ELSEVIER

Decision Support Systems 36 (2004) 297–312

Decision Support Systems

www.elsevier.com/locate/dsw

An analysis of the optimal number of servers in distributed client/server environments

Jin Hyun Son [a], Myoung Ho Kim [b,*]

[a] *Department of Computer Science and Engineering Hanyang University, 1271 Sa-1dong, Ansan, Kyunggi-do 425-791, South Korea*

[b] *Division of Computer Science, Korea Advanced Institute of Science and Technology, KAIST, 373-1 Kusung-dong, Yusung-gu, Taejon 305-701, South Korea*

Accepted 1 July 2002

Abstract

In the client/server model for distributed on-line requests processing, the concept of a server class that consists of multiple identical servers is often provided for the fast response time, high fault-tolerance, or continuous availability. As far as we are aware of, there is no concrete formulation that provides the optimal number of servers of a server class in distributed client/server systems. The number of servers has been usually chosen only by heuristic methods or statistics after monitoring system behavior for some periods of time.

In this paper, we propose a method based on queuing theory that determines the optimal number of servers in a server class for a general distributed client/server model that encompasses the state-of-the-art on-line transaction processing environment described in the literature.

Keywords: Optimal number of servers; Server class; Distributed client/server model; Queuing model

1. Introduction

A distributed client/server computing system generally has many advantages over a single powerful mainframe, such as improved availability, extensibility, and overall performance. To realize these advantages, efficient mechanisms for the system-wide resource management are needed. There are two performance perspectives that must be addressed in this regard. From the client's perspective, performance is measured by the server's average response time. On the other hand, the system throughput and the server (or resource) utilization are also important factors to be considered for the system performance. In general, it is known that the average response time of a server increases dramatically as its utilization increases [10]. To satisfy these conflicting requirements, many heuristic methods and stochastic approaches have been extensively studied so far [8,20,21].

In the client/server computing, clients ask for servers' services and servers provide their own services. For a high degree of concurrency, an application server is often replicated into a set of identical servers, called a server class, which may be placed throughout the distributed client/server system. A scheduler in

* Corresponding author. Tel.: +82-42-869-3530; fax: +82-42-869-3510.

E-mail addresses: jhson@dbserver.kaist.ac.kr (J.H. Son), mhkim@dbserver.kaist.ac.kr (M.H. Kim).

0167-9236/$ - see front matter © 2002 Elsevier Science B.V. All rights reserved.
doi:10.1016/S0167-9236(02)00142-2

this environment plays a role of a broker between clients and servers for effective three-tier client/server architectures. Service requests of clients are initially connected to a scheduler that selects an appropriate server in a server class according to the scheduling policy and also mediates the connections between clients and servers. The choice of a server should be made in such a way that overall requirements can be optimally achieved, that is, minimizing the response time while maximizing the system throughput.

There are many considerations in configuring the distributed client/server systems. Both the optimal resource placement and minimizing communication delay between hosts are typical and general issues. In addition, with the concept of a server class that is introduced in most modern on-line transaction processing systems, the optimal number of servers for each server class must be decided. If we have too small number of servers for a certain service, the number of clients waiting for the service increases exponentially, which results in long response times. Some clients may also be rejected with denial of services due to the limitation of system resources. On the other hand, if there are too many servers for a certain service, several servers may be in the idle state. This causes the waste of system resources and hence decreases the overall system throughput. As far as we are aware of, there is no concrete formulation that analyzes the optimal number of servers of a server class in distributed client/server systems. Until now, the number of servers has been usually chosen by heuristics or statistics after monitoring system behavior over some periods of time [10,16].

Even though a single multi-threaded server with a large number of threads may process a high volume of arrival requests, adopting a server class with multiple identical servers may be preferred in many cases due to fault-tolerance and availability issues in the distributed client/server computing system. Therefore, our main contribution in this paper is to propose a systematic and formal method that provides the optimal number of servers in a server class, based on the number of hosts and the effective number of threads the system can support. We first describe a general distributed client/server model that may encompass the state-of-the-art on-line transaction processing environments. And then, we develop a transformation mechanism from an open Jackson network to a simple tandem network because our distributed client/server computing model generally forms an open Jackson network that is known to be difficult to analyze. We also provide discussion and in-depth analyses of different aspects of the issue discussed in this paper, based on various experiments.

The remainder of the paper is organized as follows: In Section 2, we describe, as backgrounds, various client/server computing structures and previous research on queuing networks. Section 3 presents the distributed client/server processing model considered in this paper. We then analyze the optimal number of servers based on queuing theory in Section 4. Section 5 gives several experimental results and some considerations, and Section 6 finally concludes by summarizing the results and contributions of this paper along with directions for further research.

2. Related work

2.1. Client/server process structure

There can be various schemes to map client requests to servers. The mapping schemes are usually classified according to the multiplicity of identical servers and the relationships between schedulers and servers [10]. Fig. 1 depicts four mapping schemes. Note that there can be other variants, but those described below are the most representative of the commercial products now in use.

(a) One server per client: This is a one-to-one mapping architecture, which means that each client process is tightly coupled with one server process. This scheme is not only inflexible in load balancing, but also may generate too many server processes causing serious overhead in the system.

(b) One server, one scheduler: There is only a single process that provides all the services for clients and also has a scheduling function. While the scheduler in this structure can easily control the requests of clients, its failure can cause overall services to be unavailable. Because one server process is confined within one address space and one operating system process, this architecture is not appropriate either for multiprocessor systems that are very common these days.

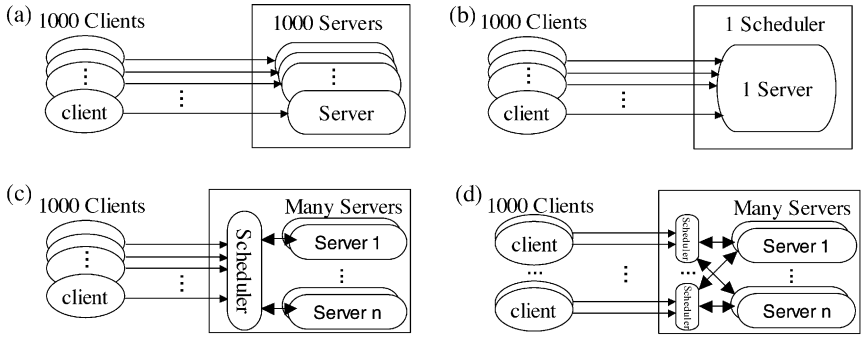

Fig. 1. Four different client/server process structures. (a) One server per client; (b) one server, one scheduler; (c) multiple server classes, one scheduler; (d) multiple server classes, multiple schedulers.

(c) Multiple server classes, one scheduler: There is a server class for each application. Each server class consists of one or more identical servers. As in (b), one scheduler on behalf of multiple server classes accepts all client requests and connects each request to a proper server process. A scheduler provides load balancing between servers in a server class. This single scheduler, however, may be bottlenecked for a high volume of client requests.

(d) Multiple server classes, multiple schedulers: Multiple schedulers simultaneously mediate the connections between clients and servers. This scheme is the most general among four schemes and hence will be the basis of our work.

Most products developed so far may belong to one of the four mapping schemes mentioned above. The first structure is the basic client/server model. IBM's CICS is based on the second one, and OSF's DCE and BEA's Tuxedo are based on the third. Tandem's Pathway and DEC's ACMS are based on the fourth. Nowadays, the client/server model tends to evolve from structures (a), (b) to (c), (d) in Fig. 1 to improve flexibility and incorporate new paradigms.

2.2. Scheduling based on queuing networks

A network of queues can model problems of contention that arise when a set of resources is shared. There are two types of queuing networks: open and closed. In a queuing network described as a group of nodes, when customers may arrive from outside the system to any node and may depart from the system from any node, we call it as an open network. When no customer may enter the system from the outside and may leave the system, it is referred to as a closed network. A tandem network, that is, a serial network without feedback, is a special case of open networks. It has been widely used for the performance evaluation of computer systems such as a virtual memory operating system and a node-to-node protocol in a computer network.

The relationship between input and output distribution of a queuing system and waiting times in a tandem network were studied in Refs. [1,2]. In Ref. [12], Jackson defined an open(closed) Jackson network. Reich [18,19] investigated the distribution of waiting times in two stage tandem queues and the correlation between job ordering in a queuing system. Maximum throughput in a tandem network with blocking due to a finite waiting room was studied by an approximation approach in Ref. [4]. Perros and Altiok [17] developed an approximation procedure for the analyses of tandem configurations and Gershwin [9] presented an efficient decomposition method that evaluates performance measures for a class of tandem queuing systems with finite buffers.

Scheduling is concerned with the allocation of limited resources to tasks, and various scheduling mechanisms have been studied according to system environments. Casey [5] set the basis of a hierarchical categorization of the scheduling problem, but did not include a large number of fundamental distinguishing features to differentiate between existing approaches.

Wang and Morris [23] provided a taxonomy of load-sharing schemes. After that, Casavant and Kuhl [3] classified most of the scheduling approaches, including categorizations of Casey [5] and Wang and Morris [23].

Lui et al. [15] modeled multiprocessors and multicomputer systems as heterogeneous multi-server queuing systems and investigated the performance of the systems operating under the minimum expected delay routing policy, that is, an arriving client is assigned to the queue that has the minimal expected value of unfinished work. On the other hand, the optimality of the join-the-shortest-queue policy for homogeneous multi-server systems has been established in Ref. [22]. Task-execution times are one of the most important parameters in scheduling. In line with this, Chu and Sit [6] used a Timed Petri-Net to model resource contention and a queuing model to derive the execution time of a task. And, Hou and Shin [11] proposed a new load sharing algorithm for real-time applications that takes into account the effect of future task arrivals on locating the best receiver for each unguaranteed task.

As mentioned in Section 2.1, because the structures of multiple server classes–one scheduler (or multiple schedulers) are nowadays mainstreams in the client/server model, we consider a general distributed client/server processing model to encompass these structures. And, we develop a concrete formulation that can decide the optimal number of servers in a server class. On the other hand, most previous work has concentrated on the optimal scheduling and load balancing policies of a scheduler to improve performance, assuming that the number of servers is fixed and cannot be changed [5,8,11,15,23,25].

3. The model under study

3.1. Client/server system model

A distributed client/server processing model considered in this paper is described in Fig. 2. This encompasses the structures of multiple server classes–one scheduler and multiple server classes–multiple schedulers mentioned in Section 2.1. Each queuing system in Fig. 2 corresponds to one multi-

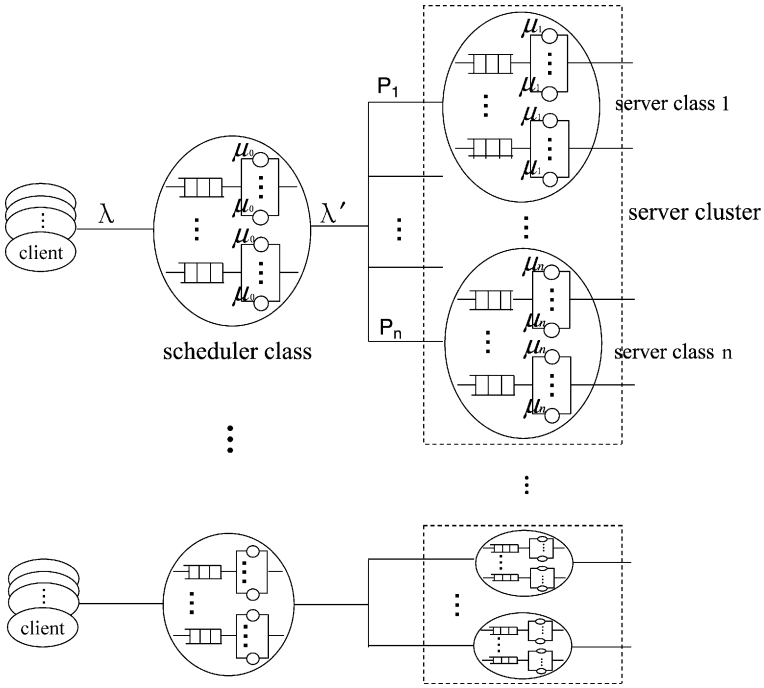

Fig. 2. Scheduling model under study.

threaded scheduler or server process. Some terms related with the model are defined as follows:

Definition 3.1. The set of identical schedulers that schedule clients' service requests is called a *scheduler class*.

Definition 3.2. The set of identical servers that are scheduled by the same scheduler class is called a *server class*.

Definition 3.3. The set of server classes that are scheduled by the same scheduler class is called a *server cluster*. A server cluster consists of one or more server classes.

We also make some assumptions regarding the model as shown in Fig. 2.

Assumption 1. Each scheduler in a scheduler class has the same load, arrival rate, scheduling policy, scheduling time (i.e., scheduling rate), and number of threads.

Assumption 2. Each server in a server class has the same load, arrival rate, services, service time (i.e., service rate), and number of threads.

Assumption 3. Queue discipline is first come–first served (FCFS), that is, clients are served in the same order in which they arrive.

Assumption 4. Once a client request enters a waiting queue of the scheduler, it is guaranteed for the request to be connected to a server.

The behavior of computer processes can be meaningfully modeled by the Poisson process and the exponential distribution. An important application of the Poisson process arises in connection with the occurrences of events of a particular type over time. The exponential distribution is frequently used as a model for the distribution of times between the occurrence of successive events such as customers arriving at a service facility. These characteristics have made the Poisson process and the exponential distribution applied to many areas of computer engineering [13]. Since the events to occur in the distributed client/server model considered in this paper may match these characteristics very well, we assume that the client requests arrive by a Poisson process and each scheduler or server has exponential service time with mean rate μ_i.

All the schedulers and servers in the same class have the same mean service rate from Assumptions 1 and 2, respectively. And, client requests arriving at the scheduler class with mean arrival rate λ are considered to be evenly distributed to each scheduler from Assumption 1. Because some client requests may be rejected by a scheduler due to the system constraints such as physical limitation of its waiting queue, the mean departure rate λ' from the scheduler class is less than or equal to its arrival rate λ. We can often notice this kind of rejection of requests from the web server, ftp, banking server, etc. When the probability for a client request to ask for the service of server class i is P_i, the arrival rate of the client requests forwarded to server class i is $\lambda' P_i$. The scheduler may evenly distribute client requests for the services of server class i into servers of server class i as in Assumption 2. Note that the problem of evenly distributing client requests to schedulers and servers for load balancing is beyond the scope of this paper. Some simple heuristics may be based on round robin or random distribution. More sophisticated methods may utilize information obtained by monitoring the resource usage and workloads of servers [14]. Assumption 3 implies that all client requests have the same execution priorities. Under Assumption 4, once a client request enters a waiting queue of a scheduler, it is guaranteed to be allocated to a server by the proper functions of the scheduler. That is, a client request can only be rejected by a scheduler, not by a server. This can simply be achieved by delaying some client requests at the scheduler.

In the general three-tier client/server model as in Fig. 2, a scheduler usually performs somewhat simple functions such as scheduling and authentication control. On the other hand, a server provides complex application services including the use of system resources, especially database accesses. Therefore, both the response time (from users' perspectives) and the system throughput (or utilization) are primarily affected by the server configuration. In this regard, our main concern is how to find the optimal number of servers in each server class, which is known to be difficult to determine as indicated in the literature [10].

3.2. M/M/c model for a scheduler and a server

A scheduler and a server are commonly multithreaded processes in modern client/server computing

systems. A multi-threaded process has one arrival entrance, one finite waiting queue, and one departure exit, and can execute multiple jobs simultaneously at a time. Hence, it can be modeled by a M/M/c/k queuing system that means c threads and a limited waiting queue. The k denotes the maximum number of client requests that can exist in the system, that is, the number of client requests in the waiting queue plus the number of client requests being served. In general, the reason to restrict the size of a waiting queue is to keep the client response time within the upper bound rather than due to the physical limitation of system resources.

By the Markovian birth–death process of a M/M/c/k model, we can observe the following result.

Observation 3.1. When p_k is the probability of state k in the M/M/c/k queuing system with mean arrival rate λ and maximum service rate $c \times \mu$, the expected number of rejected clients per unit time is λp_k.

Burke [2] proved that "In a M/M/c with an infinite queue, the distribution of departure rate of a system is the same with that of mean arrival rate of the system." This means that if all the clients arriving at a system are accepted, departure rate of the system is identical to its arrival rate. Because the expected number of rejected clients in a M/M/c/k with arrival rate λ is λp_k from Observation 3.1, the average accepted arrival rate of the M/M/c/k is $\lambda(1-p_k)$. Hence, we can approximate a M/M/c/k with mean arrival rate λ to a M/M/c with mean arrival rate $\lambda(1-p_k)$ within a negligible error. Table 1 shows that the difference of the average waiting times between M/M/c/k and M/M/c systems is negligible, which indicates that the approximation works well. Therefore, each scheduler in Fig. 2 can be modeled with a M/M/c queuing system. And, each server in a server class can also be mapped into a M/M/c queuing model because no client's request is rejected at a server from Assumption 4. From now on, the mean arrival(departure) rate and the mean service rate of a queuing system in this paper are called simply the arrival(departure) rate and the service rate if there is no ambiguity.

4. The analysis of the optimal number of servers

4.1. Transformation of an open Jackson network into a tandem network

The connection between a scheduler class and a server class in the model of Fig. 2 is under the open Jackson network. Since the open Jackson network is known to be difficult to analyze [17], we transform the open Jackson network between a scheduler class and a server class (Fig. 3(a)) into a serial two-stage tandem network (Fig. 3(b)). Though the transformation has some errors, we will show that these errors are quite small enough to be ignored in practice. In the two-stage tandem network, we can easily estimate the optimal number of servers in a server class.

Because the client requests arriving at a scheduler class with rate λ is rejected by the probability p_k as in Fig. 3, the accepted rate $\lambda(1-p_k)$ is the actual arrival rate to a scheduler class in which each scheduler is modeled with a M/M/c. Hence, the departure rate λ' of a scheduler class is also $\lambda(1-p_k)$. And then, the departure rate λ' is distributed to n server classes with probability P_i $(1 \le i \le n)$. If there are Y number of schedulers(servers) in the scheduler(server) class,

Table 1
Comparison between M/M/c/k and M/M/c

Queue size	The average waiting time (W) of M/M/c/k	The average waiting time (W') of M/M/c	$\lvert W - W' \rvert$
10	$6.9902451208024602 \times 10^{-2}$	$7.0133724475990630 \times 10^{-2}$	2.31×10^{-4}
30	$7.0158106959149341 \times 10^{-2}$	$7.0158297740533782 \times 10^{-2}$	1.91×10^{-7}
50	$7.0158305045320959 \times 10^{-2}$	$7.0158305139201266 \times 10^{-2}$	9.38×10^{-11}
70	$7.0158305141387045 \times 10^{-2}$	$7.0158305141426264 \times 10^{-2}$	3.92×10^{-14}
90	$7.0158305141426916 \times 10^{-2}$	$7.0158305141426930 \times 10^{-2}$	1.38×10^{-17}
100	$7.0158305141426930 \times 10^{-2}$	$7.0158305141426930 \times 10^{-2}$	≈ 0

Arrival rate = 100, the number of threads = 10, service rate = 15/s.

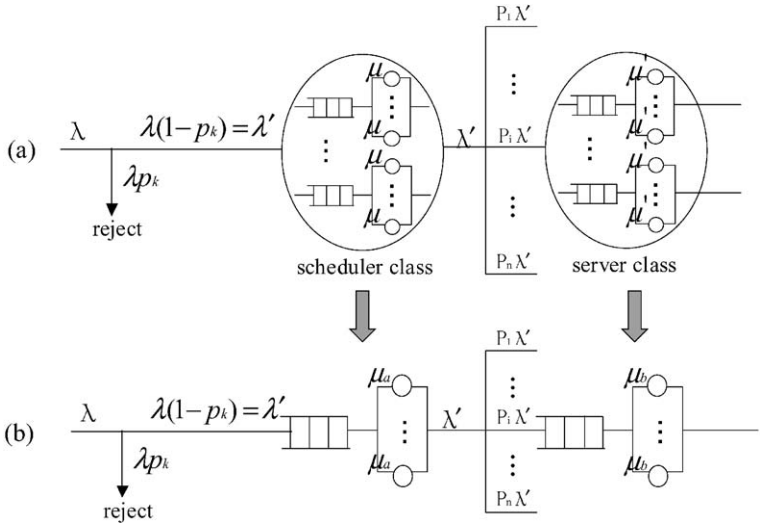

Fig. 3. Transformation from open Jackson network to tandem network.

each scheduler(server) can be approximated as a M/M/c with arrival rate λ/Y as mentioned in Section 3.2, when λ is the actual arrival rate of a scheduler(server) class (e.g., λ becomes λ' for the scheduler class and P_i λ' for server class i in Fig. 3(a)).

The transformation from the open Jackson network between a scheduler class and a server class into a two-stage tandem network includes the process to map a set of M/M/c queuing systems into a single M/M/c' queuing system as in Fig. 3. This process is essentially based on the transformation between M/M/c and M/M/1 as depicted in Fig. 4. When a scheduler class(or server class) consists of Y number of M/M/c queuing systems (Fig. 4(a)), each M/M/c is first transformed into c number of M/M/1 queues, resulting in $c \times Y$ number of M/M/1 like Fig. 4(b). And then,

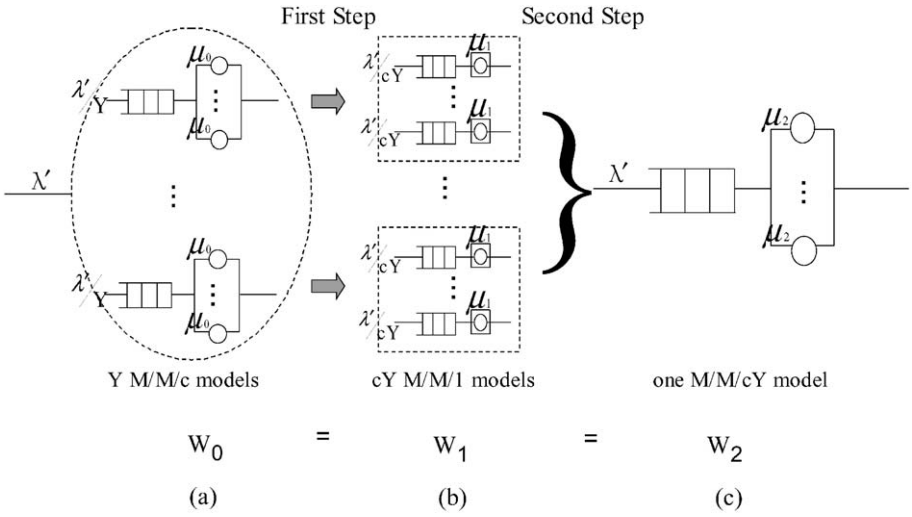

Fig. 4. Transformation from Y M/M/c to one M/M/cY.

these $c \times Y$ M/M/1 queues are finally transformed into one M/M/cY queuing system like Fig. 4(c). Note that the above transformations are performed for each scheduler(server) class independently. After performing these transformations, we have a two-stage tandem network as in Fig. 3(b) for each server class.

In summary, the transformation from a set of M/M/c into a single M/M/c' consists of two sequential steps: transforming a M/M/c into c number of M/M/1 (i.e., from Fig. 4(a) to (b)), and then c' number of M/M/1 into a M/M/c' (i.e., from Fig. 4(b) to (c)). During these two steps, we should make the average waiting times equal as in $W_0 = W_1 = W_2$ of Fig. 4. Suppose W is the average waiting time of a M/M/c with arrival rate λ and service rate μ. Suppose also that W' is the average waiting time of c M/M/1 queues with arrival rate λ/c and service rate μ' for each M/M/1. Note that either the number of channels or service rate of the target queuing system in the transformations must be adjusted to keep the average waiting time unchanged. We will preserve the equality $W = W'$ by adjusting the service rate of the target queuing system in each transformation because the change of service rate rather than the number of channels that is integer is more accurate.

It is well known in the literature [24] that the average waiting time W of a M/M/c with arrival rate λ and service rate μ is

$$W = \frac{1}{\mu} + \frac{\left(\frac{\lambda}{\mu}\right)^c \mu}{(c-1)!(c\mu-\lambda)^2} \times \left[\sum_{n=0}^{c-1} \frac{1}{n!}\left(\frac{\lambda}{\mu}\right)^n + \frac{1}{c!}\left(\frac{\lambda}{\mu}\right)^c \left(\frac{c\mu}{c\mu-\lambda}\right)\right]^{-1}$$

and the average waiting time W' of a M/M/1 with arrival rate λ' and service rate μ' is $W' = 1/(\mu' - \lambda')$, which will be used in the transformations. In the following, we make a detailed explanation on the two transformations that are proceeded in order.

Transformation 1: From one M/M/c to cM/M/1 queues

Suppose the average waiting time W of a M/M/c with given arrival rate λ and service rate μ is w that is already known as mentioned above. In this transformation, because the total arrival rate of c M/M/1 is equal to that of one M/M/c, we can compute service rate μ' of each M/M/1 (i.e., the target queuing system) with arrival rate λ/c from the equation $W = W'$, where W' is the average waiting time of the M/M/1.

$$w = W' = \frac{1}{\left(\mu' - \frac{\lambda}{c}\right)}$$

Hence,

$$\mu' = \frac{1}{w} + \frac{\lambda}{c} \tag{1}$$

This transformation is applied to the First Step between Fig. 4(a) and (b).

Transformation 2: From c M/M/1 queues to one M/M/c

Suppose the average waiting time W' of each M/M/1 with given arrival rate λ/c and service rate μ' is w' that is already known as $w' = 1/(\mu' - (\lambda/c))$. In this transformation, because the arrival rate of one M/M/c is equal to the total arrival rate of c M/M/1, we can compute service rate μ of the M/M/c (i.e., the target queuing system) with arrival rate λ from the equation $W' = W$, where W is the average waiting time of the M/M/c.

$$w' = W = \frac{1}{\mu} + \frac{\left(\frac{\lambda}{\mu}\right)^c \mu}{(c-1)!(c\mu-\lambda)^2} \times \left[\sum_{n=0}^{c-1} \frac{1}{n!}\left(\frac{\lambda}{\mu}\right)^n + \frac{1}{c!}\left(\frac{\lambda}{\mu}\right)^c \left(\frac{c\mu}{c\mu-\lambda}\right)\right]^{-1} \tag{2}$$

In the above equation,

$$\sum_{n=0}^{c-1} \frac{1}{n!}\left(\frac{\lambda}{\mu}\right)^n = 1 + \left(\frac{\lambda}{\mu}\right) + \frac{1}{2!}\left(\frac{\lambda}{\mu}\right)^2 + \frac{1}{3!}\left(\frac{\lambda}{\mu}\right)^3 + \ldots + \frac{1}{(c-1)!}\left(\frac{\lambda}{\mu}\right)^{c-1}$$

From Taylor Series,

$$e^x = 1 + x + \frac{1}{2!}x^2 + \frac{1}{3!}x^3 + \ldots + \frac{1}{n!}x^n + R_n(x,0)$$

Here, n corresponds to the number of channels c in a M/M/c. When $n = 9$, an error $R_n(x,0)$ becomes less

than 10^{-6}. Since computer processes for distributed on-line client/server processing such as banking or stock trading applications commonly have large number of threads (more than 9) that is denoted by c in a M/M/c queuing system, we can have the approximation $\sum_{n=0}^{c-1}\frac{1}{n!}\left(\frac{\lambda}{\mu}\right)^n \approx e^{\frac{\lambda}{\mu}}$ within a negligible error. Therefore, Eq. (2) can be rewritten as

$$w' = \frac{1}{\mu} + \frac{\left(\frac{\lambda}{\mu}\right)^c \mu}{(c-1)!(c\mu-\lambda)^2} \times \left[e^{\frac{\lambda}{\mu}} + \frac{1}{c!}\left(\frac{\lambda}{\mu}\right)^c \left(\frac{c\mu}{c\mu-\lambda}\right)\right]^{-1} \quad (3)$$

From the equation, we can obtain the unique service rate μ of a M/M/c, which is proven in Theorem 4.1. This transformation is applied to the Second Step between Fig. 4(b) and (c).

Theorem 4.1. *In our two transformations between c M/M/1 queuing systems and one M/M/c queuing system, the service rate of the target queuing system in each transformation is unique.*

Proof. First, the transformation from one M/M/c to c M/M/1 clearly produces a unique root by Eq. (1). For the transformation from c M/M/1 to one M/M/c, the uniqueness of μ is equal to that of $x(=\lambda/\mu)$ $(0<x<c)$, where μ is the service rate of each channel in the M/M/c. Using x, we can rewrite Eq. (3) as

$$w' = \frac{x}{\lambda} + \frac{x^c}{(c-1)!\lambda x(c-x)^2}\left[e^x + \frac{1}{c!}x^c\frac{c}{c-x}\right]^{-1}$$

After rearranging the above equation into a higher degree equation on x, we have

$$x^{c+2} - (\lambda w' + c)x^{c+1} + \lambda w' c x^c - x^{c-1} + (c-1)!(\lambda w' - x)(c^2 - 2cx + x^2)e^x = 0 \quad (4)$$

For the steady-state M/M/c, utilization factor $\rho = \lambda/(c\mu)$ must be $0<\lambda/(c\mu)<1$, that is, $0<x<c$.

From Eq. (4), let us define that

$$f(x) = x^{c+2} - (\lambda w' + c)x^{c+1} + \lambda w' c x^c - x^{c-1} + (c-1)!(\lambda w' - x)(c^2 - 2cx + x^2)e^x$$

When $x=0$,

$$f(0) = (c-1)!c^2\lambda w' > 0$$

When $x=c$,

$$f(c) = -c^{c-1} < 0$$

Hence, by the Mean Value Theorem, there exists at least one x satisfying $f(x)=0$.

Suppose there are more than two x's that satisfy $f(x)=0$. Let x_1 and x_2 be two different solutions of $f(x)=0$. Since $x=\lambda/\mu$, there are two different μ_1 and μ_2 for x_1 and x_2, respectively. Because the two M/M/c models with μ_1 and μ_2 are identical, their utilization factors must be equal. When utilization factors of the two models are $\lambda/(c\mu_1)$ and $\lambda/(c\mu_2)$, respectively, $\mu_1=\mu_2$ is due to $\lambda/(c\mu_1)=\lambda/(c\mu_2)$. But, $\mu_1=\mu_2$ contradicts $x_1 \neq x_2$. Hence, x is unique. Therefore, the theorem follows. □

As a result, to construct a tandem network between a scheduler class and a server class, we adjust the service rate in turn from μ_0 to μ_1 and from μ_1 to μ_2 to keep $W_0=W_1=W_2$ in Fig. 4, so that a scheduler (server) class is transformed into its corresponding single M/M/c queuing system. In other words, a scheduler(server) class modeled by Y M/M/c with each arrival rate λ'/Y and service rate μ_0 can finally be transformed to one M/M/cY with arrival rate λ' and service rate μ_2, resulting in a two-stage tandem network.

4.2. Optimal number of servers

The tandem network between a scheduler class and a server class is depicted in Fig. 5 as a result of the transformation process of Section 4.1. A scheduler class is transformed to one M/M/c_1 with arrival rate $\lambda(1-p_k)$ and service rate μ_a for each channel and a server class is also transformed to one M/M/c_2 with arrival rate $P_i\lambda'$ and service rate μ_b for each channel. In the scheduler class, as the rejected rate is λp_k by Observation 3.1, the accepted rate is $\lambda(1-p_k)$ that is the actual arrival rate of M/M/c_1. Because the departure rate of M/M/c_1 is $\lambda'=\lambda(1-p_k)$ equal to its actual arrival rate, the arrival rate of M/M/c_2 is $P_i\lambda'$ when the probability for client requests to require this server class's services is P_i.

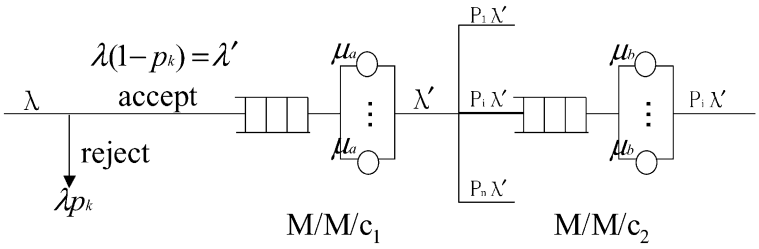

Fig. 5. Tandem network between a scheduler class and a server class.

The variable related with the optimal number of servers in a server class is c_2. Since the total service rate of the servers need not be greater than that of schedulers,

$$c_2\mu_b \le c_1\mu_a \tag{5}$$

On the other hand, for the steady-state M/M/c_2 in Fig. 5, its utilization factor $(P_i\lambda')/(c_2\mu_b)$ should be less than 1.

$$\frac{P_i\lambda'}{c_2\mu_b} < 1 \tag{6}$$

From inequalities (5) and (6), the range of c_2 becomes

$$\frac{P_i\lambda(1-p_k)}{\mu_b} = \frac{P_i\lambda'}{\mu_b} < c_2 \le \frac{c_1\mu_a}{\mu_b} \tag{7}$$

As in Fig. 4, c_2 is $c \times Y$ where c is the number of threads of each server process and Y is the number of servers in a server class. Hence, inequality (7) can be rewritten as follows:

$$\frac{P_i\lambda'}{c\mu_b} < Y \le \frac{c_1\mu_a}{c\mu_b} \tag{8}$$

If inequality (8) is satisfied, we consider that the number of servers Y is in the range of the optimal number of servers. If inequality (8) is not satisfied, the number of servers must be adjusted. Note that we can obtain the range of the optimal number of servers by applying the transformations repetitively.

Intuitively, we can notice some meaningful information from expressions (7) and (8). Because $P_i\lambda'$ in the left term of these expressions denotes the total arrival rate to the server class, inequality $(P_i\lambda')/(\mu_b) < c_2$ (or $(P_i\lambda')/(c\mu_b) < Y$) implies that the server class should have enough duplicated servers to support the incoming requests. Hence, this inequality is connected with supporting the acceptable response time for clients. On the other hand, $c_1\mu_a$ in the right term of expressions (7) and (8) denotes the total service rate of the scheduler class. Therefore, the inequality $c_2 \le (c_1\mu_a)/(\mu_b)$ (or $c_2 \le (c_1\mu_a)/(c\mu_b)$) is related to the guarantee of server utilization to some extent. In consequence, our proposed method analyzes the optimal number of servers in a server class by considering both the average response time and server utilization.

Note that the service rate of a multi-threaded process is not in direct proportion to the number of threads. In other words, when μ_0 is the service rate of a thread in a single threaded process, the service rate of a process with c threads is not $c \times \mu_0$ in general. This is because overhead from context switches, synchronization between threads, shared resource contention, etc. take a certain amount of time. Hence, we need to adjust the service rate of a multi-threaded process with weight function $\omega(h)$, where h is the number of threads, $0 < \omega(h) < 1$. If the service rate of a thread in a single threaded process is μ_0, then that of a thread in a multi-threaded process is $\omega(h) \times \mu_0$. Though $\omega(h)$ is dependent on specific system environments such as the number of processors, thread structure and so on, the value of $\omega(h)$ can be generally considered as in Fig. 6. Some experimental results from which we can approximate the $\omega(h)$ function can be found in Ref. [7]. If the service rate μ of a scheduler or a server is adjusted with the weight function $\omega(h)$, the optimal number of servers analyzed in this paper would be more reasonable and practi-

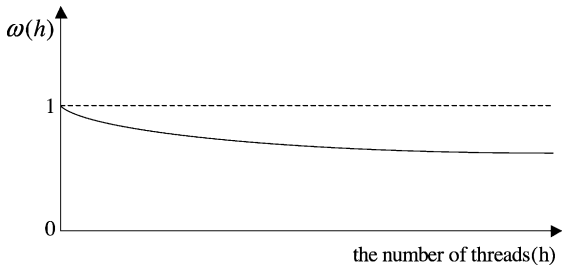

Fig. 6. The weight function $\omega(h)$.

cable. One of the applicable weight functions can be as follows, based on the experiments in Ref. [7]:

$$\omega(h) = \begin{cases} \frac{4}{5} & 1 < h \le 4 \\ \frac{3}{4} & 5 \le h \le 30 \\ < \frac{3}{4} & h > 30 \end{cases}$$

5. Experiments

In this section, we perform several experiments using DEVSim++. DEVSim++ is a C++-based discrete-event modeling framework which has been used in many areas of systems' design such as communication network design and parallel computer architecture design. Some experimental results provide new and original insight while others confirm generally known and previously established facts. For convenience, we assume that there is only one scheduler class and one server class. Because the number of threads per process is limited in general (e.g., 50 in most commercial database servers and 10 in OSF DCE) and the size of a queue is often a few times more than the number of threads [16], we assume in our experiments that the number of threads per process is 10 and the size of a queue per process is 100. Since the service rate of a scheduler is usually higher than that of a server, the mean service rate of a server and that of a scheduler is assumed to be 1/s and 10/s, respectively.

Fig. 7 depicts the range of the optimal number of servers in a server class, which is calculated using the transformations proposed in this paper, as the arrival rate increases with other parameters being fixed. For example, when the arrival rate is 100, the range of the optimal number of servers is from 11 to 14. As we analyzed in Section 4.2, the lower bound and the upper bound of the optimal number of servers are closely related with the acceptable response time for clients and the guarantee of server utilization, respectively. We can observe in Fig. 7 that the acceptable response time (i.e., the lower bound) is more sensitive to the changes of the arrival rate than the guarantee of server utilization (i.e., the upper bound). For example, when the arrival rate increases from 10 to 30, the

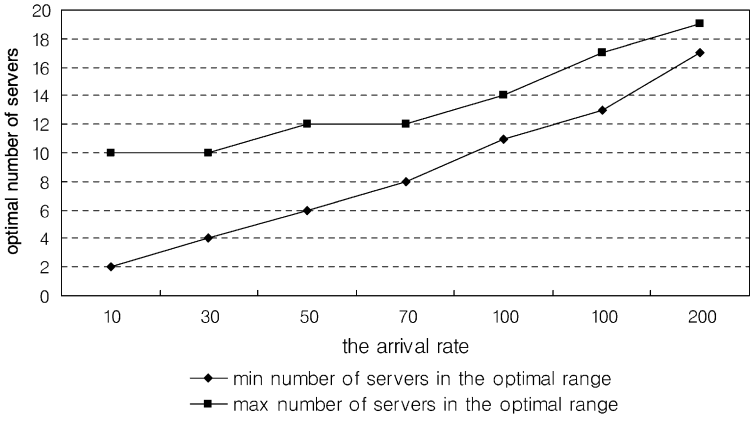

Fig. 7. The arrival rate vs. optimal number of servers.

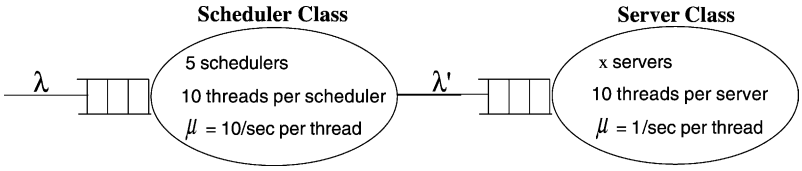

Fig. 8. A tandem network between a scheduler class and a server class.

lower bound is changed from 2 to 4 while the upper bound remains unchanged.

The following discussion illustrates the importance of a proper method of determining optimal number of servers by showing how simple intuitive estimation may not correctly determine its range. Consider Fig. 8 that is the queuing network of the system environment for Fig. 7. The scheduler class is composed of 5 schedulers, each of which has 10 threads and each thread's service rate is 10/s. A rough estimation may conclude that the scheduler class could perform 500 service requests per second (from 5 *schedulers* × 10 *threads* × 10 *service rate*), and the server class could process $10 \times x$ requests per second (from x *servers* × 10 *threads* × 1 *service rate*), where x is the number of identical servers in the server class. Suppose λ is 200 in Fig. 8. λ' is the same as λ because the scheduler class can process all client requests without rejection. Then, the number of servers should be greater than or equal to 20 from the equation $10x \geq 200$. But, the range of the optimal number of servers determined by our method is from 17 to 19. We understand that this kind of simple estimations may not correctly provide the optimal number of servers. Hence, proper mechanisms such as the method proposed in this paper are required.

Fig. 9 shows the response time with increasing the number of servers, which is simulated by DEVSim++. Note that the range of the optimal number of servers under the given condition is between 11 and 14 from Fig. 7. The experimental result of Fig. 9 implies that the range of the optimal number of servers determined by our method can achieve high system performance, considering both the response time and server utilization. If the number of servers in the server class is less than 11, it may cause a very long response time. On the other hand, when the server class is constructed with more than 14 servers, it would waste system resources (i.e., low server utilization) without many gains in the response time.

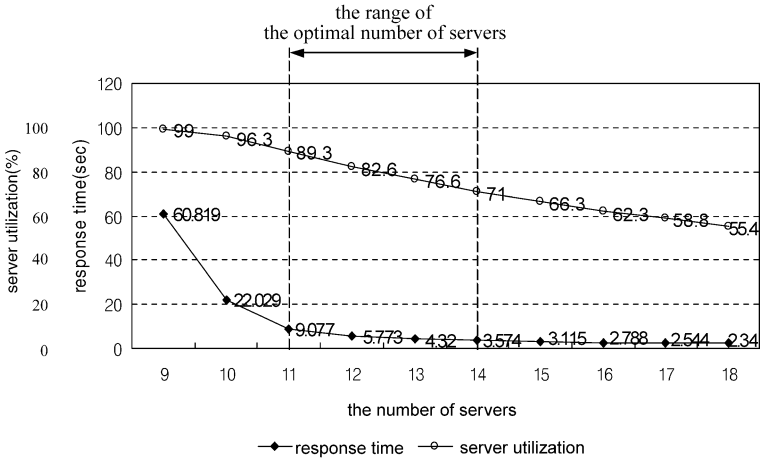

Fig. 9. The changes of the number of servers.

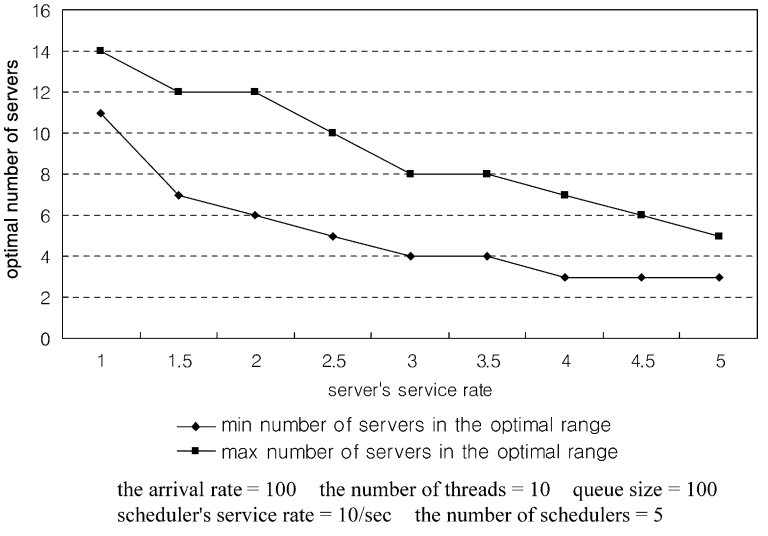

Fig. 10. Server's service rate vs. optimal number of servers.

As you expect, the increase of a server's service rate induces the decrease of the optimal number of servers as in Fig. 10 and the improvement of the response time as in Fig. 11. It also shows that the average response time and the server utilization have an inverse relationship. On the other hand, Fig. 12 depicts that the increase of the number of threads brings down the range of the optimal number of servers because its increase improves servers' total service rate. These results confirm generally well-known facts.

Though a single server with a large number of threads may provide a high service rate (note, however, that there are certain limitations in the effective number of threads, e.g., 50 in most commercial database servers and 10 in OSF DCE), there are several reasons for multiple identical servers in a server class. With a single multi-threaded server, all the services provided by the server would be unavailable if that server is down. When there is only a single server in a single host, the system capacity can also be often overloaded from many clients' requests due to the

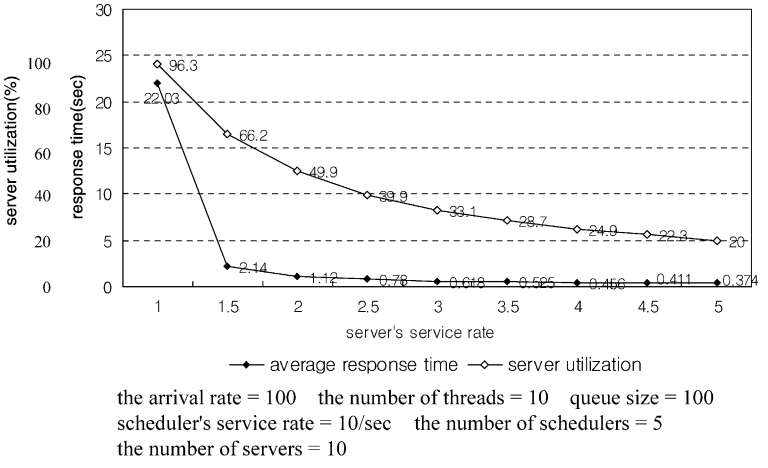

Fig. 11. Server's service rate vs. system performance.

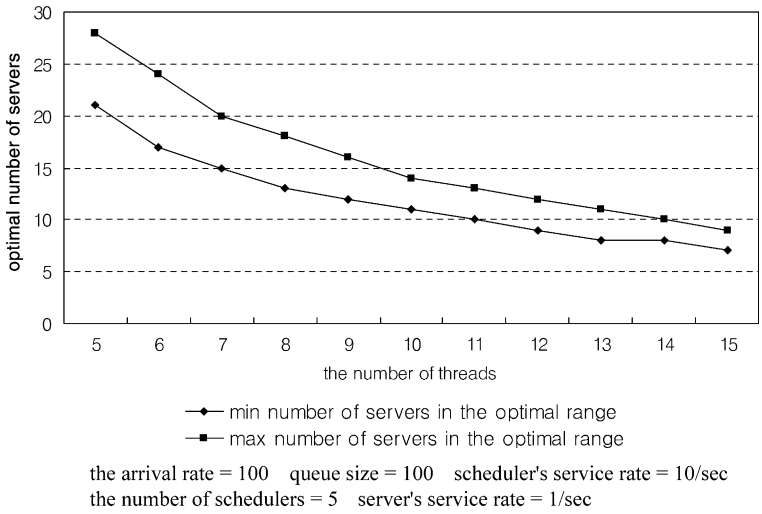

Fig. 12. The number of threads vs. optimal number of servers.

limited amount of system resources and the management of a large amount of control blocks [10]. The communication bottleneck can also be a serious problem. All these problems can be effectively avoided with the concept of the server class. This is because the multiple identical servers in each server class can be placed in several different hosts, which can physically distribute the servers.

There is another issue that deserves to be addressed. Consider the performance of the following two cases:

1. m servers with each server having n threads and
2. n servers with each server having m threads.

Though these two cases have the same total number of threads (i.e., $m \times n$), the actual performance is not the same. There are two conflicting reasons for this phenomenon. First, as mentioned in Section 4.2, the total service rate of a process with c threads, each of which has service rate μ_0, is not $c \times \mu_0$, but is $c \times \omega(h) \times \mu_0$ $(0 < \omega(h) < 1)$. Here, h is the number of threads and $\omega(h)$ decreases from 1 to 0 as h increases. Thus, if $m > n$, m servers with n threads would give better performance than n servers with m threads because $m \times n \times \omega(n) \times \mu_0 > n \times m \times \omega(m) \times \mu_0$. The second reason is based on the fact that the performance of one pooled M/M/c is better than that of c M/M/1 queues [24]. In other words, as the number of separate queuing systems increases, the average performance of the overall system decreases. This is because the workload could not be completely balanced among all separate queuing systems due to the natures of Poisson arrivals and exponential service times. Fig. 13 is the experimental result that shows the average response times and service rates with several combinations of <the number of threads, the number of servers> such that multiplying the number of threads by the number of servers is fixed to 60. According to the figure, small number of servers with large number of threads gives the better response time. Though the result may change with different $\omega(h)$ functions, we can generally observe that the effect of many separate queuing systems is not negligible. So, we may need to use as many threads as the system can effectively support. However, as mentioned before, we must also consider the fault-tolerance and availability issues, which means we need multiple servers rather than a single server with a large amount of threads even though a large number of threads can be supported by the system. As a rule of thumb, at least two or more servers in each host need to be maintained.

In summary, we can find out the following facts from the experiments. First, the average response time of a server class increases as its utilization increases. And, although over a certain extent of the number of identical servers in a server class, the server utilization just decreases without much benefit in the average

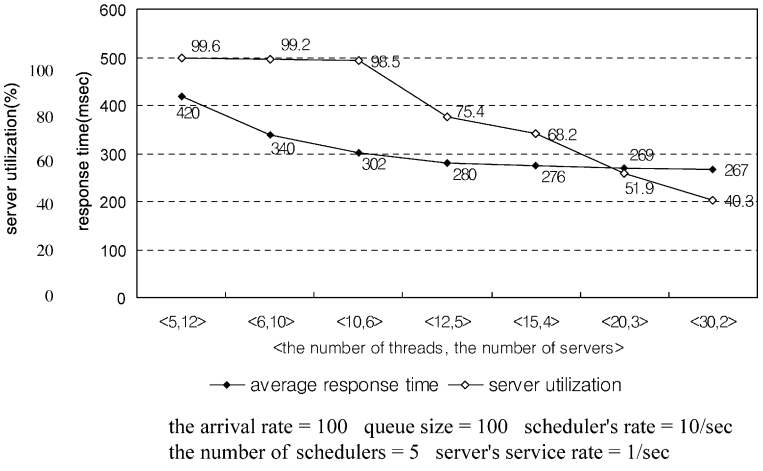

Fig. 13. The combination of the number of threads and the number of servers.

response time. Thus, to achieve high system performance, we need some mechanisms that can compromise between the average response time and the server utilization. In our proposed method, we can find out the optimal number of servers in a server class by appropriately reflecting the above facts. In addition, we notice that a server class constructed by a small number of servers with a large number of threads gives the better performance than other cases. However, in the commercial system, there are certain limitations in the effective number of threads and the concept of a server class is necessary and appealing in distributed client/server systems. Hence, our formulations in this paper can provide the optimal number of servers in a server class, based on the number of hosts and the effective number of threads the system can support.

6. Conclusion and further work

The concept of a server class that consists of multiple identical servers is important in distributed client/server computing systems, even if a single server with many threads can process a large number of arrival requests. This is because high fault-tolerance and continuous availability can make application services reliable. With a single multi-threaded server for an application, all the services provided by the server would be unavailable if that server is down. A server class consisting of multiple identical servers, however, can avoid this problem. It also effectively facilitates distributed computing because multiple identical servers can be placed in several different hosts, which can physically distribute the servers. With only a single host, the system capacity can be often overloaded from many clients' service requests due to the limited amount of system resources and the management of tremendous control blocks [10]. The communication bottleneck can also be a serious problem with a single host. All these problems can be gracefully alleviated with the concept of server classes.

However, the optimal number of servers is known to be difficult to determine in general. When the number of servers in a certain server class is too small, the response time for those requests may not be satisfactory. On the other hand, several servers would be in the idle state if there are too many servers. This means the waste of system resources. In this paper, we present a transformation mechanism from an open Jackson network to a tandem network based on queuing theory. And then, we develop a method that provides the range of the optimal number of servers for a general model of the distributed client/server processing environment.

There are many performance parameters in our model such as the number of threads, the number of schedulers, scheduler's service rate, and queue sizes, which are interrelated and affect the optimal number of servers. Though the results of the experiments may

change with different values of the parameters, our paradigm for estimating the optimal number of servers presented in this paper can be generally applicable.

We only considered the first come–first served (FCFS) queue discipline. Since other queuing discipline such as a priority queue may also be widely used in practice, we need to further investigate different performance models that can incorporate other queuing disciplines.

References

[1] P.J. Burke, The output of a queueing system, Operations Research 4 (1956) 699–704.

[2] P.J. Burke, The output process of a stationary M/M/s queueing system, The Annuals of Mathematical Statistics 39 (1968) 1144–1152.

[3] T.L. Casavant, J.G. Kuhl, A taxonomy of scheduling in general-purpose distributed computing systems, IEEE Transactions on Software Engineering 14 (Feb. 1988) 141–154.

[4] P. Caseau, G. Pujolle, Throughput capacity of a sequence of queues with blocking due to finite waiting room, IEEE Transactions on Software Engineering 5 (6) (Nov. 1979) 631–642.

[5] L.M. Casey, Decentralized scheduling, Australian Computer Journal 13 (May 1981) 58–63.

[6] W. Chu, C.M. Sit, Estimating task response time with contentions for real-time distributed systems, Proceedings of the 9th Real-Time System Symposium, 1988, pp. 272–281.

[7] P. Dickens, M. Haines, P. Mehrotra, D. Nice, Towards a thread-based parallel direct execution simulator, Technical Report, NASA Langley Research Center and The College of William and Mary (1995).

[8] D.L. Eager, E.D. Lazowska, J. Zahorjan, Adaptive load sharing in homogeneous distributed systems, IEEE Transactions on Software Engineering 12 (5) (May 1986) 662–675.

[9] S.B. Gershwin, An efficient decomposition method for the approximate evaluation of tandem queues with finite storage space and blocking, Operations Research 35 (Mar. 1987) 291–305.

[10] J. Gray, Transaction Processing: Concepts and Techniques, Morgan Kaufmann, San Mateo, CA, 1993.

[11] C.J. Hou, K.G. Shin, Load sharing with consideration of future task arrivals in heterogeneous distributed real-time systems, IEEE Transactions on Computers 43 (9) (Sept. 1994) 1076–1090.

[12] J.R. Jackson, Networks of waiting lines, Operations Research 5 (4) (1957) 518–521.

[13] L. Kleinrock, Queueing Systems: Computer Applications, Wiley, New York, 1974.

[14] T. Kunz, The Learning Behaviour of a Scheduler using a Stochastic Learning Automation, Technical Report, Technical University Darmstadt (Dec. 1991).

[15] J.C.S. Lui, R.R. Muntz, D. Towsley, Bounding the mean response time of the minimum expected delay routing policy: an algorithmic approach, IEEE Transactions on Computers 44 (12) (Dec. 1995) 1371–1382.

[16] Open Software Foundation, OSF DCE Application Development Guide, Prentice Hall, Upper Saddle River, NJ, 1995.

[17] H.G. Perros, T. Altiok, Approximate analysis of open networks of queue with blocking: tandem configurations, IEEE Transactions on Software Engineering 12 (Mar. 1986) 450–461.

[18] E. Reich, Waiting times when queues are in tandem, The Annuals of Mathematical Statistics 28 (1957) 768–773.

[19] E. Reich, Note on queues in tandem, The Annuals of Mathematical Statistics 34 (1963) 338–341.

[20] J.A. Stankovic, Stability and distributed scheduling algorithms, IEEE Transactions on Software Engineering 11 (1985) 1141–1152.

[21] J.A. Stankovic, An application of Bayesian decision theory to decentralized control of job scheduling, IEEE Transactions on Computers 34 (1985) 117–130.

[22] D. Towsley, P. Sparaggis, C. Cassandras, Optimal routing and buffer allocation for a class of finite capacity queuing systems, IEEE Transactions on Automatic Control 37 (9) (Sep. 1992) 1446–1451.

[23] Y.T. Wang, R.J.T. Morris, Load sharing in distributed systems, IEEE Transactions on Computer 34 (1985) 204–217.

[24] R.W. Wolff, Stochastic Modeling and The Theory of Queues, Prentice Hall, Englewood Cliffs, NJ, 1989.

[25] Y. Zhao, W.K. Grassmann, The shortest queue model with jockeying, Naval Research Logistics 37 (1990) 773–787.

Jin Hyun Son is a full-time lecturer of the Department of Computer Science and Engineering at Hanyang University, Ansan, Korea. He received his BS degree in Computer Science from Sogang University, Seoul, Korea, in 1996, and his MS and PhD degrees in Computer Science from KAIST in 1998 and 2001, respectively. His research interests include database systems, distributed processing, object-oriented systems and e-commerce.

Myoung Ho Kim is a professor of the Division of Computer Science at Korea Advanced Institute of Science and Technology (KAIST), Daejon, Korea, from 1989. He received his BS and MS degrees in Computer Engineering from Seoul National University, Seoul, Korea, in 1982 and 1984, respectively, and his PhD degree in Computer Science from Michigan State University, East Lansing, MI, in 1989. His research interests include database systems, OLAP, mobile computing, transaction management, information retrieval, workflow and distributed processing. He is a member of the ACM and IEEE Computer Society.