Proceedings of the 2000 Winter Simulation Conference
J. A. Joines, R. R. Barton, K. Kang, and P. A. Fishwick, eds.

AN ANALYTICAL MODEL AND AN OPTIMAL SCHEDULING HEURISTIC FOR COLLECTIVE RESOURCE MANAGEMENT

Qiang Sun

Siebel Systems, Inc.
2207 Bridgepointe Parkway
San Mateo, CA 94404, U.S.A.

ABSTRACT

In the this paper, we study the problem of collective resource management. We first introduce the problem through real-world examples. Then we generalize the problem and build an analytical model using queuing theory. Based on this model, we evaluate the expected average waiting time of tasks. We present data from simulations, and compare the expected average waiting time from theoretical calculations to that from our experiments. We propose an optimal task scheduling heuristic. We conclude with a brief discussion of our future research plans.

1 INTRODUCTION

First let's consider the following three scenarios.

- We need to organize a soccer team before the next game. The team should consist of any four computer science professors and any seven computer science majors. We need to check the schedules of professors and students, and make appointments for eleven people who will have made no other commitments for the time of the game.
- We need to transfer a video file from a network hard drive to a VCR for recording. Suppose we have smart electronics, and the reservation is done automatically with the hard drive and the VCR.
- Suppose we have a microprocessor with multiple floating point calculation units, integer point calculation units, and caches. The execution of each instruction needs to utilize one or more of these different units. There are precedence requirements for the execution of the instructions.

The preceding are all examples of what we call collective resource management (*CRM*) problems. A *CRM* problem should have the following properties:

1. There exists a resource pool made up of different types of resources. Each type of resource has multiple homogeneous instances. For example, professors, a VCR, and integer point units are all considered to be resources.
2. The use of the any resource instance in the pool is strictly exclusive. For example, we can't use a VCR to record two tapes at the same time.
3. Each *task*, an atomic action on the associated resources, requires a collection of resources. Multiple instances of any resource type might be required. For example, schedules of more than one professor need to be modified for the soccer game.
4. There might be lease (Sun Microsystems 1999), priority, precedence, deadline, or other requirements associated with each task. For example, instructions need to be executed in certain order.

In this paper, we focus on *CRM* problems that fulfill the first three properties.

2 RELATED WORKS

CRM problems appear in the study of distributed databases, CPU scheduling, real-time systems, etc. Priority assignments (Chang and Livny 1985 and Huang et al. 1991), task scheduling (Abbot and Garcia-Molina 1988 and Huang et al. 1989), concurrency control(Bestavros and Braoudaks 1994, Gupta et al. 1996, and Sha, Rajkumar, Lehoczky 1988), and fault resilience (Chandy 1998 and Sun, Zhang, and Zhang 2000) are some of the many commonly faced issues when we try to solve *CRM* problems. The unique challenge introduced by *CRM* lies in the fact that there are homogeneous instances of each resource type, which is what makes solving *CRM* problems complicated.

3 ALGORITHMS FOR CRM

3.1 Problem Generalization

Before we get into our algorithm, we need to generalize *CRM* problems. We adopt the following notations:

- Resource types: $R_1, R_2, \ldots, R_n$;

- Number of instances for R_i: N_i;
- A task request is represented by: $T((r_1, r_2, \ldots, r_n), E)$, where r_m is the number of type R_m required for this task and E is the expected execution time of the task. If r_i is non-zero, we say that R_i is required by T, and R_i belongs to the *resource set* of T. Note that $r_i < N_i$ must hold.

3.2 Description of the Algorithm

We call a source that generates Ts a client; and each resource instance is managed by an server. Once a T is generated, it is submitted to all the servers that manage resources required by T. Each server has a *FIFO* waiting queue . If a server can not fulfill a T immediately—because the resource is temporarily unavailable—T is put into the waiting queue. Once a T is taken from the waiting queue, the resource is locked by the client which submits T. We use a two-phase locking scheme (Bestavros and Braoudaks 1994 and Gupta et al. 1996) to ensure the atomicity of the task across multiple resources. Preemption is not allowed and thus the resources is held by the client until the completion of the task.

Client Events (See Figure 1):

- A – T is generated and submitted to servers
- B – Client receives a notification from server that T has been dequeued, and lock on this server is obtained. If enough instances of a particular resource type have been locked, notify those servers which have not grant locks to the client to remove T from their waiting queues.
- C – Locks for all required resources have been obtained. Task initialization notification is sent to locked servers.
- D – Task is completed and locks are released.

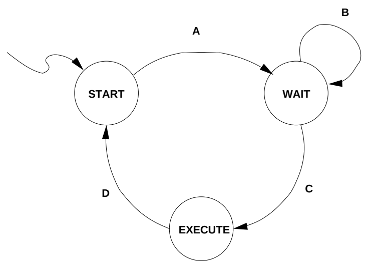

Figure 1: Client State Diagram

Server Events (see Figure 2):

- A – Server dequeues the next T from the queue, and it sends a notification to the client where T originates from.
- B – Server receives notification from client to remove T.

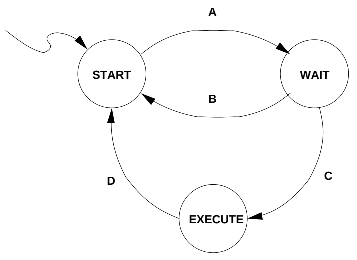

Figure 2: Server State Diagram

- C – Task is initiated.
- D – Task is completed.

We also define for:

- T_{queue} to be the amount of time that T spends in the queue.
- T_{block} to be the amount time between the T is dequeued and start of the execution.
- T_{exec} to be the amount time for execution. Note that $T_{exec} = E$.
- T_{wait} to be the sum of T_{queue} and T_{block}

4 ANALYTICAL MODEL

4.1 Assumptions

We model the arrival of Ts from each client using a *Possion* distribution with a mean arrival rate λ. Thus the probability of exactly one arrival of T in time δt is $\lambda \delta t$, and the inter-arrival density is $\lambda e^{-\lambda t}$ (Robertazzi 1994).

T_{exec} is assume to satisfy an exponential decay distribution. We use $1/\beta$ to denote the mean of T_{exec}. Thus the corresponding probability density of T_{exec} being t is $\beta e^{-\beta t}$ (Robertazzi 1994).

Given any T the number of required resource types and the number of instances required within a give resources type are assumed to be exponentially distributed random variables. We also assume that the any given resource type has the same probability of being required by a T.

Problems with *Possion* arrival time of jobs and exponential execution time on servers can be model using *Markovian* queues. In fact, if only one of the r's required by a T has non-zero value, and this non-zero value is 1, *CRM* problem degenerates to problem of networks of queues, and each R_i can be modeled using a $M/M/N_i/\infty$ queue (Kleinrock 1976 and Robertazzi 1994). Notice that we make the assumption that the capacity of the waiting queue is infinite.

4.2 Abstraction

In order to use theories from queuing systems to help build our analytical model, further abstractions are necessary.

First, we merge the waiting queues of the different instances of the same resource type into one waiting queue (see Figure 3). By doing so, we have essentially constructed a $M/M/N_i/\infty$ queue. Let such queue of R_i be Q_i. We restrict our model to systems that are saturated at equilibrium. In other words, we are only interested in the cases when the waiting queue is not empty. Because of parallelism in execution, the equivalent mean task time of Q_i becomes $1/(\beta N_i)$. We simplify the notation to $1/\beta_i$. Due to T_{block}, the effective mean task time is longer than $1/\beta_i$. Therefore, we use $\beta_i - d_i$ to represent the expected number of task executed per unit time.

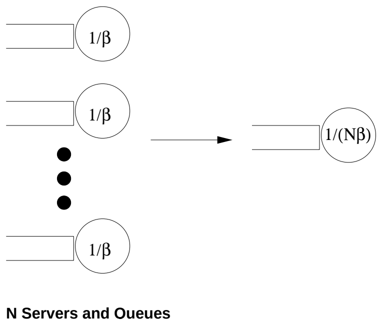

Figure 3: Queue Merging

Second, we unify all the clients into a client. We know that a group of *n Possion* processes each with a mean arrival rate λ is equivalent to a single *Possion* process with a mean arrival rate λn. In the rest of the paper, we simply use λ to denote the mean arrival rate of T from the unified client. Let σ_i be the probability that a T requires R_i. Thus the mean arrival rate for Q_i is $\sigma_i \lambda$. Notice that $\sum_{i=1}^{n} \sigma_i = m > 1$, where m is the expected number of resource types required by a T.

The state of Qs is described using vectors

$$\vec{S} \overset{\text{def}}{=} (s_1, s_2, \ldots, s_n)$$

where s_i is the number of Ts in Q_i (Robertazzi 1994). From this point on, we consider the T that is being executed to be in the queue as well. This is slight modification is necessary for the derivation presented in section 4.3. Similarly,

$$\vec{1_i} \overset{\text{def}}{=} (0, 0, \ldots, 1, \ldots, 0)$$

describes a state in which Q_i is the only non-empty queue with one T. And,

$$\vec{s_i} \overset{\text{def}}{=} (s_1, s_2, \ldots, s_i, \ldots, s_n)$$

describes the set of states in which there are s_i T's in Q_i.

4.3 Evaluation of T_{total}

We define the throughput of Q_i to be $\lambda\sigma_i$, and denote it with Φ_i. $\Phi_i = \lambda\sigma_i$ is also called *Traffic Equation* (Robertazzi 1994). Using results from networks of queues, we can set up the following equations:

$$\Phi_i P(\vec{S} - \vec{1_i}) = (\beta_i - d_i) P(\vec{S}). \tag{1}$$

$$\Phi_i P(\vec{S}) = (\beta_i - d_i) P(\vec{S} + \vec{1_i}). \tag{2}$$

$$(\lambda + \sum_{i=1}^{n} (\beta_i - d_i)) P(\vec{S}) = \sum_{i=1}^{n} \Phi_i P(\vec{S} - \vec{1_i}) + \sum_{i=1}^{n} (\beta_i - d_i) P(\vec{S} + \vec{1_i}). \tag{3}$$

Equation (1) and (2) are *Balance Equations* (Robertazzi 1994) of Q_i. It states that, under equilibrium, the net flow into and out of Q_i are the same. Equation (3) is the Global Balance Equation. It states that, under equilibrium, the net flow in and out of a state is the same (Schwartz 1987). These equations enable us to derive the probability of the system in state $\vec{S}$ (Chen 1987):

$$\begin{aligned} P(\vec{S}) &= (\frac{\Phi_i}{\beta_i - d_i}) P(\vec{S} - \vec{1_i}) \\ &= (\frac{\Phi_i}{\beta_i - d_i})^{s_i} P(s_1, s_2, \ldots, 0, \ldots, s_n) \\ &= \prod_{i=1}^{n} (\frac{\Phi_i}{\beta_i - d_i})^{s_i} P(\vec{0}). \end{aligned} \tag{4}$$

We know that:

$$\sum_{\vec{S}} P(\vec{S}) = 1$$

. Therefore, we can use the normalization of probability to solve $P(\vec{0})$ (Schwartz 1987):

$$P(\vec{0}) X = 1$$

$$\begin{aligned} X &= \sum_{\vec{S}} \prod_{i=1}^{n} (\frac{\Phi_i}{\beta_i - d_i})^{s_i} \\ &= \prod_{i=1}^{n} \sum_{s_i=0}^{\infty} (\frac{\Phi_i}{\beta_i - d_i})^{s_i} \\ &= \prod_{i=1}^{n} (1 - \frac{\Phi_i}{\beta_i - d_i})^{-1}. \end{aligned}$$

Thus we get:

$$P_i(\vec{0}) = 1 - \frac{\Phi_i}{\beta_i - d_i}. \quad (5)$$

If we substitute equation (5) into (4), we have:

$$P_i(\vec{s_i}) = (1 - \frac{\Phi_i}{\beta_i - d_i})(\frac{\Phi_i}{\beta_i - d_i})^{s_i}$$

.

Using $Little's Law$, we can calculate the expected delay of T's in Q_i:

$$T_{wait} = \frac{\sum_{s_i=1}^{\infty} s_i P_i(\vec{s_i})}{\sum_{s_i=1}^{\infty} (\beta_i - d_i) P_i(\vec{s_i})} \quad (6)$$

where the numerator is the mean number of T in Q_i, and the denominator is the mean rate of T passing through Q_i. We observe that the denominator can be simplified to $\beta_i - d_i$. We know that the value of the numerator can be approximated in the following way:

$$Let\ K = \frac{\Phi_i}{\beta_i - d_i},\ thus$$

$$\begin{aligned} \sum_{s_i=1}^{\infty} s_i P_i(\vec{s_i}) &= \sum_{s_i=1}^{\infty} s_i (1-K) K^{s_i} \\ &= (1-K)\int_1^{\infty} s_i K^{s_i} ds_i \\ &= (1-K)K(\frac{1}{(\ln K)^2} - \frac{1}{\ln K}). \quad (7) \end{aligned}$$

The above approximation is valid only when the constraint $K < 1$ is satisfied. The real-world implication is that the arrival rate of tasks should never be higher than the process rate of the server. Otherwise the length of the waiting queue would approach infinity.

Using equations (6) and (7), We obtain T_{wait} as a function of β_i, Φ_i:

$$T_{wait} = (1 - \frac{\Phi_i}{\beta_i - d_i})\frac{\Phi_i}{(\beta_i - d_i)^2}(\frac{1}{(\ln \frac{\Phi_i}{\beta_i - d_i})^2} - \frac{1}{\ln \frac{\Phi_i}{\beta_i - d_i}}). \quad (8)$$

5 SIMULATIONS

We carry out simulations in order to verify the derivation of T_{wait}. In the first set of experiments, the value of Φ_i is fixed and the the value of β_i is manipulated. In the second set of experiments, the value of β_i is fixed and the value of Φ_i is manipulated. The results of these experiments compared with the theoretical predictions are illustrated in Figure 4 and 5. We observe that formula (8) correctly predicts both the shape and the magnitude of T_{wait}.

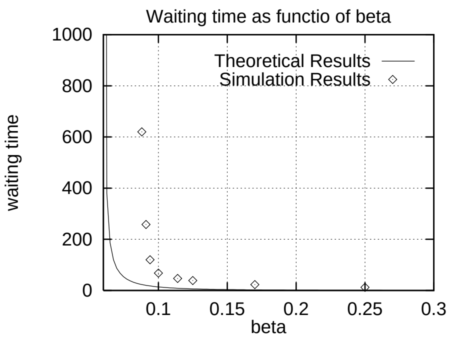

Figure 4: Simulation Results with Fixed Phi = 0.06

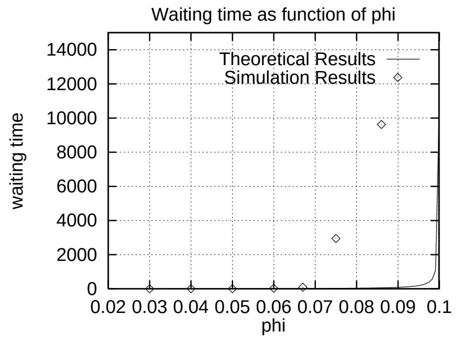

Figure 5: Simulation Results with Fixed Beta = 0.1

The discrepancy between the results from simulations and those obtained by using formula (8) is caused by the fact d_i is omitted from the calculation of theoretical results. As we can see from the discussion so far, to minimize T_{wait}, the key is to minimize d_i, which offsets β Our model would predict T_{wait} more accurately if the analytical form of d_i could be derived.

We propose that the value of d_i satisfies an exponential decay distribution. And the mean value of d_i is k_i/β, where k_i is a variable of λ, N_i, and r. At any time, among all Ts on different Qs, there is at least one T that is in the processing phase. In other words, not all Ts in the system are waiting in the Qs or are blocked in the locking phase. If we can have a partial order of T based on their completion time, then the expectation value of d_i should be proportional to the sum of T_{exec} of all correlated Ts that enter execution phase prior to T. We call two Ts correlated when the joint of their resource sets is not an empty set. Notice that there are overlapping between the executions of non-correlated Ts.

6 HEURISTIC

In order to reduce d_i, we must delay the entry into the locking phase of Ts. Before granting the lock to any T, we need to check if delaying such a grant will allow other T to complete early without delaying the completion of this task. We first check if any lock has been granted to this task, T_1, from other Qs. If not, we calculate the expected completion time of T_1 and the next task in Q, T_2, under the condition that the lock is granted to T_1 by Q_i. Then we try to switch the order of T_1 and T_2 in all Qs. We check if such a switch causes deadlock. If not, we re-calculate the expected completion time for T_1 and T_2, under the condition that the lock is granted to T_2 by Q_2. If the expected completion time for T_1 is not increased and the expected completion time for T_2 is decreased, such a switch should be carried out.

This heuristic is similar to that proposed in (Sun, Zhang, and Zhang 2000), but there are some difference. First, we check if any lock has been granted to T_1 before proceed with the heuristic. This procedure helps to avoid redundant calculations. Second, our heuristic is only meant to be used in a non-distributed environment, and thus it is possible to have information of hard global state. This allows us to prevent deadlock and re-order Ts globally. Here is the pseudo code for the heuristic:

```
optimize(queue Q_i){
  task T_1 = getTask(Q_i, 1);
  if(no locks has been granted to T_1){
    time t_1=calculateCompletionTime(Q_i,1);
    time t_2=calculateCompletionTime(Q_i,2);
    task T_2=getTask(Q_i,2);
    switch the order of T_1 and T_2
                                in all Qs;
    if(detect no deadlock){
      time t_3 =
             calculateCompletionTime(Q_i,1);
      time t_4 =
             calculateCompletionTime(Q_i,2);
      if(!(t_4 > t_1) && t_3 < t_2)
        switch is valid;
      else
        restore the order of T_1 and
                            T_2 in all Qs;
     }
  }
}
```

The method $Queue\ getTask()$ returns the T at the given index in Q. The index of the first T in Q is 1. The method $calculateCompletionTime()$ returns the estimation of expected completion time of T at the given index in Q. Both methods are easy to implement, so we will skip their details in this paper.

7 SUMMARY AND FUTURE RESEARCH

In this paper, we introduce the problem of Collective Resource Management(CRM). We formalize the problem by specifying the special characteristics of *CRM.* An analytical model is built and the expected waiting time of tasks are derived. We use simulation to obtain data to support our analytical model. We present a heuristic to reduce expected waiting time.

For future research, we plan to build simulations to test the effectiveness of our heuristic and to derive an empirical form of d_i. We also plan to study some unique issues of *CRM* in distributed environments.

ACKNOWLEDGMENTS

I would like to express my greatest gratitude to Professor Duane Bailey at Williams College, MA, Irena Pashchenko, an undergraduate student at Smith College, MA, and Hao Zhang, a graduate student at UC Berkeley for their advice and invaluable help in the writing of this paper.

BIBLIOGRAPHY

Robert Abbott and Hector Garcia-Molina. 1988. Scheduling real-time transactions: A performance evaluation. In *Proceedings of Very Large Data Base Conference*, 1–12.

Azer Bestavros and Spyridon Braoudaks. 1994. Timeliness via speculation for real-time database. In *Proceedings of IEEE Real-Time Systems Symposium*, 36–43.

K. M. Chandy. 1998. Using announce-listen with global events to develop distributed control systems. *Concurrency:Practice and Experience*.

Hung-Yang Chang and Miron Livny. 1985. Priority in distributed system. In *Proceedings of IEEE Real-Time Systems Symposium*, 123–130.

W. Chen. 1987. *Solution manual for telecommunications networks: Protocol, modeling and analysis*. Addison-Wesley.

Ramesh Gupta, Jayant Haritsa, Krithi Ramamritham, and S. Seshadri. 1996. Commit processing in distributed real-time database systems. In *Proceedings of IEEE Real-Time Systems Symposium*, 220–229.

Jiandong Huang, John A Stankovic, Krithi Ramamritham, and Don Towsley. 1989. Experimental evaluation of real-time transaction processing. In *Proceedings of IEEE Real-Time Systems Symposium*, 144–153,

Jiandong Huang, John A Stankovic, Krithi Ramamritham, and Don Towsley. 1991. On using priority inheritance in real-time databases. In *Proceedings of IEEE Real-Time Systems Symposium*.

Leonard Kleinrock. *Queueing theory*. Wiley, 1976.

Thomas G. Robertazzi. 1994. *Computer networks and systems : Queueing theory and performance evaluation*. Springer-Verlag.

M. Schwartz. 1987. *Telecommunication networks: Protocol, modeling and analysis*. Addison-Wesley.

Lui Sha, Ragunathan Rajkumar, and John P. Lehoczky. 1988. Concurrency control for distributed real-time databases. *ACM Sigmod Record*, 17(1):82–98.

Q. Sun, H. Zhang, and J.H. Zhang. 2000. A time-stamp based solution for collective resource acquisition in distributed systems. In *HICSS-33*.

Sun Microsystems, Inc. 1999. *Jini Distributed Leasing Specification*.

AUTHOR BIOGRAPHY

QIANG SUN currently works as a Software Engineer in the server infrastructure group at Siebel Systems, Inc. He enjoyed his undergraduate life at Williams College, MA and California Institute of Technology. He received a bachelor's degree in Computer Science and Physics, Summa Cum Laude, from Williams College in June, 2000. His research interests include distributed systems and issues in concurrent computation. His email address is `<qsun@siebel.com>`.