPU Utilization versus Number of Database Agents

We assume that the more agents allocated to service requests for a particular database workload, the more CPU time this workload receives. We conducted experiments to verify the relationship between the number of database agents and system CPU utilization for a workload. As shown in Figure 4, the more database agents allocated to serve requests for a particular workload, the more system CPU shares this workload receives. To ensure that the CPU utilization responds to increases in the number of database agents used in the system, we configured the database server such that CPU was the bottleneck resource. Each data point present in Figure 4 is the average over 5 runs with a 90% confidence level and ±10% confidence intervals [25].

4.1.2 Total Amount of CPU Resources

Based on Figure 4, we notice that the system CPU utilization increases with the increase in the number of database agents used in the system. This trend is basically linear until we reach 25 database agents, at which the performance begins to level off, the system becomes saturated, and the database agents begin to interfere with each other. The increase in the CPU utilization after this point is more likely due to the additional CPU cycles required to maintain the large number of database agents running in the system.

Based on our experimental results, we chose 25 database agents as the maximum number available in the resource allocation framework for allocation in our experiments. By using 25 database agents, a workload can receive a significant improvement in its CPU utilization when it is assigned one additional database agent, and the database system is able to obtain the highest CPU utilization. The total amount of system CPU resources for allocation is set as a parameter in the economic model, so our approach can adapt to different database sever configurations.

4.2 CPU Resource Utility Function

We define the CPU resource utility function based on a performance model. The model estimates system performance of a DBMS with concurrently running workloads. In our experiments, several OLTP workloads are used to compete for the CPU resources on a DB2 DBMS. System throughput is used to measure the DBMS performance, where throughput is the number of transactions completed per unit time [38], and a transaction is one execution of a client application in the DBMS [30].

4.2.1 System Throughput Estimation

Queueing Network Models (QNMs) are analytical performance models that can be used to predict the throughput, utilization, and latency of a computer system and its components (see

Appendix A for details). QNMs represent a computer system as a set of service centers that process customers (also known as transactions, jobs, or arrivals). Service centers can be any active resource such as a CPU, disk or network link [21][38].

We use QNMs to model the DB2 DBMS used in the experiments and to predict the system throughput when running OLTP workloads under different configurations. As we focus on system CPU resource management in this study, our DB2 QNM is simply a single service center representing the system CPU resources. The throughput of the DBMS can be therefore estimated by applying the utilization law [21]:

$$Throughput = Utilization \,/\, Demand \tag{4.1}$$

where *Utilization* is the percentage of time that the CPU is busy serving requests in the database system, and *Demand* is the average CPU service time of transactions in the workloads.

4.2.2 DB2 QNM Parameterization

To parameterize the DB2 QNM with respect to specific workloads, we must determine the average service demand of transactions in the workloads from the CPU service center, and define the relationship between CPU utilization and the number of database agents used in the system. In our experiments, we use the OLTP workloads from the TPC-C benchmark [36] (see Appendix B for details). For a specific database system and a specific workload, the average CPU service demand of a transaction is constant and is independent of the number of transactions concurrently existing in the system. By monitoring the CPU usage of transactions with varying numbers of database agents used in the system, we experimentally obtained CPU service demand values with using the equation (4.2).

$$Demand_{CPU} = (U * T) / C \tag{4.2}$$

where U is the CPU utilization, T is the total workload execution time, and C is the number of completed transactions in the experiment. For TPC-C workload transactions with our specific experimental environments described in Section 4.3, the calculated average CPU service demand is 0.0181 (seconds/transaction). Figure 5 shows the calculated average CPU service demand value as compared to actual obtained values. All numbers represent averages over 5 runs, and a 90% confidence level with ±10% confidence intervals [25] is used for each data point.

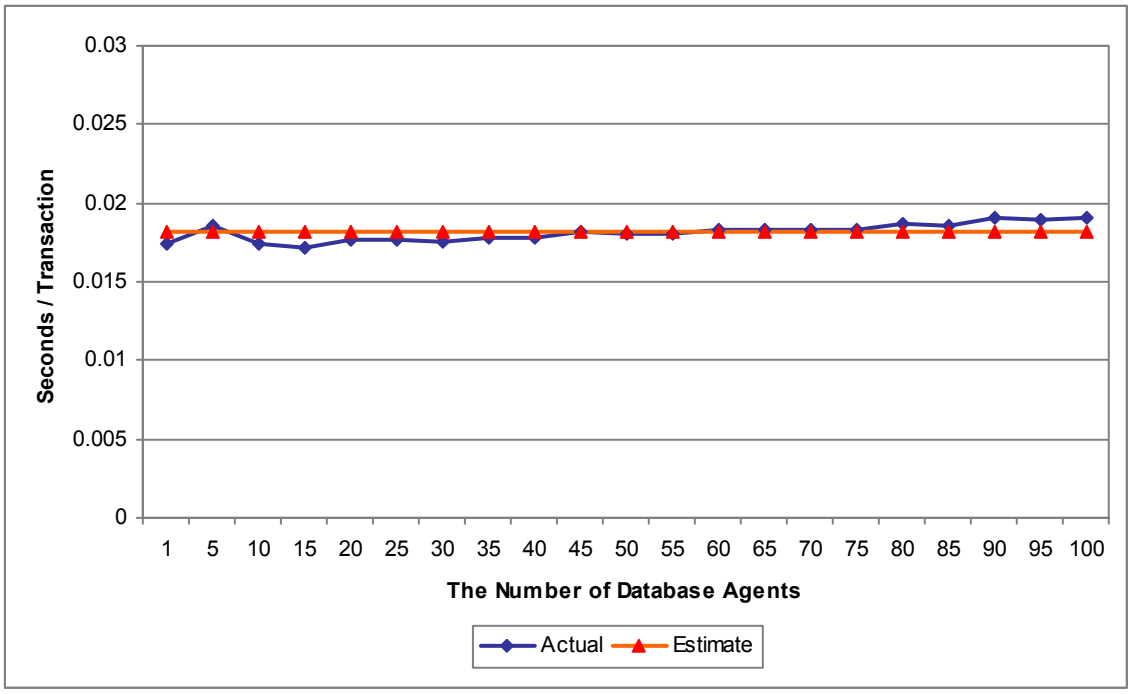

Figure 5: Validation of CPU Service Demand

To define the relationship between CPU utilization and the number of database agents used in the system, we apply linear regression [25] on the line up to 25 database agents, as shown in Figure 4, to express the increase in CPU utilization and the number of database agents used in the system. The derived function is:

$$Utilization_{CPU}(N) = (2.744 * N + 19.962)/100 \quad (4.3)$$

where N is the number of database agents, $N \in [1, \ldots, 25]$. For $N > 25$, the CPU is fully utilized and reaches its maximum utilization. With the DB2 QNM parameterized, the system throughput can be estimated by using equation (4.1) for a standard OLTP workload running on the DB2 DBMS given a fixed amount of system CPU resources.

Figure 6 shows values obtained by equation (4.1) compared to actual throughput values. Since linear regression is applied to predict CPU utilization defined in equation (4.3), there is a discrepancy between the estimate and actual throughput values shown in Figure 6. Instead of accurately estimating particular throughput values, what we need in this study is to accurately predict system performance trend with the changes in CPU resources. Therefore, the validation is acceptable, and the performance model can be used in the following experiments.

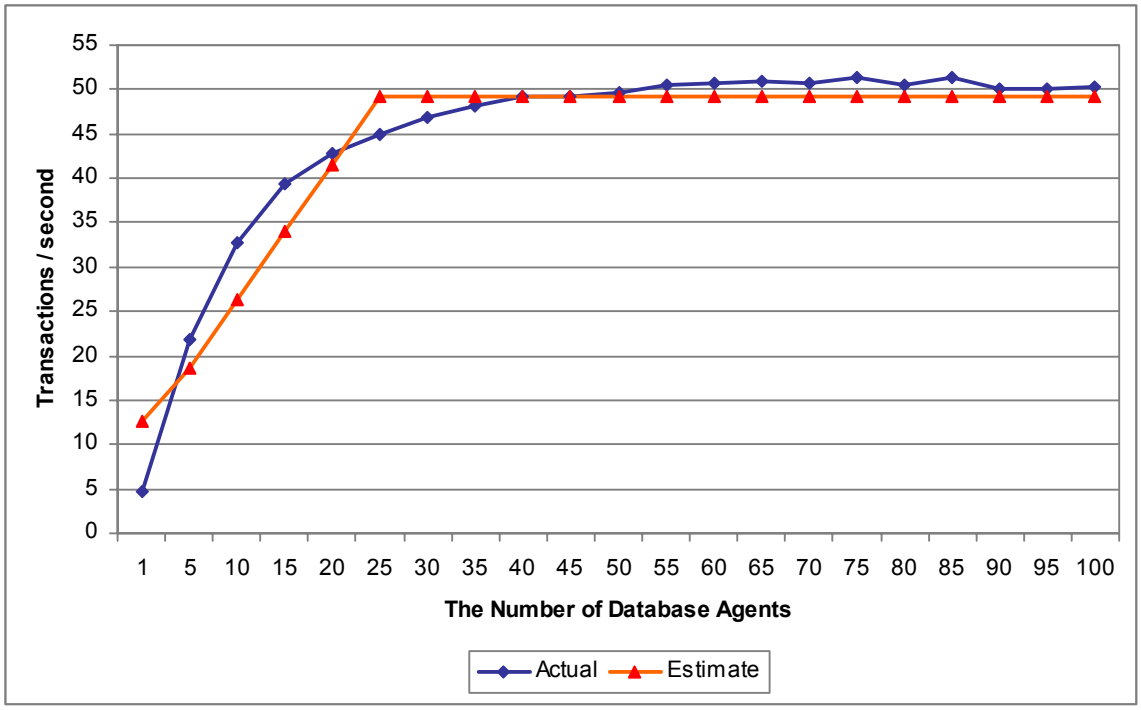

Figure 6: Validation of DB2 QNM

4.2.3 Normalized Throughput

Following the guidelines of defining utility functions described in Section 3.3.2, we define the CPU resource utility function by normalizing the estimated throughput with respect to the maximum throughput that the workload could achieve when all CPU resources are allocated to it. The utility function is given by:

$$Utility_{CPU}(N) = (U_{CPU}(N)/S_{CPU}) / X_{Max} \tag{4.4}$$

where $U_{CPU}(N)$ is the CPU utilization function defined by equation (4.3), S_{CPU} is the average CPU service demand of transactions in the workloads as shown in Section 4.2.2 and X_{Max} is the maximum throughput achieved by the workload when assigned all CPU resources. A graph of this utility function versus the number of database agents is shown in Figure 7.

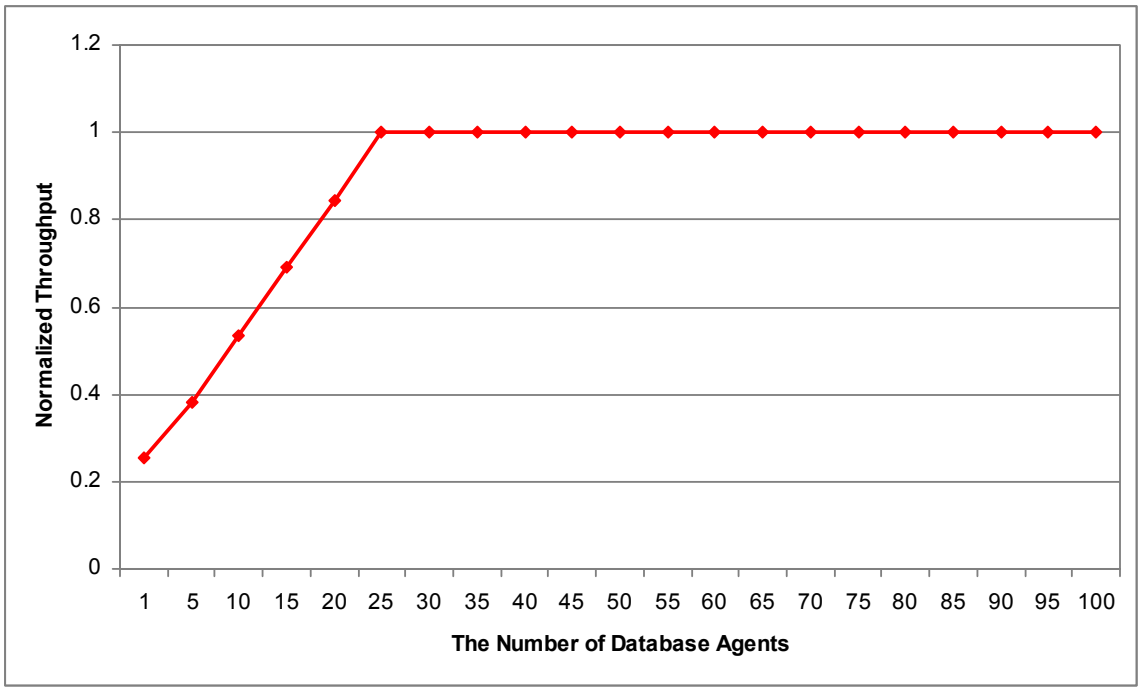

Figure 7: Utility Function for System CPU Resources

As described earlier, the CPU is the bottleneck resource in our database server, so the DBMS is saturated when the CPU is fully utilized. At this point, the system throughput remains constant even when more database agents are allocated. Since all of our experimental workloads are of the same type, the same utility function can be applied for all the workloads.

4.3 Experimental Validation

We extend the simulator of the economic model created by Boughton [7] to provide an experimental environment in which to validate the approaches proposed in the resource allocation framework for the CPU resource allocation problem. The validation of the approach addresses two main issues:

- Does the resource allocation framework generate the CPU resource allocations for competing workloads on a DBMS so that the resource allocations match the database workload business importance policies?
- Are the allocations resulting from the auctions sufficient for the competing workloads to achieve performances that match the workloads' business importance levels?

4.3.1 Experimental Environment

We apply the economic model to a single DB2 instance with an identical database for each of the competing workloads from different importance classes. Each database is configured with two buffer pools where one buffer pool holds data pages and the other buffer pool holds index pages. The buffer pools sizes are experimentally determined to be 168 MB each in order to ensure that the CPU is the bottleneck resource and determines the database throughput. Each database has one workload running on it, thus we can control the CPU shares that each workload receives by controlling the number of agents assigned to the database. We use three standard OLTP workloads in our experiments, so the economic model allocates system CPU resources across

three identical databases based on the given workload business importance policies. We use one database agent as the allocation unit of CPU resource in this study as this granularity gives a reasonable workload performance increment and makes the resource allocation efficient.

All experiments are conducted with DB2 Universal Database Version 8.2 [15] running on an IBM xSeries® 240 PC server with the Windows XP operating system. The database server is equipped with two 1 GHz Pentium 3 processors, 2 GB of RAM and an array of 11 disks. To ensure that the CPU is the throughput-limiting resource, and to eliminate the need to account for CPU parallelism in the DB2 QNM introduced in Section 4.2.1, we configure the system with one processor available. We use the Windows XP operating system's performance monitor to collect CPU utilization data.

The database and workloads are taken from the OLTP TPC-C benchmark [36]. The size of each database is 10GB. The three workloads are TPCC-like OLTP batch workloads stored in three script files which are the same files used for buffer pool memory space management studies [7]. The workloads send transactions to the DBMS with zero think time during their run. Each workload consists of 120,000 requests based on the 5 different types of the transactions of TPC-C benchmark [36]. A workload is divided into 12 resource allocation intervals. The start of each interval provides a checkpoint at which a resource allocation process takes place to reallocate resources to workloads based on their current importance levels and wealth.

The simulator is written in Java™ (see Appendix C for UML class diagram) and the workload script files are used as the simulator input. The output of the simulator is the plan of CPU resource allocations, that is, a list of the number of database agents for the workloads at each of their checkpoints. According to the output, the number of database agent configuration commands is inserted into the workload script files at the checkpoints, allowing for a database to

dynamically reconfigure during a run. Before each run of the workloads, the databases are restored to their initial states. The normalized throughput, as defined in Section 4.2.3, is taken to measure the workloads' performance. Each experiment was run five times, and the average of the five runs is reported.

4.3.2 CPU Resource Allocations

A set of experiments were conducted to determine whether our approach generates CPU resource allocations which match a given workload business importance policy. The workloads are assigned one of three different importance classes, namely the high importance class, the normal importance class, and the best effort class, as described earlier. We experimented with three different sets of importance multipliers to validate the economic model. An importance multiplier set represents the importance multiplier values of the three importance classes in order of {best effort, normal importance, high importance}. The three sets used were {1, 1, 1}, {1, 5, 6}, and {1, 5, 10}. These configurations were designed to examine how CPU resource allocation changes with the importance multipliers.

Figure 8 shows the database agent allocations produced by the economic model using the three workload business importance multiplier sets. The workload importance multiplier set {1, 1, 1} represents the case where the three competing workloads are from three different business importance classes of equal importance. In this case, the three workloads are allocated approximately the same amount of CPU resources as shown in Figures 8. Using the importance multiplier set {1, 5, 6}, the high importance and the normal importance classes are much more important than best effort class, and the high importance class is also slightly more important than the normal importance class. When the economic model is used to allocate resources in this case, the high importance and normal importance workloads are allocated significantly more resources

than the best effort workload, while the high importance workload is allocated slightly more resources than the normal importance workload. The set {1, 5, 10} represents the case where the high importance class is much more important than the normal importance class, and the normal importance class is much more important than the best effort class. In this case, the high importance workload is allocated more resources than the normal important workload, and the normal importance workload wins significantly more resources than the best effort workload.

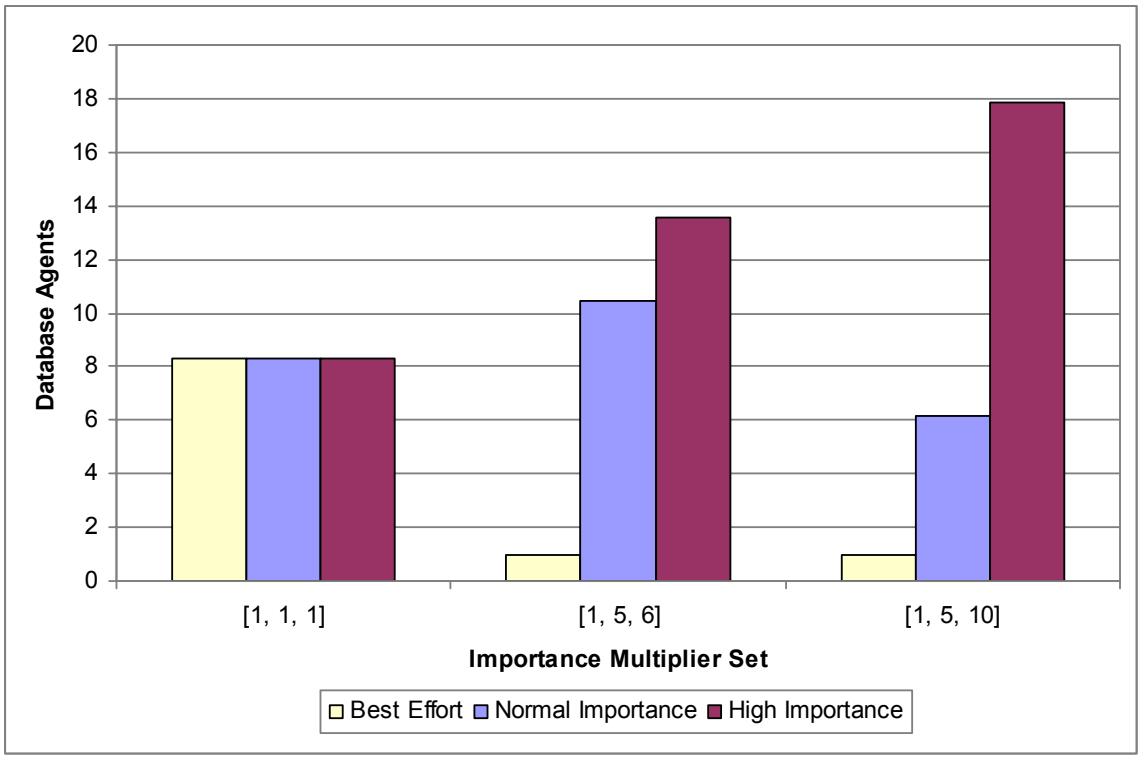

Figure 8: CPU Resource Allocations for Different Importance Multiplier Sets

We conclude that CPU resource allocations generated by the economic model for competing workloads match the workload business importance policies with more important workloads winning more CPU resources than less important workloads.

4.3.3 Workload Performance

A set of experiments were conducted to determine whether the performance achieved by the workloads with resource allocations generated by the economic model match the business importance policy. The workload importance multiplier sets used in these experiments are the same as the sets used in the experiments shown in Section 4.3.2. We can therefore observe the achieved performance of the workloads with the allocated resources obtained in the first set of experiments. The workloads were run on DB2 using the resource allocations suggested by the simulator. The workloads' normalized throughputs are shown in Figure 9.

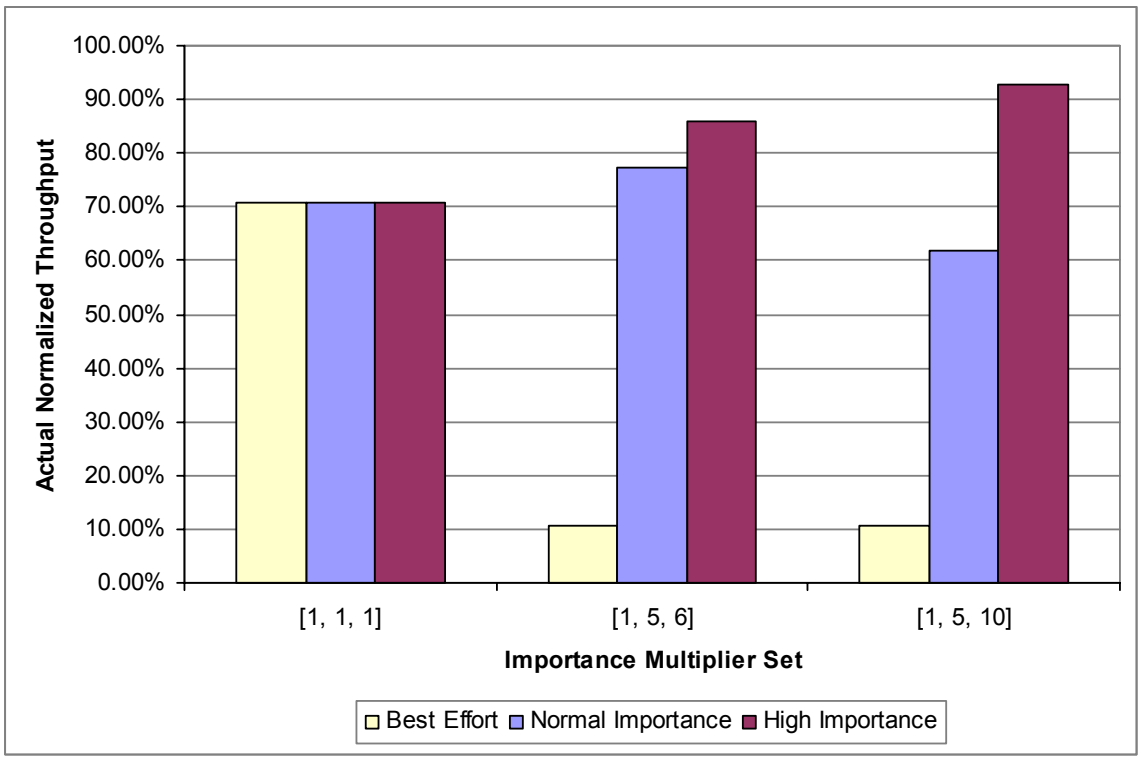

Figure 9: Normalized Throughput for Different Importance Multiplier Sets

In Figure 9 we observe that the throughput of the workloads match the given business importance policies. When the workloads had the same business importance, namely applying importance multiplier set {1, 1, 1} on the economic model, the workloads achieve a similar normalized throughput. When the workloads are of different business importance, namely applying multiplier sets {1, 5, 6} and {1, 5, 10} on the model respectively, then the important workloads achieve higher performance than the less important workloads. The trend shows that the more important a workload is, the more CPU resources it receives and hence, the workload performance is greater.

Based on the results, we conclude that using the economic model for CPU resource allocation is successful in obtaining workload performance that matches the given workload business importance policies.

4.4 Summary

In this chapter, we demonstrate an approach that uses an economic model to allocate system CPU resources for competing workloads on a DBMS where the database workloads are from multiple business importance classes with unique performance objectives. We propose a CPU resource modeling method, through controlling the number of database agents used in a DBMS, to partition system CPU resources. By using the CPU resource modeling method, we experimentally determine the reasonable total amount of CPU resources for allocations in the economic model. In order to define the CPU resource utility function, we apply a QNM on our DB2 DBMS to predict the system performance. The DB2 QNM is built as a simple application of QNMs with single CPU service center, and the utilization law is applied on the DB2 QNM to estimate the system performance for a workload running on it with certain amount of allocated CPU resources.

We implement the economic model as a simulator to validate the approach for CPU resource allocation. In the experiments, the TPC-C benchmark workloads are used to represent competing workloads running on a DB2 DBMS. We use the simulator to simulate the CPU resource allocation processes and run the workloads on the DB2 database server to verify their performance. Normalized throughput is taken as a performance metric to measure the workload performance. The experimental results show that with using the economic model, the system CPU resource allocations matches the given workload business importance policy, and the workloads' performance with the allocated CPU resources matches their business importance levels.

Through showing the application of using economic models to allocate CPU resources and the previous application of utilizing the economic models to manage buffer pool memory space [7], we have demonstrated that economic models can be successfully applied on single resource allocations in a DBMS. In Chapter 5, we illustrate that the economic model can be extended to manage multiple resources simultaneously, namely buffer pool memory space and CPU resources in a DBMS.

Chapter 5
Allocating Multiple Resources

As illustrated earlier in our resource allocation studies, economic models have been successfully used to provide timely and sufficient allocations of single critical DBMS resources to competing workloads based on the workload business importance policies. To further investigate the use of the economic model to manage shared DBMS resources, we extend our approach to the case of simultaneously allocating multiple resources in a DBMS. In order to use an economic model in a DBMS to make system-level multiple resource allocation plans for competing workloads, there are several issues that must first be addressed in this resource management study. These issues include how to partition the shared DBMS resources among the competing workloads, how to identify the resource preferences of a workload among the multiple resources in a resource allocation process, and how to predict performance of a workload with its certain amount of allocated resources. In our study, we specifically consider allocating buffer pool memory space and system CPU resources described in Section 3.1. To address these issues, there are three main components in our approach:

- **Resource model**: The resource model determines how to partition the buffer pool memory space and system CPU resources and what are reasonable total amounts of the two resources for allocation.
- **Resource allocation method**: The resource allocation method determines how to obtain optimal buffer pool memory and CPU share allocations in order to benefit the workload and system-wide performance most and achieve the workload performance objectives.

- **Performance model**: The performance model predicts the performance of a workload with certain amount of allocated resource shares in order to determine the benefit of the resources.

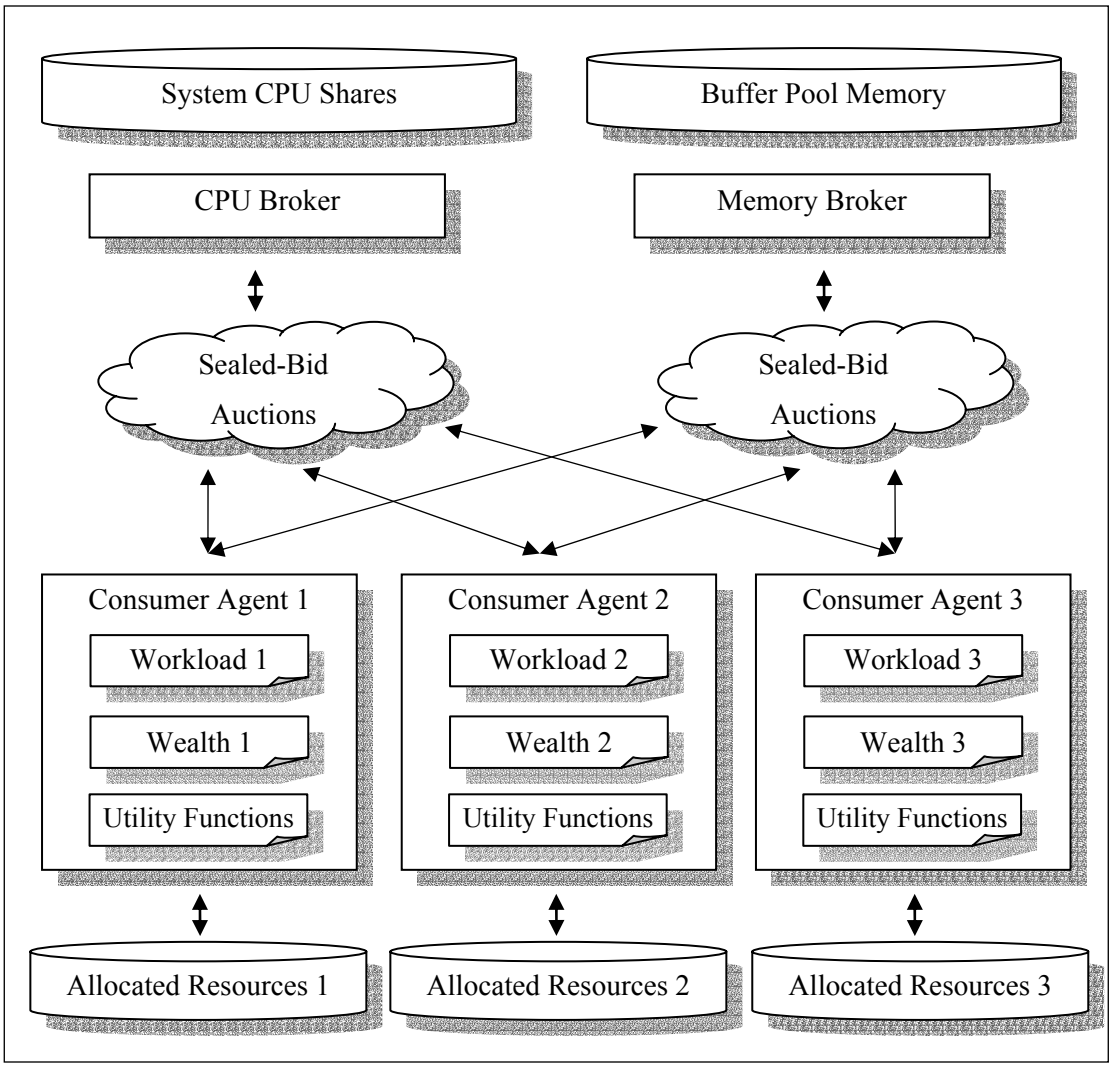

Figure 10: Economic Model for Multiple Resource Allocation

In this chapter, we describe a framework that uses an economic model to simultaneously allocate multiple resources to competing database workloads in order to achieve the workload

performance objectives. The competing workloads, as described in Chapter 4, are from different business importance classes and have unique performance objectives. The framework for multiple resource allocation, shown in Figure 10, consists of a set of shared resources, namely system CPU shares and buffer pool memory space, two resource brokers, one for CPU and one for memory, and consumer agents. The consumer agents represent competing workloads, which are in different importance classes, namely high importance, normal importance and best effort, as described in Section 3.4, to compete for the shared resources. The CPU and memory brokers conduct auctions simultaneously to sell their resources, and the consumer agents bid for shares of the resources. The total amount of the two resources for allocations in the economic model is determined by resource model and the multiple-resource utility function is defined in the consumer agents.

The validation of our approach is presented at the end of this chapter. We extend the economic model simulator from single-resource allocation to simulate multiple-resource allocation processes and run the database workloads with their allocated resource shares on a DBMS to verify that their performance matches a given business importance policy. In experiments, OLTP workloads are used to run on a DB2 DBMS and system throughput and database buffer pool hit rate are measured to represent the DBMS performance under different configurations.

5.1 Resource Model

As discussed earlier, the resource model is used to partition resources and to determine a reasonable total amount of the resources for allocation. Simply assigning OS priorities to competing workloads according to the workloads' business importance levels is not sufficient since the workloads may not be able to meet their performance requirements at the same time.

Therefore, we use the relative importance to describe workload business importance in our approach. We assume shared resources can be partitioned, thus individual competing workloads can be assigned a certain amount of the resource shares to achieve their performance objectives.

We combine our previous resource models for buffer pool memory and CPU resources together in this multiple-resource allocation framework. Each workload is assigned its own buffer pool so buffer pool memory pages can be assigned directly to a workload. The CPU resource is partitioned by controlling the number of database agents that are available to serve requests on a database server. We configure the DBMS such that one database agent maintains one client connection from the competing workloads. The total amounts of resources are parameters in the multiple-resource allocation framework, so the approach can adapt to different database server configurations.

5.2 Resource Allocation Method

As introduced in Section 3.3, the goal of the consumer agents in the economic model is to acquire more shared resources in order to maximize the performance of the competing workloads. In the case of multiple resource allocations, a consumer agent must capture the resources in appropriate amounts such that none of the resources become a performance-limiting resource [28]. As an example, in a resource allocation process a consumer agent might randomly capture a large amount of system CPU shares and a small amount of buffer pool memory resources. With this result, the buffer pool memory resources would become the bottleneck resource and significantly limit the workload performance. The CPU resources, on the other hand, would be under-utilized. In order to maximize workload performance and resource utilization, a consumer agent needs a mechanism to identify the resource preferences of a workload in order to obtain an optimal $<cpu_{opt}, mem_{opt}>$ resource pair.

In our approach, consumer agents use a greedy algorithm to identify the optimal <cpu_{opt}, mem_{opt}> resource pairs for their workloads. The multiple resource allocation is determined iteratively. In an iteration of the greedy algorithm, a consumer agent bids for a unit of the resources (either buffer pool memory or CPU) that it predicts will yield the greatest benefit to its performance.

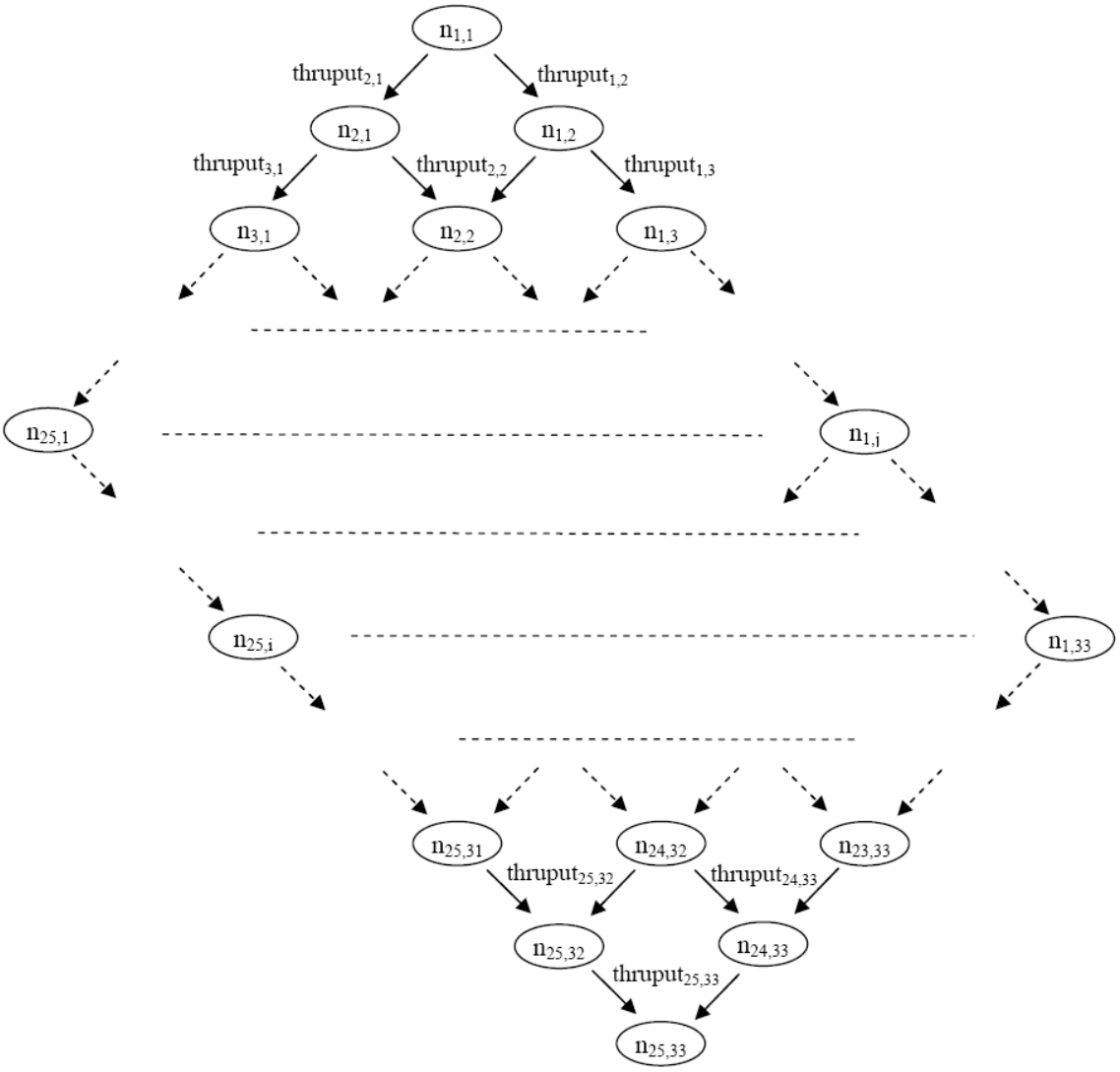

Figure 11: Buffer Pool Memory and CPU Resource Pairs

Figure 11 shows a representation of the search state space for the allocation of buffer pool memory and CPU resources to an OLTP workload in our experiments. The starting node, $n_{1,1,}$ represents the minimum allocation given to each consumer agent, namely one unit of buffer pool memory and one unit of CPU resources, at the beginning of a resource allocation process. A consumer agent then traverses this directed weighted graph to search for the optimal $<cpu_{opt}, mem_{opt}>$ resource pair for its workload in order to achieve the highest performance based on its wealth.

A node $n_{i,j}$ in the graph represents a possible resource allocation pair of i units of CPU and j units of buffer pool memory. The weight on the edge $(n_{i,j}, n_{i,j+1})$ represents the predicted performance (in our case throughput) that could be achieved by the workload if the consumer agent acquires one more unit of buffer pool memory. Similarly, the weight on an edge $(n_{i,j}, n_{i+1,j})$ represents the predicted performance of the workload that could be achieved if one more unit of CPU resource is acquired. A consumer agent always chooses to bid for the resource that will yield the best performance. If a consumer agent's bid is successful then conceptually it moves to the new node in the graph for the next iteration of the algorithm. If a consumer agent's bid is not successful then it remains at the same node in the graph and bids for the same resource again in the next iteration.

The graph has no cycles and the weight on each edge is determined by the performance model described in Section 5.3. The boundary nodes in Figure 11 have one or no outgoing edges when there is no more associated resource left. The proof that, for the optimal resource pair searching problem, a greedy algorithm generates a global optimum is easy to show since the addition of more resources will always result in a workload performance improvement at each step in the resource allocation algorithm, and this predicate (loop invariant) holds since the DBMS is not saturated until all the resources are claimed.

Besides the greedy algorithm, there might be some other methods which could be used to determine the resource allocations in the multiple resource allocation process. An interesting random method is the two resources alternating. Instead of identifying the critical resource at each resource allocation step, workload just bids on one unit of a resource that differs from the resource obtained in the previous resource allocation. For example, a workload is going to bid on a unit of CPU resources if it assigned a unit of buffer pool memory resources in its previous resource allocation. We compare these two methods and show their results in Section 5.3.2.

5.3 Performance Model

We predict performance of our DB2 DBMS and define the multiple resource utility function based on a performance model. The performance model estimates system throughput of the DB2 DBMS with competing workloads running on it. As introduced in Section 4.2, a QNM can be used to predict the throughput, utilization, and latency of a computer system and its components, thus we apply QNMs to build a DB2 DBMS performance model in this study.

5.3.1 DB2 Queueing Network Model

We model the DB2 DBMS used in our experiments with a simple QNM. This DB2 QNM is used to predict performance of a workload at each step of the greedy algorithm, that is, to assign the weights to the edges of the graph in Figure 11. We use a closed QNM in our study as the number of requests in a database system would be fixed after a resource allocation step. Figure 12 shows our closed QNM which consists of a CPU service center, an I/O service center, and a number of users concurrently running on it. The CPU service center represents the system CPU resources and the I/O service center represents buffer pool memory and disk system resources. We apply mean value analysis (MVA) [21] (see Appendix A for details) on this closed DB2 QNM to estimate throughput of the DBMS running an OLTP workload with a certain number of database

agents and a particular size of buffer pool. Single class workloads are considered on the DB2 QNM in this study, but this DB2 QNM could also be extended to multiple classes to handle workloads of different types.

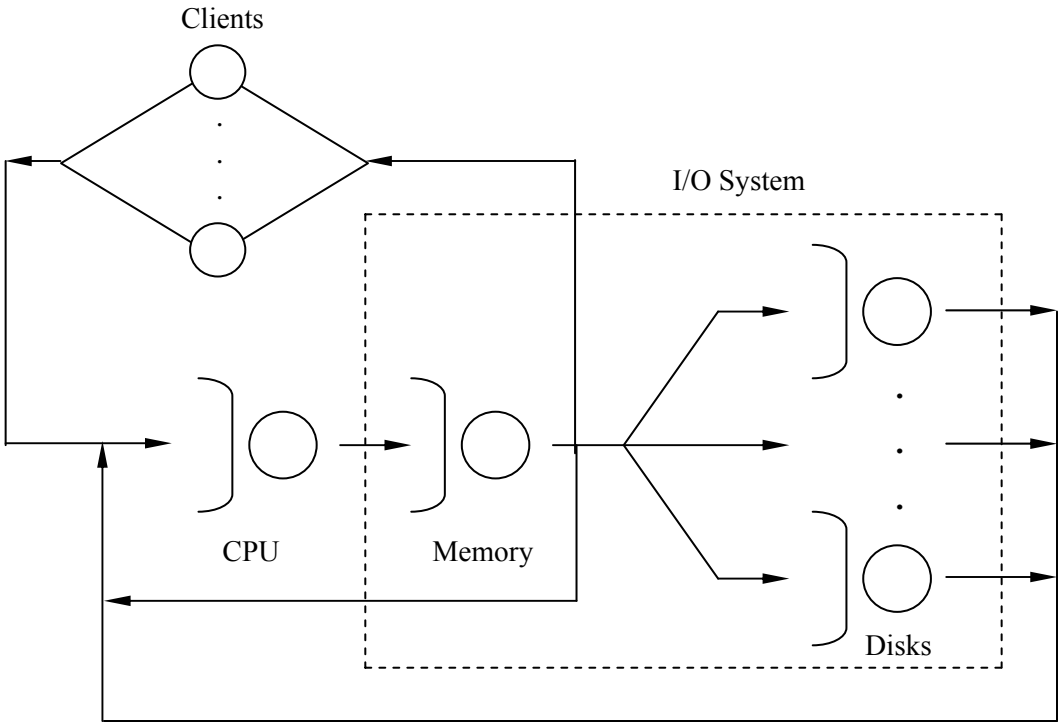

Figure 12: DB2 Queueing Network Model

5.3.2 DB2 QNM Parameterization and Validation

Since we consider OLTP workloads in our experiments, to parameterize the DB2 QNM with respect to the specific workloads, we need to determine the average service demand of transactions in the workloads from the CPU service center and I/O service center respectively. The OLTP workloads are taken from the OLTP TPC-C benchmark system [36] (see Appendix B for details).

As described in Section 4.2.2, for a specific database system and a specific workload, the average CPU service demand of a transaction is constant and independent of the number of

transactions concurrently in the system, and the service demand value can be experimentally determined using equation (4.2). Since our experimental environment is the same as the one described in Section 4.3.1, we can use the previous determined values in this study, namely the average CPU service demand, S_{cpu}, *is 0.0181* (*seconds/transaction*) for a transaction in the standard OLTP workloads under our specific experimental environment described in Section 5.4.

I/O service demand of transactions in a workload directly depends on the database buffer pool size. If a database buffer pool size is large, then it can reduce the number of I/O operations required, which means a smaller average I/O service time for a request. On the other hand, if the buffer pool size is small, it can increase the number of I/O operations from disk system and so increases the average I/O service time for a request. For OLTP workloads, the average I/O service demands can be expressed as a function of buffer pool memory size, which is derived from Belady's equation [2]. The derived average I/O service demand equation is [40]:

$$S_{IO} = c * M^{b} \tag{5.1}$$

where, c and b are constants, and M is buffer pool size. In the equation (5.1) the constants c and b can be determined by equations (5.2) and (5.3) respectively. Different database systems would have different b and c values.

For constant b:

$$b = \ln\left(\frac{S_{IO}1}{S_{IO}2}\right) \Big/ \ln\left(\frac{M1}{M2}\right) \tag{5.2}$$

For constant c:

$$c = \left(S_{IO}1\right)/\left(M1\right)^{b} \tag{5.3}$$

where $S_{IO}1$ and $S_{IO}2$ are I/O service demand values at buffer pool sizes of *M1* and *M2* buffer pages respectively. $S_{IO}1$ and $S_{IO}2$ are determined experimentally. For the TPC-C benchmark system with our specific experimental environment, we determined that the values for constants *b* and *c* are *-0.513* and *7.481*, respectively. Figure 13 shows a validation of the I/O service demand model defined in the equation (5.1). It shows the estimated and actual I/O service demand values for various buffer pool sizes when standard OLTP workloads are running on the database system. All the experimental numbers represent averages over 5 runs, and a 90% confidence level with ±10 confidence intervals for each data point.

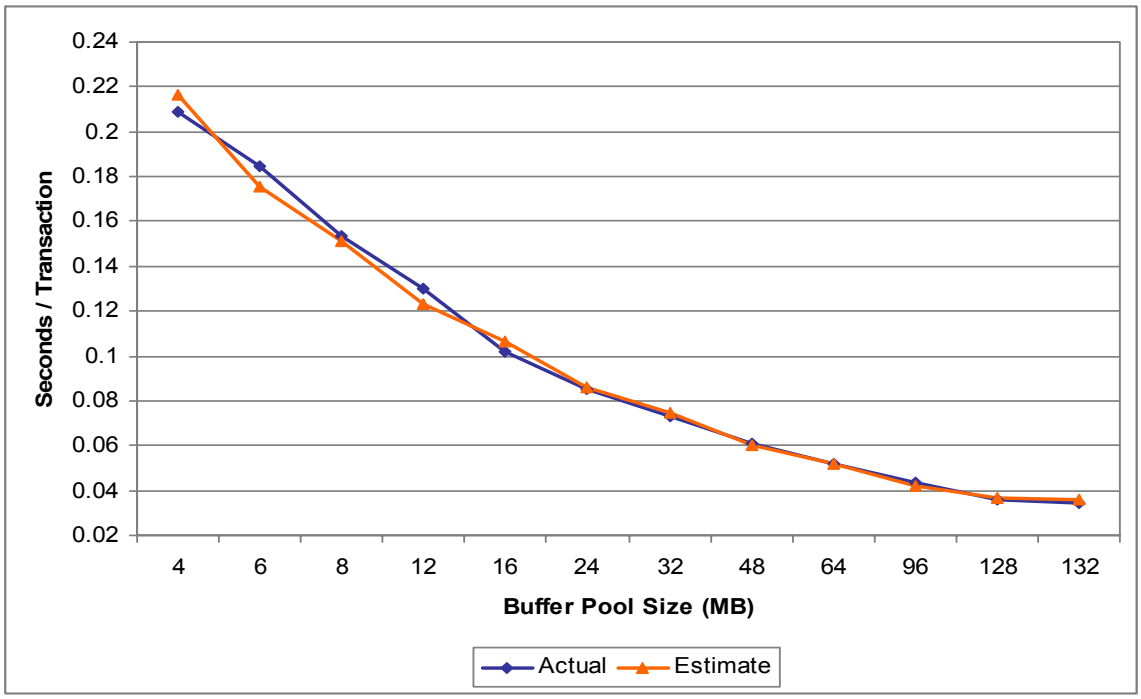

Figure 13: Validation of I/O System Service Demand

With the DB2 QNM parameterized, the system throughput can be estimated by using MVA [21] (see Appendix A for details) on the closed QNM for a standard OLTP workload running on it with specific amounts of buffer pool memory and CPU resources. Figure 14 shows

a validation of the DB2 QNM. It presents the predicted and actual throughput values for various resource pairs for the DBMS and TPC-C benchmark workload. The resource pairs are evenly chosen across the two resource ranges. All the experimental numbers are averages over 5 runs, and a 90% confidence level with ±10 confidence intervals is used for each data point.

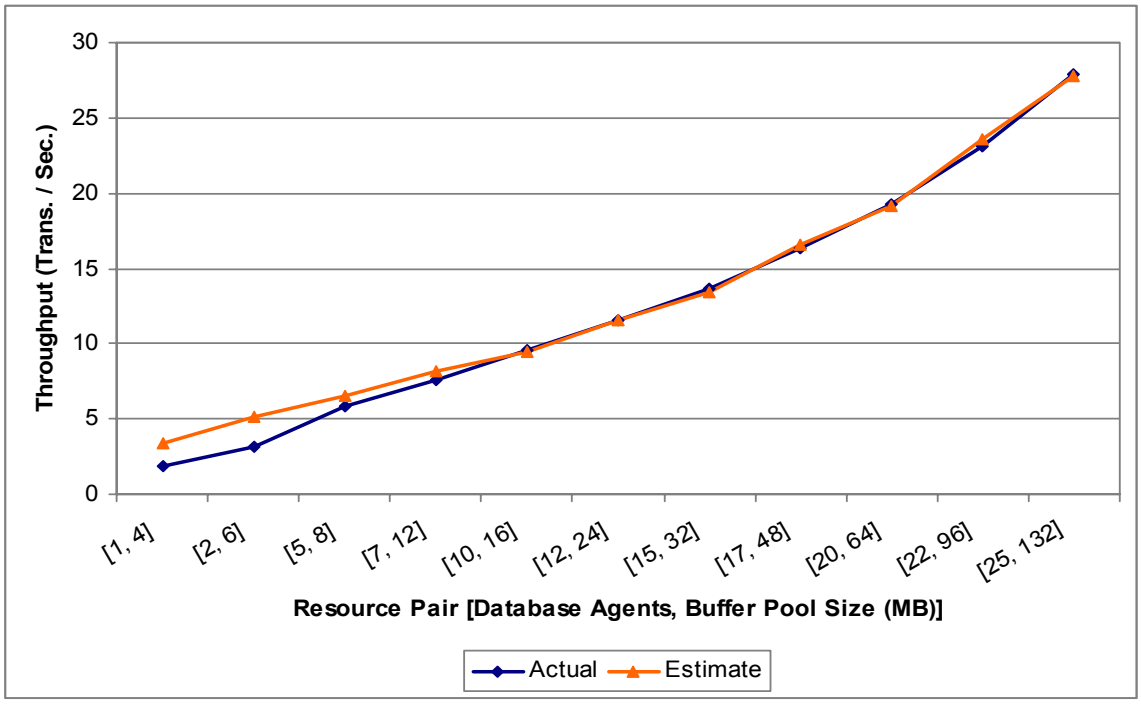

Figure 14: Validation of Performance Model

Figure 15 shows a comparison of using the greedy algorithm and alternating approach in our resource allocation method described in Section 5.2. For a competing workload, by utilizing the two different methods to determine the resource allocations, the workload would obtain different resource assignments at each step in the multiple resource allocation process. Applying the proposed performance model, we can predict the workload performance with its allocated resources by the two methods at each step. Based on the results shown in Figure 15, we see that

the greedy algorithm generates adequate resource allocation at each step to a workload, and hence the workload achieves better performance then the alternating approach provides. If all resources are assigned to a single workload, then the workload would achieve its maximum performance no matter what resource allocation method is used.

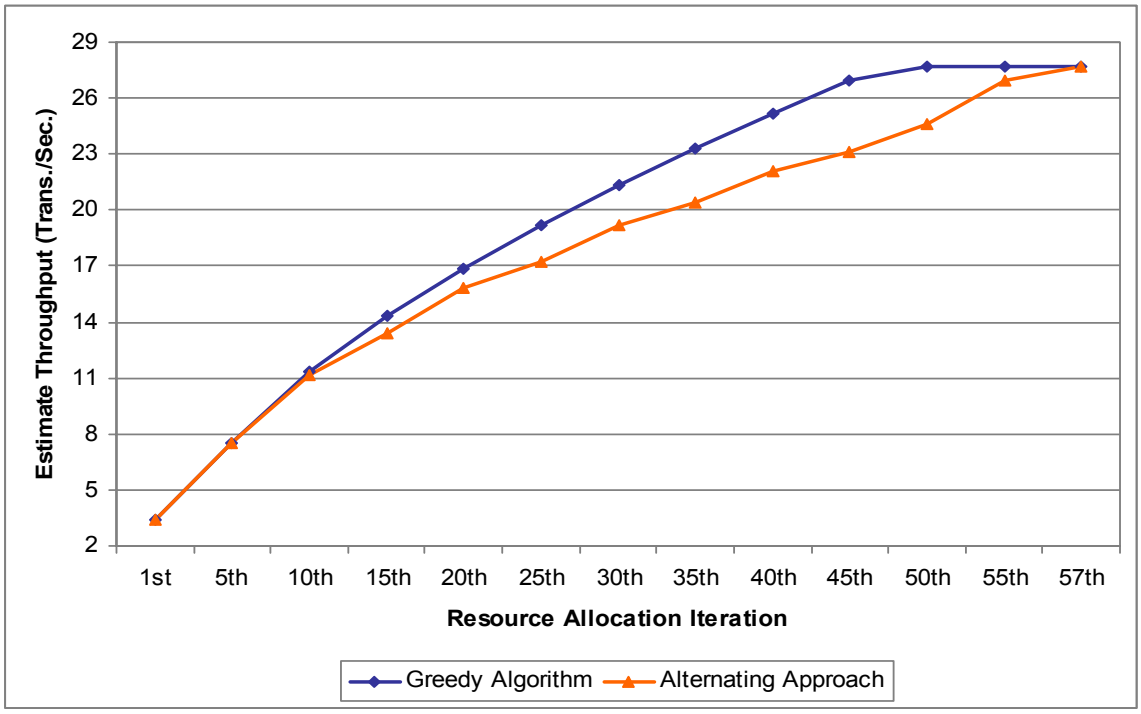

Figure 15: Comparison between Greedy Algorithm and Alternating Approach

5.3.3 The Multiple Resource Utility Function

Following the guidelines of defining utility functions described in Section 3.3.2, we employ a utility function that normalizes the predicted throughput from the performance model relative to the maximum throughput that the workload could achieve when all resources are allocated to it. The utility function is given by:

$$Utility(c_{CPU}, m_{MEM}) = MVA_{Throughput}(c_{CPU}, m_{MEM}) / X_{Max} \quad (5.4)$$

where $MVA_{Throughput}(x, y)$ is the throughput provided by the QNM, c_{CPU} is the number of database agents and m_{MEM} is the number of buffer memory pages. X_{Max} is the maximum throughput achieved by a workload with all resources allocated. Since all of the experimental workloads are of the same type, this defined utility function can be applied to all the workloads.

5.4 Experimental Validation

We extended our economic model simulator for single resource allocations described in Section 4.3 to handle the multiple resource case. The experiments designed for validating the multiple resource allocation approach are similar to the ones used for single resource allocation studies. The validation addresses two main issues in the multiple-resource management studies, namely:

- Does the resource allocation framework generate the multiple-resource allocations for competing workloads on a DBMS so that the resource allocations match the workload business importance policies?
- Are the resource allocations resulting from the auctions appropriate for the competing workloads to achieve performances that match the workloads' business importance levels?

5.4.1 Experimental Environments

All experiments are conducted with the system platform described in Section 4.3.1, namely DB2 Universal Database Version 8.2 [15] running on an IBM xSeries 240 PC server with the Windows XP operating system. The database server is equipped with two 1 GHz Pentium 3 processors, 2 GB of RAM and an array of 11 disks. To eliminate the need to account for CPU parallelism in the DB2 QNM introduced in Section 5.2, we configure the system with one processor available. Windows XP operating system's performance monitor is used to collect CPU and disk utilization data.

We apply the economic model to a single DB2 instance with an identical database for each competing workload running on the database server. The workloads have different importance levels and performance objectives. Each database is configured with one buffer pool and some number of database agents. Each database has one workload running on it, thus each workload has its own buffer pool and some system CPU shares. We use three standard OLTP workloads in the experiments, so the economic model allocates buffer pool memory space and system CPU resources across three identical databases based on a given workload business importance policy.

We experimentally determined the appropriate amount of total resources for their allocations under the given DBMS configuration and the set of workloads. We use 32768 4KB buffer memory pages as the total buffer pool memory resources [7] and 25 database agents as the total CPU resources described in Section 4.1.

The unit sizes of buffer pool memory and CPU resources are parameters in the economic model framework. We use 1000 buffer memory pages as one unit of buffer memory and 1 database agent as one unit of CPU resources in this study as these granularities give a reasonable workload performance increment and make the resource allocation efficient.

As described in Section 4.3.1, we take the experimental database and workloads from the OLTP TPC-C benchmark [36]. The size of each database is 10GB. The three workloads are TPCC-like OLTP batch workloads stored in three script files which are the same files used for the single resource allocation studies described in Section 4.3.1. The workloads send transactions to the DBMS with zero think time during their run. Each workload consists of 120,000 requests based on the 5 different types of the transactions of TPC-C benchmark. A workload is divided into 12 resource allocation intervals. The start of each interval provides a checkpoint at which a

resource allocation process takes place to reallocate resources to workloads based on their current importance levels and wealth.

The economic model simulator is written in Java (see Appendix C for UML class diagram) and the workload script files are used as the simulator input. The output of the simulator is the plan of the multiple resource allocations, that is, a list of the number of buffer pool memory pages and database agents for the competing workloads at each of their checkpoints. According to the output, database configuration commands are inserted into the workload script files at each of the checkpoints, allowing for a database to dynamically reconfigure the resources during a run. By running the workloads on the DB2 database server, we verify that the workloads' performance matches their business importance levels. Before each run of the workloads, the databases are restored to their initial states. Buffer pool hit rate and workload throughput are taken to measure the workloads' performance. Each experiment was run five times, and the average over the five runs and 90% confidence level with ±10% confidence intervals are used for each reported number.

5.4.2 Resource Allocations

The first set of experiments was conducted to determine whether our approach generates the multiple resource allocations that match a given workload's business importance policy. The workloads are assigned one of three different importance classes, namely the high importance class, the normal importance class, and the best effort class, as described in Section 3.4. As in Section 4.3.2, we experimented with three different sets of importance multipliers to validate the multiple resource allocations with the economic model. An importance multiplier set represents the importance multiplier values of the three importance classes in order of {best effort, normal importance, high importance}. The three sets used were {1, 1, 1}, {1, 5, 6}, and {1, 5, 10}. These

configurations were designed to examine how resource allocations change with the importance multipliers.

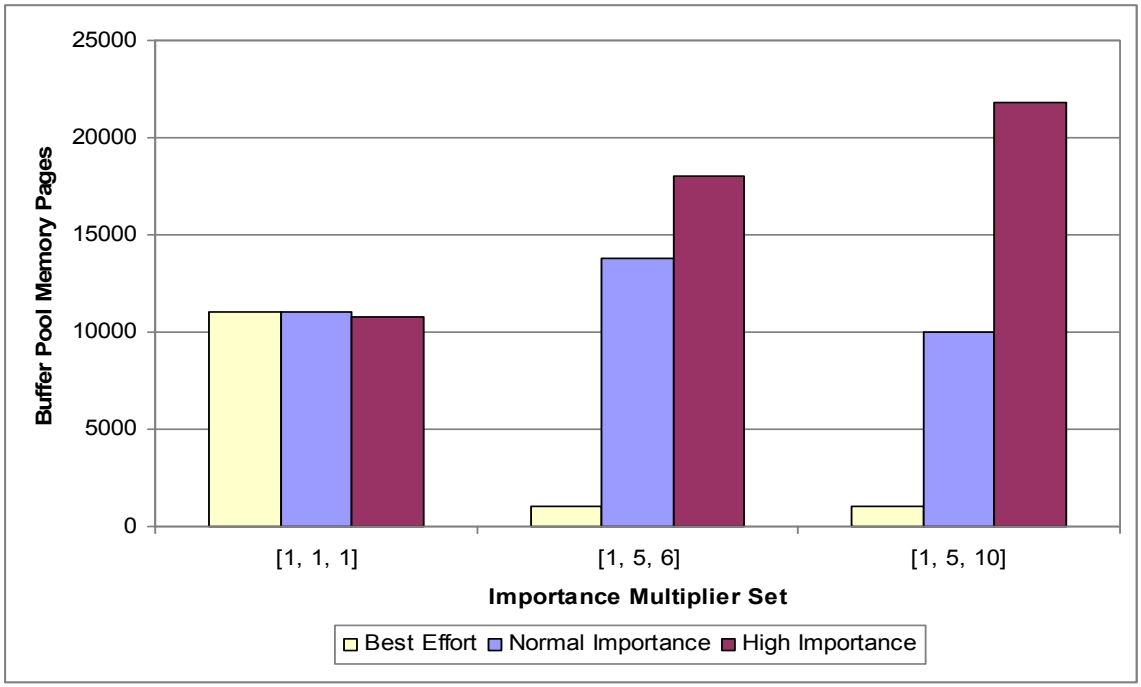

Figure 16: Buffer Pool Memory Allocations for Different Importance Multiplier Sets

Figure 16 and Figure 17 show buffer pool memory page and system CPU share (represented by database agents) allocations produced by the economic model using the three workload business importance multiplier sets. The workload importance multiplier set {1, 1, 1} represents the case where the three competing workloads are from three different business importance classes of equal importance. In this case, the three workloads are allocated approximately the same amount of buffer memory and CPU resources as shown in Figures 16 and 17. Using the importance multiplier set {1, 5, 6}, the high importance and the normal importance classes are much more important than best effort class, and the high importance class is also slightly more important than the normal important class. When the economic model is used to

allocate resources in this case, the high importance and normal importance workloads are allocated significantly more resources than the best effort workload, while the high importance workload is allocated slightly more resources than the normal importance workload. The set {1, 5, 10} represents the case where the high importance class is much more important than the normal importance class, and the normal importance class is much more important than the best effort class. In this case, the high importance workload is allocated more resources than the normal important workload, and the normal importance workload wins significantly more resources than the best effort workload.

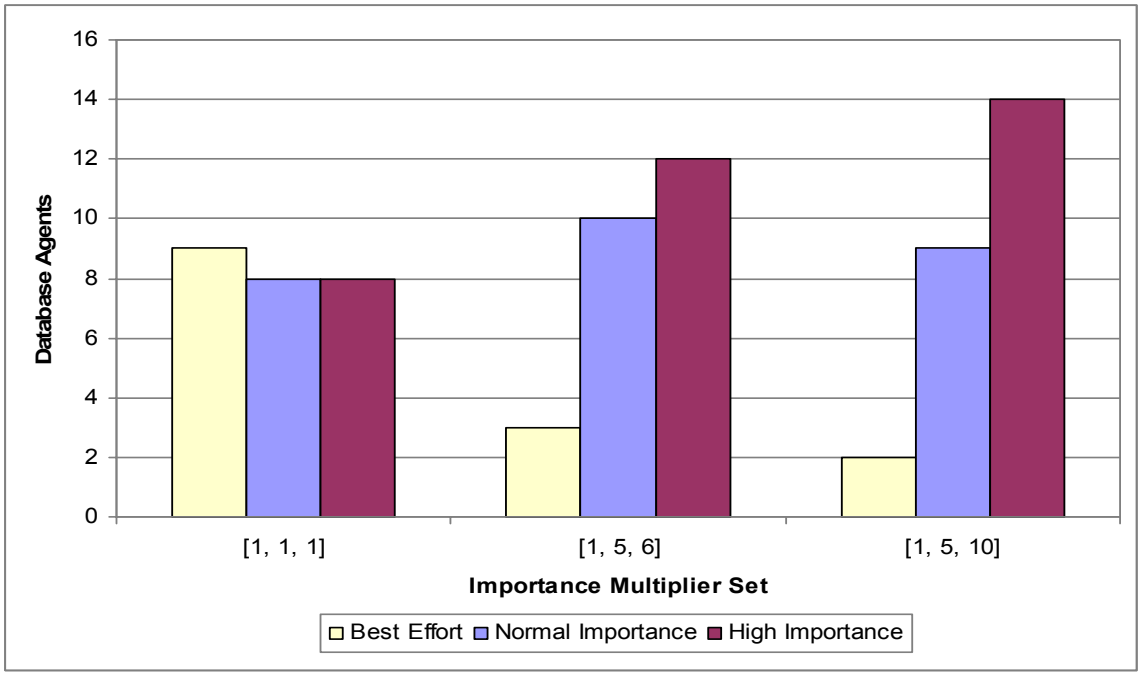

Figure 17: CPU Resource Allocations for Different Importance Multiplier Sets

Based on these experimental results, we conclude that the multiple resource allocations generated by the economic model for competing workloads match the given workload business importance policies, and shows more important workloads winning more resources than less

important workloads. Comparing these results with the results obtained in our previous single resource allocation studies, we see that the trend of multiple resource allocations is same as the trend shown in the single resource allocations, which is more important workloads assigned more resources than less important workloads. The amounts of assigned resources to individual workloads, however, are different. The difference is attributed to the fact that the resource allocation methods and the resource utility functions are different in the single and multiple resource management studies. We notice that the amount of allocated resources for competing workloads, shown in Figure 16 and Figure 17, presents a small difference when applying the importance multiplier set {1, 1, 1}. This difference is due to the fact that the economic model simulator assigns the first bidder resources when all the bidders submit the same bids.

5.4.3 Workload Performance

The second set of experiments was conducted to determine whether the performance achieved by the competing workloads with their allocated resources generated by the economic model match a given business importance policy. In the experiments, the workload importance multiplier sets were the same as the sets used in the experiments shown in Section 5.4.2. We could therefore observe the achieved performance of the workloads with the allocated resources obtained in the first set of experiments. The workloads were run on the DB2 DBMS using the resource allocations suggested by our simulator. Database throughput and buffer pool hit rate were measured as performance metrics. The experimental results are shown in Figure 18 and Figure 19 respectively.

In Figure 18 we observe that the throughput of the workloads achieved match the given business importance policies. When the competing workloads have the same business importance, namely applying the importance multiplier set {1, 1, 1} on the economic model, the

workloads achieve a similar throughput. When the workloads are of different business importance levels, namely applying multiplier sets {1, 5, 6} and {1, 5, 10} on the economic model respectively, then the important workloads achieve higher throughput than the less important workloads. The trend shows that the more important a workload is, the more resources it receives and hence, the workload performance is greater. Based on the results shown in Figure 18, we notice that throughputs of the workloads achieved present little differences with applying the importance multiplier set {1, 1, 1}. The differences are due to the fact that the amounts of allocated resources to the competing workloads have little differences described in Section 5.4.2.

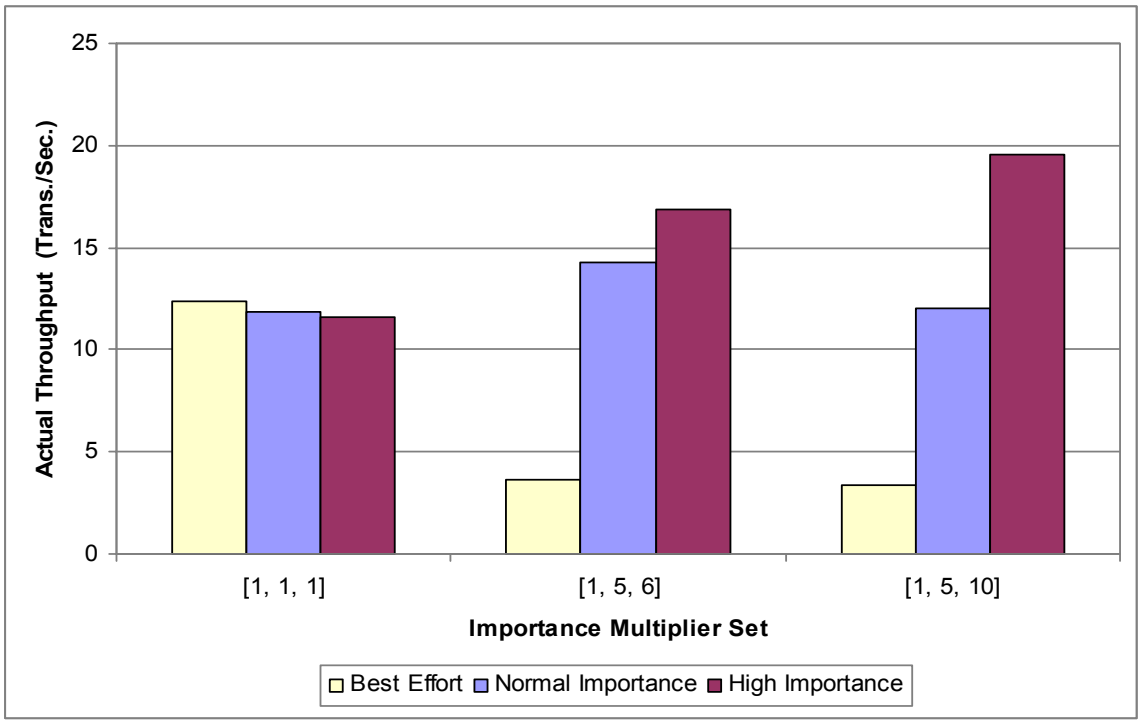

Figure 18: Workload Throughput for Different Importance Multiplier Sets

Figure 19 shows the buffer pool hit rate of the workloads for the three different importance multiplier sets. The results show the same trend as the throughput results shown in

Figure 18. When the workloads had the same business importance, then database buffer pool hit rates are similar, while with different business importance levels, the important workloads achieve higher buffer pool hit rates than the less importance workloads. We notice that the difference in the hit rate between the high importance class and the normal importance class is small when the difference between their importance multipliers is large (that is, with using the multiplier set {1, 5, 10}). This is likely due to fact that the total amount of buffer pool memory available for allocation is small, about 132MB in our experiments. So even if the high importance workload obtained more buffer pool memory pages, it could not achieve significantly higher buffer pool hit rate than the others. Based on our experimental results, we conclude that our framework for multiple resource allocations is successful in obtaining performance that matches the competing workloads' business importance policies.

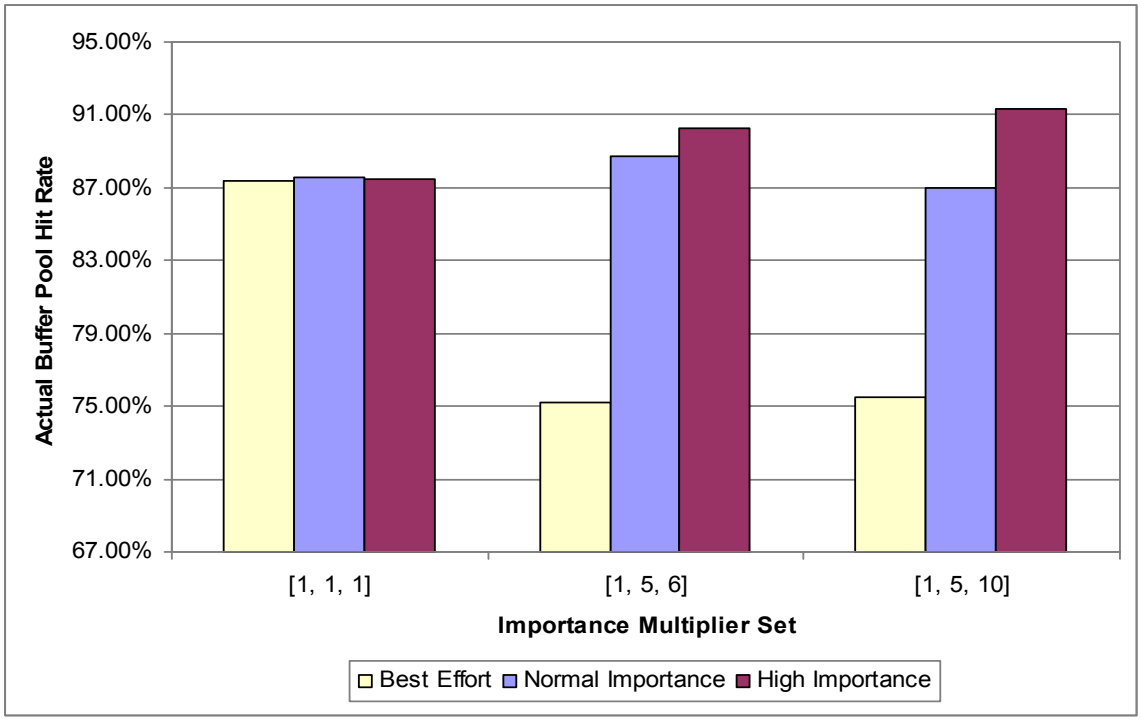

Figure 19: Buffer Pool Hit Rate for Different Importance Multiplier Sets

5.5 Summary

In this chapter, we demonstrate an approach that uses an economic model to allocate multiple resources to competing workloads on a DBMS. The approach consists of three main components, namely resource models, a resource allocation method, and a performance model. The competing database workloads have unique performance objectives and are labeled with different business importance. We specifically investigate simultaneously allocating buffer pool memory space and system CPU resources.

The buffer pool memory and CPU resource modeling methods, proposed in our previous studies, are applied to partition the multiple resources respectively and to determine the reasonable total amount the resources for allocation in the economic model. We propose a resource allocation method in the approach, based on a greedy algorithm, to find optimal resource pairs for the competing workloads to achieve their performance objectives. By using the resource allocation method, consumer agents can determine bottleneck resources and identify resource preferences of a workload in the resource allocation process. In order to predict system performance and define the multiple resource utility function, we apply QNMs to estimate the system performance. The DB2 QNM is built as a closed QNM with a CPU and an I/O service centers to represent system CPU and I/O resources respectively. MVA is applied on the closed DB2 QNM to estimate the system performance for a workload running on it with certain amount of buffer pool memory and CPU resources.

We implement the economic model as a simulator to validate our approach for the multiple resource allocation. In the experiments, the TPC-C benchmark workloads are used to represent competing workloads running on a DB2 DBMS. We use the simulator to simulate the multiple resource allocation processes and run the workloads with their allocated resources on the DB2 database server to verify the workloads' performances. Workload throughput and database

buffer pool hit rate are taken as performance metrics to measure the competing workloads' performance. The experimental results show that by using the economic model, the multiple resource allocations match the given workload business importance policies and the workloads' performances obtained with their allocated resources match the workload business importance levels.

Chapter 6
Conclusions and Future Work

6.1 Conclusions

In this thesis we study resource allocation in a DBMS. We consider the business-level workload importance policy as an administrative objective given to an autonomic DBMS to manage resource allocation for its competing workloads.

We present a general framework that uses an economic model to allocate resources in an autonomic DBMS. The economic model consists of four components, namely shared resources, resource brokers, a trade mechanism, and consumer agents. Resource brokers conduct auctions to sell the shared resources, and consumer agents represent workloads to bid for the shared resources via an auctioning and biding based trade mechanism. A consumer agent, in the economic model, consists of a workload, a certain amount of wealth, and a utility function. The wealth is determined by the workload cost and the workload business importance level. By using its utility function, the consumer agent determines the maximum bids in the resource auctions.

To verify the effectiveness of the framework, we apply the economic model to allocate system CPU shares and to simultaneously allocate multiple resources, namely buffer pool memory space and system CPU shares, to competing workloads in a DBMS based on the workload business importance.

For allocating system CPU resources, we propose the resource modeling method, through controlling the number of database agents used in a DBMS, to partition system CPU resources and experimentally determine the reasonable total amount of CPU resources for allocation. In order to define the CPU resource utility function, we apply a queueing network model on our DB2 DBMS to estimate the system performance for a workload running on it with certain amount

of allocated CPU resources. For allocating the multiple resources, we propose a resource allocation method that uses a greedy algorithm to find optimal resource pairs for the competing workloads to achieve their performance objectives. In order to predict system performance and define the multiple resource utility function, we apply a queueing network model again on our DB2 DBMS to estimate the system performance for a competing workload with certain amount of buffer pool memory and CPU resources.

We extend the economic model simulator created by Boughton [7] to validate our approaches for the CPU and multiple resource allocations. In the experiments, TPC-C benchmark workloads are used on a DB2 DBMS. We use the simulator to simulate the resource allocation processes and run the workloads with their allocated resources on the DB2 database server to verify the competing workloads' performances. The experimental results show that with using the economic model, the resource allocations match the given workload business importance policies and the workloads' performances achieved with their allocated resources match the workload business importance levels.

A challenge of using the economic model for the resource allocation is to define the resource utility function. As a key mechanism in the resource allocation process, resource utility functions must first be defined in an economic model. In our study, a utility function maps usefulness of resources to performance of a workload achieved, and it also needs to be a non-decreasing function with return values in real number range [0, …, 1]. To achieve these features, in our approach we define the resource utility functions based on our proposed performance models.

In conclusion, our study makes three contributions to performance management. The first contribution of the work is an investigation of using an economic model to tune CPU resource

allocations in a DBMS for multi-class workloads with unique performance requirements. The second contribution is the specification of a resource allocation framework that uses an economic model to manage multiple resources for multi-class workloads with unique performance objectives. The third contribution is the extension of the economic model simulator created by Boughton [7] to validate the resource allocation framework. By using the economic model, we show a way that a high-level business importance policy can be automatically translated into an autonomic DBMS system-level resource tuning actions. Through experiments we have validated the effectiveness of the framework.

6.2 Future Work

To implement the resource allocation framework within a DBMS, there is an issue that needs first to be addressed. In this study, we use batch workloads in which all queries are known in advance. But when integrating the resource allocation framework into a DBMS, the issue is to predict the cost of a workload in an upcoming resource allocation interval. There are several possible approaches that could be used to address this issue. One approach is to estimate a workload cost for an upcoming interval based on the costs of the workload in previous intervals. Another approach is to allow consumer agents to request additional resources at any time by utilizing the feedback loop feature provided by the autonomic DBMS to verify that the workload SLAs are met, and not wait for the next resource allocation interval. The third approach is to intercept the incoming requests by using the DBMS performance management component, such as DB2 Query Patroller [16], to estimate the query costs in a workload.

In the next step of our research, we plan to add OLAP workloads to study the effectiveness of the economic model with more complex workloads in order to extend the economic model to accept random workloads. We also plan to try exchange based economy to

replace the current price based economy to allow the consumer agents to directly exchange their resources in order to compare to the trading mechanism we currently use in the economic model.

References

[1] M. Amirijoo, P. Brännström, J. Hansson, S. Gunnarsson, and S. H. Son. “Toward Adaptive Control of QoS-Importance Decoupled Real-Time Systems”. IEEE International Workshop on Feedback Control Implementation and Design in Computing Systems and Networks, Germany, 2007.

[2] L. A. Belady. “A Study of Replacement Algorithms for a Virtual-Storage Computer”. IBM Systems Journal, Volume 5, Number 2, Page 78, 1966.

[3] M. N. Bennani, D. A. Menasce. "Resource Allocation for Autonomic Data Centers using Analytic Performance Models," Proc. Intl. Conf. Autonomic Computing, (ICAC 2005), pp.229-240, 13-16 June 2005.

[4] M. N. Bennani and D. A. Menasce. “Assessing the Robustness of Self-managing Computer Systems under Variable Workloads”, Proc. Intl. Conf. Autonomic Computing (ICAC’04), New York, NY, May 17–18, 2004.

[5] K. P. Brown, M. Mehta, M. J. Carey, and M. Livny. “Towards Automated Performance Tuning for Complex Workloads”, Proc. of the 20th Intl. Conf. of Very Large Data Bases, pages 72-84, Santiago, Chile, Sept. 1994.

[6] H. Boughton, P. Martin, W. Powley, and R. Horman. “Workload Class Importance Policy in Autonomic Database Management Systems”, Seventh IEEE Intl. Workshop on Policies for Distributed Systems and Networks, pages 13-22, London, Canada, June 5 - 7, 2006.

[7] H. Boughton. “Policy-Based Tuning in Autonomic Database Management Systems Using Economic Models”. MSc Thesis, School of Computing, Queen's University, January, 2006.

[8] K. A. Delic. “Enterprise IT Complexity”. Ubiquity Information Everywhere, ACM, 2001, http://www.acm.org/ubiquity/views/k_delic_1.html.

[9] D. L. Davison, G. Graefe. “Dynamic Resource Brokering for Multi-User Query Execution”. In Proc. of the ACM SIGMOD Intl. Conf. on Management of Data, pages 281-292, San Jose, California, United States, June, 1995.

[10] D. F. Ferguson, C. Nikolaou, J. Sairamesh, Y. Yemini. “Economic Models for Allocating Resources in Computer Systems”. In Scott Clearwater, Editor, Market-Based Control: A Paradigm for Distributed Resource Allocation, Scott Clearwater. World Scientific, Hong Kong, 1996.

[11] A. G. Ganek and T. A. Corbi. “The Dawning of the Autonomic Computing Era”, IBM Systems Journal, Vol. 42, No. 1, 2003.

[12] IBM. “Autonomic Computing Concepts”. 2003. http://www-03.ibm.com/autonomic/pdfs/AC_Concepts.pdf.

[13] IBM. “Autonomic Computing and IBM”. 2002. http://www-03.ibm.com/autonomic/pdfs/AC_BrochureFinal.pdf.

[14] IBM DB2 Universal Database. Application Development Guide: Building and Running Applications. Version 8.2. ftp://ftp.software.ibm.com/ps/products/db2/info/vr82/pdf/en_US/db2axe81.pdf.

[15] IBM DB2 Universal Database. Administration Guide: Performance. Version 8.2. ftp://ftp.software.ibm.com/ps/products/db2/info/vr82/pdf/en_US/db2d3e81.pdf.

[16] IBM DB2 Query Patroller. DB2 Query Patroller Guide: Installation, Administration, and Usage. Version 8.2. ftp://ftp.software.ibm.com/ps/products/db2/info/vr82/pdf/en_US/db2dwe81.pdf.

[17] IBM. “Practical Autonomic Computing: Roadmap to Self Managing Technology". January, 2006. http://www-03.ibm.com/autonomic/pdfs/AC_PracticalRoadmapWhitepaper_051906.pdf.

[18] J. O. Kephart and D. M. Chess. "The Vision of Autonomic Computing". IEEE Computer, Volume 36, Issue 1, pages 41-52, 2003.

[19] J. F. Kurose and R. Simha. "A Microeconomic Approach to Optimal Resource Allocation in Distributed Computer Systems". IEEE Transactions on Computers. Vol. 38, No. 5, May 1989.

[20] S. Krompass, A. Scholz, M. C. Albutiu, H. Kuno, J. Wiener, U. Dayal and A. Kemper. "Quality of Service-Enabled Management of Database Workloads". Special Issue of IEEE Data Engineering Bulletin on Testing and Tuning of Database Systems, IEEE Computer Society, 2008.

[21] E. Lazowska, J. Zahorjan, G. S. Graham and K. C. Sevcik "Quantitative System Performance: Computer System Analysis Using Queueing Network Models". Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1984.

[22] D. A. Menasce and M. N. Bennani. "On the Use of Performance Models to Design Self-Managing Computer Systems," Proc. 2003 Computer Measurement Group Conf., Dallas, TX, Dec. 7-12, 2003.

[23] D. A. Menasce, "Automatic QoS Control," IEEE Internet Computing, Vol. 7, No. 1, Jan.-Feb. 2003.

[24] D. A. Menasce, R. Dodge, and D. Barbara, "Preserving QoS of E-commerce Sites through Self-Tuning: A Performance Model Approach," Proc. 2001 ACM Conf. Ecommerce, Tampa, FL, Oct. 14-17, 2001.

[25] D. S. Moore and G. P. McCabe. "Introduction to the Practice of Statistics" (3rd Edition). W. H. Freeman & Co. and Sumanas, Inc., 1998.

[26] B. Niu, P. Martin, W. Powley and P. Bird. "Quantifying Workload Importance in Self-Managing DBMSs". Fourth International Workshop on Engineering Autonomic Software Systems held in conjunction with CASCON 2007, Toronto, ON, Canada, October 23 - 24, 2007.

[27] B. Niu, P. Martin, W. Powley, R. Horman, and P. Bird. "Workload Adaptation in Autonomic DBMSs", Proc. of the 2006 Conf. of the Centre for Advanced Studies on Collaborative Research, Article No. 13, Toronto, Canada, Oct. 2006.

[28] D. Narayanan, E. Thereska, A. Ailamaki. "Continuous Resource Monitoring for Self-Predicting DBMS", Proc. of the 13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS'05), page 239 - 248, Sept. 27-29, 2005.

[29] G. Pacifici, M. Spreitzer, A.Tantawi, and A.Youssef. "Performance Management for Cluster Based Web Services", IEEE Journal on Selected Areas in Communications, Volume 23, Issue 12, page 2333-2343, Dec. 2005.

[30] R. Ramakrishnan and J. Gehrke. "Database Management Systems" (3rd Edition). McGraw-Hill Companies, Inc., 2003.

[31] M. Stonebraker, R. Devine, M. Kornacker, W. Litwin, A. Pfeffer, A. Sah, and C. Staelin. "An Economic Paradigm for Query Processing and Data Migration in Mariposa". Technical Report 49, University of California, Berkley, April 1994.

[32] B. Schroeder, M. Harchol-Balter, A. Iyengar, and E. Nahum. "Achieving Class-Based QoS for Transactional Workloads", Proc. of the 22nd Intl. Conf. on Data Engineering (ICDE'06), page 153, Apr. 03 - 07, 2006.

[33] N. Stratford, R. Mortier. "An Economic Approach to Adaptive Resource Management". Proc. of the Seventh Workshop on Hot Topics in Operating Systems, pages 142 -147, Rio Rico, Arizona, USA, March 29 - 30, 1999.

[34] G. Tesauro, R. Das, W. E. Walsh, J. O. Kephart. "Utility-Function-Driven Resource Allocation in Autonomic Systems," Proc. Intl. Conf. Autonomic Computing (ICAC 2005), pp.342-343, 13-16 June 2005.

[35] TPC Benchmark C, Standard Specification, Revision 5.9, June 2007. http://www.tpc.org/tpcc/spec/tpcc_current.pdf.

[36] Transaction Processing Performance Council. http://www.tpc.org/tpcc/

[37] B. Violino. "Reducing IT Complexity". United Business Media, 2007. http://www.smartenterprisemag.com/articles/2007winter/coverstory.jhtml.

[38] C. Walton. "FrontRunner Computer Performance Consulting". 2002. http://www.frontrunnercpc.com/index.htm.

[39] W. E. Walsh, G. Tesauro, J. O. Kephart, R. Das. "Utility Functions in Autonomic Systems". In Proc. of Intl. Conf. on Autonomic Computing (ICAC'04), pages 70 - 77, New York, USA, May 17 - 18, 2004.

[40] H. Zawawy, P. Martin, H. Hassanein. "Supporting Capacity Planning for DB2 UDB". Proc., of the 2002 Conf. of the Center for Advanced Studies on Collaborative Research, page 15, Toronto, Canada, September, 2002.

Appendix A
Queueing Network Models

A.1 Introduction

Queueing network modeling is an approach to computer system modeling in which the computer system is represented as a network of queues which is evaluated analytically. A network of queues is a collection of service centers, which represent system resources, and customers, which represent users or transactions. Analytic evaluation involves using software to solve efficiently a set of equations induced by the network of queues and its parameters [21].

In a queueing network model (QNM), service centers can be any active resource such as a CPU, disk, or network link. QNMs can predict the latency, throughput and utilization of a system and its components [38]. In most cases, throughput and utilization are predicted within 10% of measured values, while latency predictions are within 30% [21]. There are several types of QNMs, namely asymptotic models, open network models, and closed network models [38].

Asymptotic models are the simplest application of queuing network models. They are used to predict the best-case and worst-case latency and throughput of a single service center. A typical use of asymptotic bounds is to examine the behavior of a bottleneck resource under various loads. In an open network model, customers pass through a system once. For example, a network device processing a stream of packets would be modeled as an open system. The number of customers in the system is calculated from the customer arrival rate and the characteristics of the service centers. If there are several distinct types of customers, a multi-class open model is used. In a closed network model, the number of customers is fixed. The classic example is a time-sharing computer system with a fixed number of user terminals. In a closed model, the number of

customers is known and the throughput is calculated. An iterative method known as Mean Value Analysis (MVA) is used to calculate throughput and latency [38].

Although queueing network models are a versatile and powerful way to model system performance, they have some limitations. All of the solution techniques require that model meets certain conditions that make it a *separable* model; in some cases this could make the model deviate significantly from the real system [38].

A.2 QNM Service Center

A.2.1 Single Service Center

In a single service center, customers arrive at the service center, wait in the queue if necessary, receive service from the server, and depart. In fact, this service center and its arriving customers constitute a basic queuing network model.

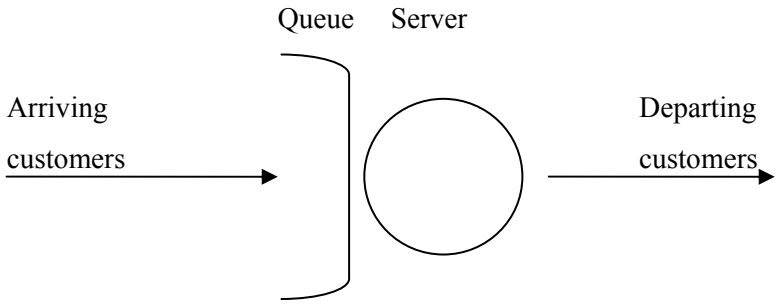

Figure 20: A Single Service Center [21]

A model such as the one in Figure 20 has two parameters: the workload intensity and the service demand. The workload intensity in this case is the rate at which customers arrive (e.g. one customer every two seconds, or 0.5 customers/second). The service demand is the average service requirement of a customer (e.g. 1.25 seconds). For specific parameter values, it is possible to

evaluate this model by solving some simple equations, to yield performance measures such as utilization (the proportion of time the server is busy), residence time (the average time spent at the service center by a customer, both queueing and receiving service), queue length (the average number of customers at the service center, both waiting and receiving service), and throughput (the rate at which customers pass through the service center) [21].

A.2.2 Multiple Service Centers

Characterizing a contemporary computer system by two parameters is an oversimplification of reality. Figure 21 represents a more practical model in which each system resource (in this case a CPU and an array of disks) is represented by a separate service center.

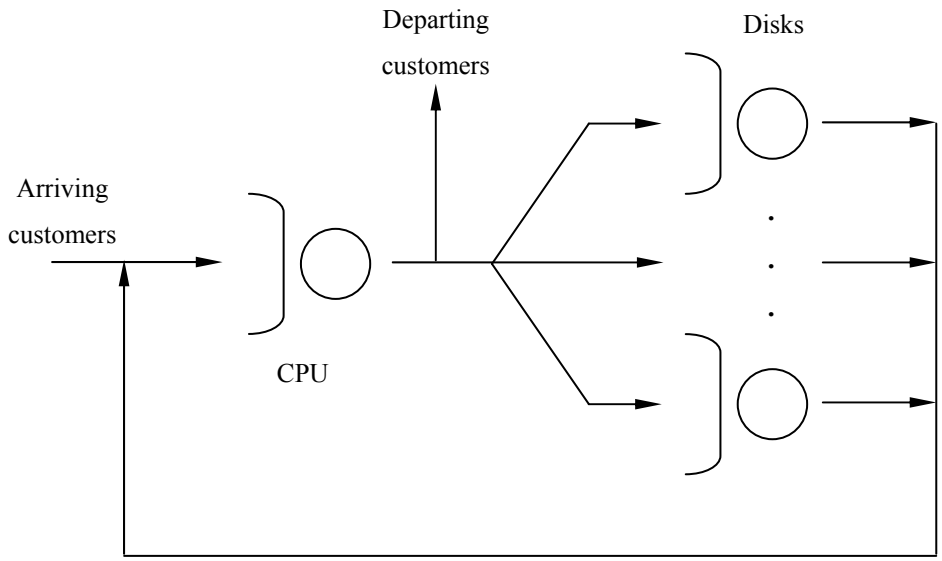

Figure 21: A Network of Queues [21]

The parameters of this model are similar to those of the previous one: the workload intensity, which is the rate at which customers arrive, and the service demand. However, this time a separate service demand is provided for each service center. If the customers in the model

represent transactions in the system, then the workload intensity corresponds to the rate at which users submit these transactions to the system, and the service demand at each service center represents the total service requirement per transaction at the corresponding resource in the system. The customers are arriving, circulating among the service centers, and then leaving the system. The pattern of circulation among the centers is not important, however; only the total service demand at each center matters [21].

A.3 QNM Parameterization and Validation

The parameterization process of queueing network models is relatively straightforward. For example, to calculate the CPU service demand for a customer in a queueing network model of an existing system, we would observe the system in operation and would measure two quantities: the number of seconds that the CPU was busy, and the number of transactions that were processed. Then, we would divide the busy time by the number of transactions completed to determine the average number of seconds of CPU service needed for each transaction to complete.

There are two main approaches to evaluating queueing network models. The first involves calculating bounds on performance measures, rather than specific values. This approach involves determining upper and lower bounds on performance parameter values such as response time for a particular set of inputs (workload intensity and service demands). The virtue of this approach is that the calculations are simple enough to be carried out by hand, and the resulting bounds can contribute significantly to understanding the system under study.

The second approach, called the mean value analysis (MVA), involves calculating the values of the performance measures. While the algorithms for doing this are sufficiently complicated that the use of computer programs is necessary, it is important to emphasize that these algorithms are extremely efficient. Approximate MVA algorithms have been devised in

order to reduce the complexity of calculations while keeping an acceptable accuracy. The running time of the most efficient general algorithm of the MVA grows as the product of the number of service centers with the number of customer classes, and is independent of the number of customers in each class.

The performance of any particular system is determined by carefully analyzing certain performance measures like response times, throughputs and utilization. The response time is the average time interval elapsed between the instant a transaction is submitted to the system for processing until the answer begins to appear at the user's terminal. Throughput is the average number of transactions or jobs executed per unit time. In other words, it is the rate at which requests are serviced. Utilization is the percentage of time the device is being used, during a given time interval. A device is saturated if its utilization approaches 100% [21].

A.4 Fundamental Laws and Mean Value Analysis

A.4.1 Utilization Law

The utilization of a resource is equal to the product of the throughput of that resource and the average service requirement at that resource. The utilization law is stated as:

$$U = X * S$$

where: U is the utilization of the resource, X is the throughput, and S is the average service requirement.

A.4.2 Little's Law

Little's Law states that the average number of requests in a system is equal to the product of the throughput of that system and the average time spent in that system by a request. Little's law is stated as:

$$N = X * R$$

where: N is the average number of requests in the system, X is the throughput, and R is the system residence time per request.

A.4.3 Forced Flow Law

This law is based on the fact that the flows or throughput in all parts of a system must be proportional to each other. The forced flow law is stated as:

$$X_k = V_k * X$$

where: X_k is the throughput at a resource k in the system, V_k is the visit count of the resource k and, X is the throughput of the system.

A.4.4 Mean value analysis

Mean value analysis is the technique most commonly used to resolve closed queueing networks models such as the model used in this thesis. The mean value analysis is based on three equations:

1 – The service center residence time equation:

$$R_k(N) = D_k * (1 + Q_k(N-1))$$

where: $R_k(N)$ is the residence time at center k, D_k is the time needed at resource k for a transaction to be completed, and $Q_k(N-1)$ is the queue length at resource k when there are $N-1$ customers in the system.

2 - Little's Law applied to the queueing network as a whole:

$$X(N) = \frac{N}{Z + \sum_{k=1}^{K} R_k(N)}$$

where: $X(N)$ is the system throughput, $R_k(N)$ is the residence time at center k when there are $\boldsymbol{N}$ customers in the system, and Z is the think time.

3 - Little's Law applied to the service centers individually:

$$Q_k(N) = X(N) * R_k(N)$$

where: $Q_k(N)$ is the queue length at resource k when there are N customers in the system, $X(N)$ is the throughput of the system, and $R_k(N)$ is the residence time at center k.

For the trivial case when the number of request is 1, $N = 1$, there is no queueing at any of the resources, $Q_k(0) = 0$, and the residence time at each of the resources is equal to the service demand from the resource. Therefore, the queue length, when there is one customer in the system, is obtained. Based on this result, the 3 equations are resolved again for the case when $N = 2$, and so on. The exact MVA algorithm is shown in Figure 22 [21].

for $k \leftarrow 1$ to K do $Q_k \leftarrow 0$

for $n \leftarrow 1$ to N do

begin

$$\text{for } k \leftarrow 1 \text{ to } K \text{ do } R_k \leftarrow \begin{cases} D_k & \text{(delay centers)} \\ D_k * (1 + Q_k) & \text{(queuing centers)} \end{cases}$$

$$X \leftarrow \frac{N}{Z + \sum_{k=1}^{K} R_k}$$

for $k \leftarrow 1$ to K do $Q_k \leftarrow X * R_k$

end

Figure 22: Exact MVA Algorithm [21]

Appendix B

TPC-C Benchmark

The Transaction Processing Performance Council (TPC) is a non-profit corporation founded to define transaction processing and database benchmarks and to disseminate objective, verifiable TPC performance data to industry. One of the TPC database benchmarks is the TPC Benchmark™ C (TPC-C), which is an on-line transaction processing (OLTP) benchmark [36].

TPC-C simulates a complete computing environment where a population of users executes transactions against a database. The benchmark is centered on the principal activities (transactions) of an order-entry environment. These transactions include entering and delivering orders, recording payments, checking the status of orders, and monitoring the level of stock at the warehouses. TPC-C involves a mix of five concurrent transactions of different types and complexity either executed on-line or queued for deferred execution. The database is comprised of nine types of tables with a wide range of record and population sizes. TPC-C is measured in transactions per minute (tpmC) [36].

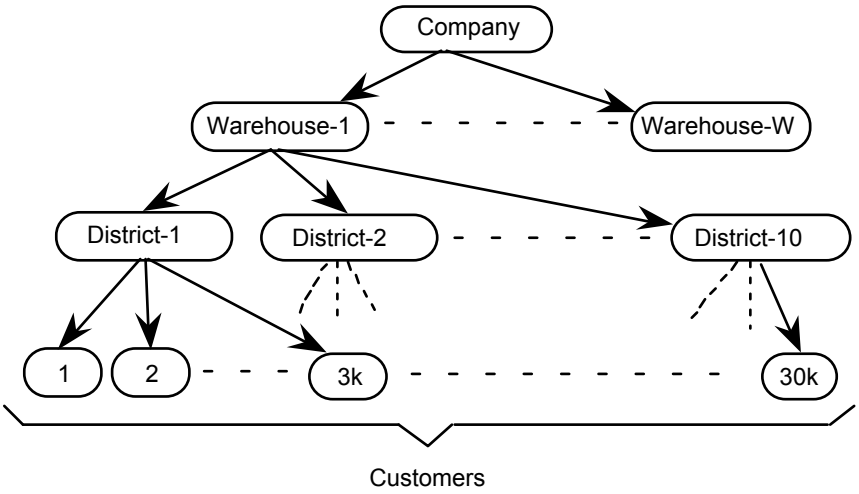

Figure 23: Hierarchy of the Business Environment [35]

The Company portrayed by the benchmark is a wholesale supplier with a number of geographically distributed sales districts and associated warehouses. As the Company's business expands, new warehouses and associated sales districts are created. Each regional warehouse covers 10 districts. Each district serves 3,000 customers. All warehouses maintain stocks for the 100,000 items sold by the Company. Figure 23 illustrates the warehouse, district, and customer hierarchy of TPC-C's business environment [35].

The components of the TPC-C database are defined to consist of nine separate and individual tables. The relationships among these tables are defined in the entity-relationship diagram shown in Figure 24.

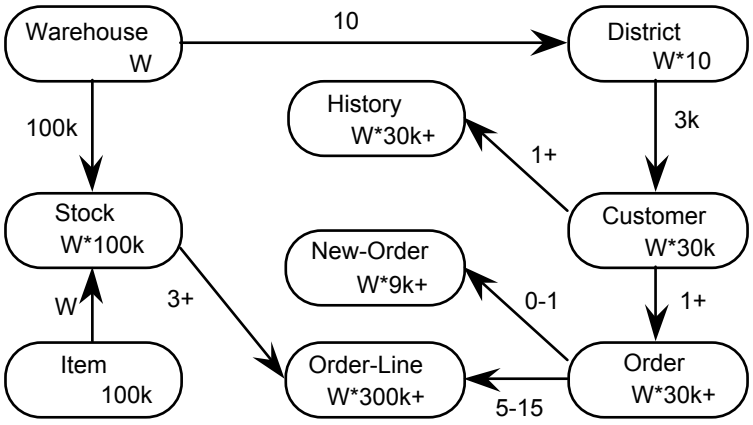

Figure 24: Database Schema for TPC-C Benchmark [35]

In Figure 24, all numbers shown illustrate the database population requirements. The numbers in the entity blocks represent the cardinality of the tables (number of rows). These numbers are factored by W, the number of Warehouses, to illustrate the database scaling. The numbers next to the relationship arrows represent the cardinality of the relationships (average number of children per parent). The plus (+) symbol is used after the cardinality of a relationship

or table to illustrate that this number is subject to small variations in the initial database population over the measurement interval as rows are added or deleted. The nine individual tables are defined as the following:

Field Name	Field Definition	Comments
W_ID	2*W unique IDs	*W Warehouses are populated*
W_NAME	variable text, size 10	
W_STREET_1	variable text, size 20	
W_STREET_2	variable text, size 20	
W_CITY	variable text, size 20	
W_STATE	fixed text, size 2	
W_ZIP	fixed text, size 9	
W_TAX	signed numeric(4,4)	*Sales tax*
W_YTD	signed numeric(12,2)	*Year to date balance*
Primary Key: W_ID		

Table 1: WAREHOUSE Table Layout [35]

Field Name	Field Definition	Comments
D_ID	20 unique IDs	*10 are populated per warehouse*
D_W_ID	2*W unique IDs	
D_NAME	variable text, size 10	
D_STREET_1	variable text, size 20	
D_STREET_2	variable text, size 20	
D_CITY	variable text, size 20	
D_STATE	fixed text, size 2	
D_ZIP	fixed text, size 9	
D_TAX	signed numeric(4,4)	*Sales tax*
D_YTD	signed numeric(12,2)	*Year to date balance*
D_NEXT_O_ID	10,000,000 unique IDs	*Next available Order number*
Primary Key: (D_W_ID, D_ID) D_W_ID Foreign Key, references W_ID		

Table 2: DISTRICT Table Layout [35]

Field Name	Field Definition	Comments
C_ID	96,000 unique IDs	*3,000 are populated per district*
C_D_ID	20 unique IDs	
C_W_ID	2*W unique IDs	
C_FIRST	variable text, size 16	
C_MIDDLE	fixed text, size 2	
C_LAST	variable text, size 16	
C_STREET_1	variable text, size 20	
C_STREET_2	variable text, size 20	
C_CITY	variable text, size 20	
C_STATE	fixed text, size 2	
C_ZIP	fixed text, size 9	
C_PHONE	fixed text, size 16	
C_SINCE	date and time	
C_CREDIT	fixed text, size 2	*"GC"=good, "BC"=bad*
C_CREDIT_LIM	signed numeric(12, 2)	
C_DISCOUNT	signed numeric(4, 4)	
C_BALANCE	signed numeric(12, 2)	
C_YTD_PAYMENT	signed numeric(12, 2)	
C_PAYMENT_CNT	numeric(4)	
C_DELIVERY_CNT	numeric(4)	
C_DATA	variable text, size 500	*Miscellaneous information*
Primary Key: (C_W_ID, C_D_ID, C_ID) (C_W_ID, C_D_ID) Foreign Key, references (D_W_ID, D_ID)		

Table 3: CUSTOMER Table Layout [35]

Field Name	Field Definition	Comments
H_C_ID	96,000 unique IDs	
H_C_D_ID	20 unique IDs	
H_C_W_ID	2*W unique IDs	
H_D_ID	20 unique IDs	
H_W_ID	2*W unique IDs	
H_DATE	date and time	
H_AMOUNT	signed numeric(6, 2)	
H_DATA	variable text, size 24	*Miscellaneous information*
Primary Key: none (H_C_W_ID, H_C_D_ID, H_C_ID) Foreign Key, references (C_W_ID, C_D_ID, C_ID) (H_W_ID, H_D_ID) Foreign Key, references (D_W_ID, D_ID)		

Table 4: HISTORY Table Layout [35]

Field Name	Field Definition	Comments
NO_O_ID	10,000,000 unique IDs	
NO_D_ID	20 unique IDs	
NO_W_ID	2*W unique IDs	
Primary Key: (NO_W_ID, NO_D_ID, NO_O_ID) (NO_W_ID, NO_D_ID, NO_O_ID) Foreign Key, references (O_W_ID, O_D_ID, O_ID)		

Table 5: NEW-ORDER Table Layout [35]

Field Name	Field Definition	Comments
O_ID	10,000,000 unique IDs	
O_D_ID	20 unique IDs	
O_W_ID	2*W unique IDs	
O_C_ID	96,000 unique IDs	
O_ENTRY_D	date and time	
O_CARRIER_ID	10 unique IDs, or null	
O_OL_CNT	numeric(2)	*Count of Order-Lines*
O_ALL_LOCAL	numeric(1)	
Primary Key: (O_W_ID, O_D_ID, O_ID) (O_W_ID, O_D_ID, O_C_ID) Foreign Key, references (C_W_ID, C_D_ID, C_ID)		

Table 6: ORDER Table Layout [35]

Field Name	Field Definition	Comments
OL_O_ID	10,000,000 unique IDs	
OL_D_ID	20 unique IDs	
OL_W_ID	2*W unique IDs	
OL_NUMBER	15 unique IDs	
OL_I_ID	200,000 unique IDs	
OL_SUPPLY_W_ID	2*W unique IDs	
OL_DELIVERY_D	date and time, or null	
OL_QUANTITY	numeric(2)	
OL_AMOUNT	signed numeric(6, 2)	
OL_DIST_INFO	fixed text, size 24	
Primary Key: (OL_W_ID, OL_D_ID, OL_O_ID, OL_NUMBER) (OL_W_ID, OL_D_ID, OL_O_ID) Foreign Key, references (O_W_ID, O_D_ID, O_ID) (OL_SUPPLY_W_ID, OL_I_ID) Foreign Key, references (S_W_ID, S_I_ID)		

Table 7: ORDER-LINE Table Layout [35]

Field Name	Field Definition	Comments
I_ID	200,000 unique IDs	*100,000 items are populated*
I_IM_ID	200,000 unique IDs	*Image ID associated to Item*
I_NAME	variable text, size 24	
I_PRICE	numeric(5, 2)	
I_DATA	variable text, size 50	*Brand information*
Primary Key: I_ID		

Table 8: ITEM Table Layout [35]

Field Name	Field Definition	Comments
S_I_ID	200,000 unique IDs	*100,000 populated per warehouse*
S_W_ID	2*W unique IDs	
S_QUANTITY	signed numeric(4)	
S_DIST_01	fixed text, size 24	
S_DIST_02	fixed text, size 24	
S_DIST_03	fixed text, size 24	
S_DIST_04	fixed text, size 24	
S_DIST_05	fixed text, size 24	
S_DIST_06	fixed text, size 24	
S_DIST_07	fixed text, size 24	
S_DIST_08	fixed text, size 24	
S_DIST_09	fixed text, size 24	
S_DIST_10	fixed text, size 24	
S_YTD	numeric(8)	
S_ORDER_CNT	numeric(4)	
S_REMOTE_CNT	numeric(4)	
S_DATA	variable text, size 50	*Make information*
Primary Key: (S_W_ID, S_I_ID) S_W_ID Foreign Key, references W_ID S_I_ID Foreign Key, references I_ID		

Table 9: STOCK Table Layout [35]

The mixed five different concurrent transactions of TPC-C as follow [35]:

- **New-Order**: consists of entering a complete order through a single database transaction. It represents a mid-weight, read-write transaction with a high frequency of execution and stringent response time requirements to satisfy on-line users. This transaction is the backbone of the workload. It is designed to place a variable load on the system to reflect on-line database activity as typically found in production environments.
- **Payment**: updates the customer's balance and reflects the payment on the district and warehouse sales statistics. It represents a light-weight, read-write transaction with a high frequency of execution and stringent response time requirements to satisfy on-line users. In addition, this transaction includes non-primary key access to the CUSTOMER table.
- **Order-Status**: queries the status of a customer's last order. It represents a mid-weight read-only database transaction with a low frequency of execution and response time requirement to satisfy on-line users.
- **Delivery**: consists of processing a batch of 10 new (not yet delivered) orders. Each order is processed (delivered) in full within the scope of a read-write database transaction. The number of orders delivered as a group (or batched) within the same database transaction is implementation specific. The business transaction comprised of one or more (up to 10) database transactions, has a low frequency of execution and complete within a relaxed response time requirement. The Delivery transaction is intended to be executed in deferred mode through a queuing mechanism, rather than interactively, with terminal response indicating transaction completion.

- **Stock-Level**: determines the number of recently sold items that have a stock level below a specified threshold. It represents a heavy read-only database transaction with a low frequency of execution, a relaxed response time requirement, and relaxed consistency requirements.

Table 10 lists each transaction type is available to a TPC-C workload. Over a run, the TPC-C workload maintains a percentage of mix for each transaction type as follows:

Transaction Type	Minimum % of mix
New-Order	45.0
Payment	43.0
Order-Status	4.0
Delivery	4.0
Stock-Level	4.0

Table 10: TPC-C Transaction Percentage

Appendix C

Economic Model Simulator

We extend the economic model simulator created by Boughton [7] to validate our approaches for the CPU and multiple resource allocations respectively. The simulators were created using JAVA™ programming system and implemented as shown in the UML class diagrams presented in Figure 25 and Figure 26. The Simulator class contains the main method in each of the diagrams and controls the simulator.

Figure 25: UML Class Diagram for CPU Resource Allocation Simulator

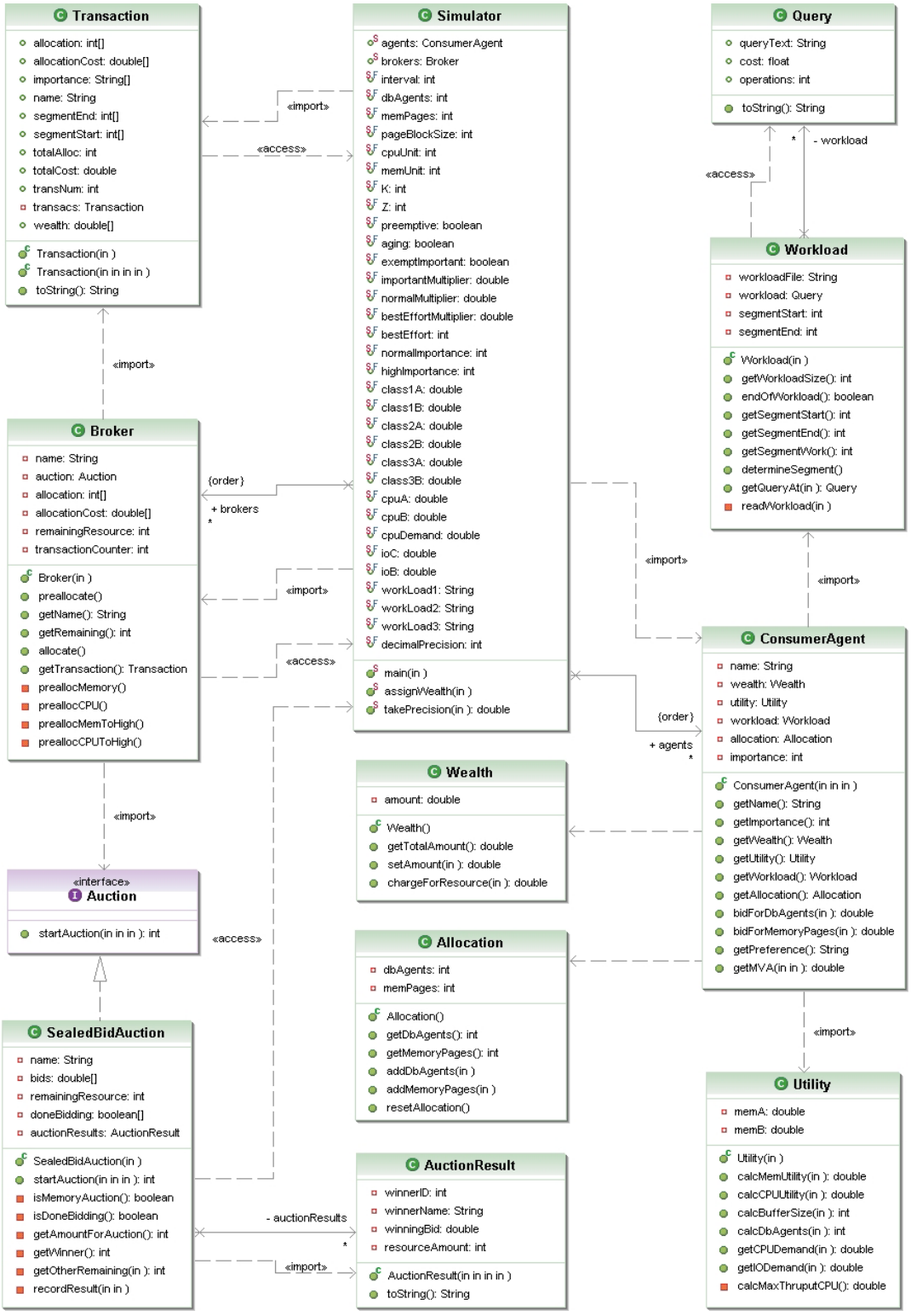

Figure 26: UML Class Diagram for Multiple-Resource Allocation Simulator