# SYNCHRONOUS QUEUING: A CO-ALLOCATION MECHANISM FOR MULTIMEDIA ENABLED GRIDS

*FARAG AZZEDIN and MUTHUCUMARU MAHESWARAN*

Advanced Networking Research Laboratory
Department of Computer Science
University of Manitoba
Winnipeg, MB R3T 2N2, Canada
{fazzedin, maheswar}@cs.umanitoba.ca

## Abstract

*The multimedia enabled Grid (MEG) is an extension of the Grid concept to support the deployment of multimedia services. To provide an adequate level of service to multimedia applications, it is often necessary to simultaneously allocate the resources including predetermined capacities from the interconnecting networks to the applications. The simultaneous allocation of resources is often referred to as co-allocation in the Grid literature. In this paper, we propose a novel scheme called synchronous queuing (SQ) for implementing co-allocation with quality of service (QoS) assurances in Grids. Unlike existing approaches, SQ does not require advance reservation capabilities at the resources. In the simulation, we increase the imposed load with increasing number of machines to test SQ's effectiveness in addressing the co-allocation problem. This is because the co-allocation complexity is known to increase with the increase of the imposed load. The simulation studies performed to evaluate SQ indicate that it outperforms an admission control-based scheme by a significant margin.*

**Keywords:** Resource allocation & management, real time scheduling, multimedia technologies & applications.

## 1. INTRODUCTION

The *multimedia enabled Grid* (MEG) is an extension of the Grid [5, 7] to meet the ever-increasing demand for multimedia from users engaging in a wide range of activities. For some multimedia applications, to provide an adequate level of service to the users, it is often necessary to allocate these resources including predetermined capacities from the interconnecting networks simultaneously to the particular applications. Examples of applications that require simultaneous allocation of resources include multimedia conferencing, virtual reality based distributed interactive simulation, distance learning, etc. The simultaneous allocation of resources is often referred to as *co-allocation* in the Grid literature.

We propose a novel scheme called *synchronous queuing* (SQ) for co-allocation that does not require advance reservation capabilities at the resources. The scheme provides co-allocation with QoS constraints, i.e., it is possible to perform co-allocation with hard QoS guarantees as well as soft QoS guarantees and best effort. The SQ is an aggregate-based scheme that assures the total work accomplished by each subtask $s_i$ does not fall behind or exceed its agreed QoS. In addition, SQ is also an environment-aware scheme that assures the aggregate work accomplished by a subtask $s_i$ does not fall behind or exceed the aggregate work done by other subtasks belonging to task $t$. This QoS guarantee is different from the traditional admission control-based QoS guarantee which is (a) instantaneous guarantee and requires the application to be adaptive and sense its own progress, (b) probabilistic in the sense that a subtask $s_i$ requiring $m\%$ of a local machine's resources might get for each schedule cycle a different value $x$ in the neighborhood of $m$ depending on the machine's load, and (c) environment-unaware in that it does not know about other subtasks' progress to assure that the co-allocation skew is minimized for all subtasks belonging to the same task.

Section 2 presents the notation and mathematically defines the co-allocation problem. Section 3 examines the related work. Section 4 describes the SQ algorithm. Simulation results and discussion are presented in Section 5.

## 2. NOTATION AND PROBLEM DEFINITION

Let $t$ denote a task submitted by a client to the Grid for processing and let this task $t$ be composed of $n$ subtasks $s_0,\ldots,s_{n-1}$. Consider the situation where a Grid-level scheduler maps the different subtasks to different machines in the Grid. The Grid-level schedulers assign to a particular machine various tasks and subtasks, which are further scheduled by the local scheduler that controls the machine according to a local policy. Some of these tasks/subtasks might have co-allocation requirements and others may not.

Once the subtasks $s_0,...,s_{n-1}$ of task $t$ are assigned to the different machines, it is the responsibility of the local schedulers to allocate sufficient machine resources (e.g., CPU quanta) to execute each subtask. Because the different local schedulers will have different mix of tasks and subtasks their behavior will be different. Note that because task $t$ has co-allocation requirements all its subtasks *must* proceed with their execution simultaneously. Some of these subtasks might be delayed because they are not allocated sufficient resources. This delay is referred to as *co-allocation skew.* The goal of the SQ algorithm is to minimize this co-allocation skew for all applications that require co-allocation.

Consider two subtasks $s_i$ and $s_j$ that become runnable at the first schedule cycle. Let $r_{s_i}$ be the weight of subtask $s_i$ and $W_k^{s_i}$ be the work done by subtask $s_i$ at the $k^{th}$ schedule cycle. Then, subtasks $s_i$ and $s_j$ are said to be synchronized if, for any $k^{th}$ schedule cycle the normalized aggregate work done since the two subtasks $s_i$ and $s_j$ became runnable are identical (i.e., $\frac{1}{r_{s_i}} * \mathop{}\limits_{k=1}^{m} W_k^{s_i} - \frac{1}{r_{s_j}} * \mathop{}\limits_{k=1}^{m} W_k^{s_j} = 0$). This is an idealized definition of synchronization that assumes infinitely divisible subtasks. Hence, the objective of synchronous queuing is to minimize $\left| \frac{1}{r_{s_i}} * \mathop{}\limits_{k=1}^{m} W_k^{s_i} - \frac{1}{r_{s_j}} * \mathop{}\limits_{k=1}^{m} W_k^{s_j} \right|$ for all $i, j,\ i \neq j$.

## 3. RELATED WORK

The *Globus Architecture for Reservation and Allocation* (GARA) system [6] extends the Globus resource management architecture [2] by providing new features such as support for (a) advance reservation and (b) heterogeneous resource types. Another approach to the co-allocation problem is the Tenet Real-Time Protocol Suite. This system is a suite of tools developed for multi-party communications and it offers advance reservation capabilities to its network clients [4]. The SQ is different from GARA and Tenet in several ways. Our approach addresses the co-allocation without the need for advance reservation capability at the target nodes.

While performing co-allocation via advance reservations simplifies the problem, this approach has several drawbacks. One of the drawbacks is that this model does not allow over subscription of the resources, which could potentially cause under utilization of the overall system. Another drawback is that the advance reservation-based approach imposes strict timing constraints on the client side.

Implicit co-scheduling [1] is a new time-sharing approach for scheduling parallel applications that uses the communication and synchronization that occur naturally within the application to coordinate scheduling across workstations. Here, two events *response time* and *message arrival* are used to decide whether to continue with executing a subtask or to block it and schedule another subtask. The basic idea is that, if a response to a request arrives, or a message arrives from a cooperating subtask executing on a different processor, it means that the remote subtask was scheduled at that time. Therefore, it is beneficial to continue executing the local subtask. On the other hand, if message arrivals do not occur, then the executing subtask will use a two-phase spin blocking mechanism to wait. Under certain situations, waiting might be better than context switching to another subtask. While implicit co-scheduling presents a new approach for improving the global performance of parallel applications, it falls short of addressing real-time multimedia applications. Further, unlike SQ, the implicit co-scheduling is targeted towards message passing subtasks. Implicit co-scheduling provides an application-level solution to the co-allocation problem (i.e., the application has to sense its own progress and adapt accordingly) whereas; SQ addresses the problem at the scheduler level. Thus, SQ does not require changes to the applications.

## 4. SYNCHRONOUS QUEUING

A local machine's load is due to three types of tasks: Grid QoS, Grid best effort, and local tasks. The tasks that belong to the different types are assigned to the different queues for execution as shown in Figure 2. A hierarchy of schedulers is used within each local machine. The interQueue scheduler determines which queue should be selected whereas the intraQueue scheduler decides which task or subtask should be scheduled from the selected queue. The *round robin* (RR) scheduler is used as the intraQueue scheduler for the local and Grid best-effort tasks queues while a *Start-time Fair Queuing* (SFQ) [3] is used for the Grid QoS tasks queue. The SFQ based scheduler provides fairness in resource allocation that is necessary to provide QoS guarantees. The SFQ is also used as the interQueue scheduler. The SFQ achieves fair CPU allocation among the three queue or among the QoS tasks/subtasks based on their weights.

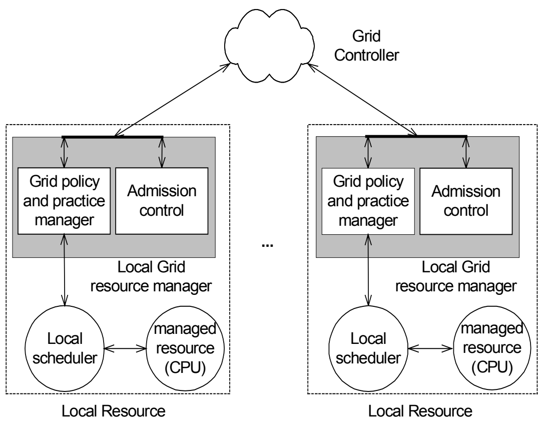


**Figure 1:** Grid topology used in simulation.

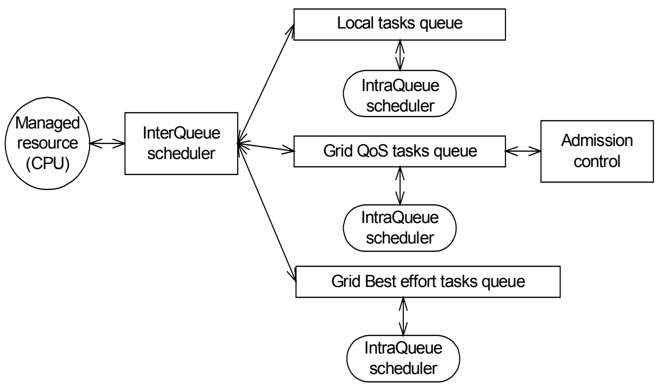


**Figure 2:** Components of local scheduler.

After each schedule cycle or a much larger interval (e.g. a group of schedule cycles), the local scheduler reports the progress of the hard QoS co-allocation subtasks to the Grid controller as shown in Figure 1. The work done by each of these subtasks is calculated by the subtask's local machine and monitored by the Grid controller to assure that aggregated work accomplished by each subtask does not fall behind the other subtasks belonging to task $t$. The aggregated work done is calculated by using real ($RT$) and virtual ($VT$) time clocks [9]. Let $t_0$ be the starting $RT$ when $s_0$ starts execution. Initially $RT = pRT = VT = 0$, where $pRT$ is the previous $RT$. For each schedule cycle ($y$), $RT$ will be advanced by $y$. However, for the same schedule cycle, $VT$ will be advanced by $\frac{1}{x} * (RT - pRT) * x'$, where $x$ is the agreed quantum allocated for $s_i$, and $x'$ is the actual quantum $s_i$ gets. After $VT$ is computed, $pRT$ is set to $RT$.

## 4.1. SELECTING A PIVOTAL POINT

Upon receiving the progress information of the hard QoS co-allocation subtasks from the local machines, the Grid controller selects a pivotal point as $pp = \frac{1}{n} * \sum_{i=0}^{n-1} VT_i$, where $n$ is the number of subtasks belonging to task $t$, and $VT_i$ is the virtual time for subtask $s_i$. So, *pp* is essentially the average of virtual time for the $n$ subtasks belonging to task $t$.

## 4.2. DETECTION OF CO-ALLOCATION SKEW

As mentioned in Section 2, the objective of SQ is to satisfy $\left| \frac{1}{r_{s_i}} * \sum_{k=1}^{m} W_k^{s_i} - \frac{1}{r_{s_j}} * \sum_{k=1}^{m} W_k^{s_j} \right| \leq threshold$, for all $i, j,\ i \neq j$). The value of the threshold is provided as follows. For each Grid hard QoS task, the clients provide two QoS attributes: *asynchrony*, and *overall deviation*. *Asynchrony* is the acceptable co-allocation skew that a task $t$ can tolerate and is calculated as, $async = VT_f - VT_s$, where $VT_f$ is the virtual time of the fastest subtask, and $VT_s$ is the virtual time of the slowest subtask among all subtasks belonging to task $t$. The o*verall deviation* is the acceptable retardation or acceleration that a task $t$ can tolerate for its subtasks.

For each task $t$, its *pp* is checked whether it falls within the *overall deviation* window. If it does, then the asynchrony test is performed to ensure that it is within the asynchrony window. If the *pp* of task $t$ does not fall within the *overall deviation* window or the asynchrony test fails, corrective action is required as discussed in the next subsection. The pseudo-code for detection of asynchrony is presented in Figure 3.

## 4.3. CORRECTIVE ACTION

When asynchrony is detected, the Grid controller signals a local machine for a corrective action which may be to speedup or slowdown subtask $s_i$. The local machine might succeed or fail in carrying out the corrective action locally. Failure can happen in situations where subtask $s_i$ needs to speed up and the local machine is already overloaded. In other words, the local machine has no extra

CPU cycles to spare and all the CPU cycles are allocated to tasks/subtasks. In this case, the local machine reports back to the Grid controller for a global corrective action to take place. On the other hand, success always happens in situations where subtask $s_i$ needs to slow down and also under situations where subtask $s_i$ needs to speed up and its local machine in under loaded (i.e. CPU cycles can be borrowed easily).

```
Monitor(Grid QoS queue) {
    while (Grid QoS queue is not empty)
        //dequeue all subtasks belonging to a hard QoS task $t$
        queue = dequeue(QoSQ)
        //calculate pivotal point of task $t$
        pp = calculate_pp(queue)
        if (pp is within the overall window)
            if (async > asynchrony-window)
                corrective_action(queue)
        else
            corrective_action(queue)
    endwhile
}
```

**Figure 3:** Asynchrony detection routine.

```
CorrectiveAction( queue) {
    while (queue is not empty)
        //dequeue a subtask
        subtask= dequeue(queue)
        //Determine the appropriate action to be taken
        // The action can be speeding up or slowing down the subtask
        action = determine_action(subtask)
        //if action can be carried locally
            //signal the subtasks's local machine to carry the action
        //else
            //mark this subtask's action to be taken globally
    endwhile
}
```

**Figure 4:** Corrective action routine.

A hard QoS task/subtask subject to a corrective action will have its weight increased or decreased and this will not affect any other hard QoS tasks/subtask because their weights are not affected and hence SFQ will assure their share of the CPU remains the same. Furthermore, whatever happens (increasing or decreasing tasks/subtasks' weights) in Grid QoS tasks queue does not affect the other two queues (local and Grid best effort tasks queues) because their weights are not affected and thus the interQueue scheduler (SFQ) will assure the local and Grid QoS tasks queues that their share of CPU remains the same. Therefore, SQ guarantees a total isolation between the tasks/subtasks in the Grid QoS queue as well as a total isolation between the three different queues. The pseudo-code is presented in Figure 4.

## 5. SIMULATION RESULTS AND DISCUSSION

The Grid topology used consists of 5 to 25 local machines each with a local generator uniformly generating [1000, 2000] tasks. There are also 2 Grid generators generating GridQoS and GridBE tasks in the range of [1000, 2000]. For each simulation run, the generators generate a Poisson stream of tasks with specified InterArrival Time ($\lambda$) in the range of [10, 100, 200, 500] seconds. For each hard QoS task, two QoS attributes: (*asynchrony*, and *overall deviation*) are uniformly generated in the range of [100, 500] seconds. Furthermore, each Grid task is uniformly composed of [0, number of local machines] subtasks and each of these subtasks is randomly assigned an execution time in the range of [1500, 2000] seconds.

The SQ is compared with a QoS and admission control-based scheme using four performance metrics: average co-allocation skew, acceptance ratio, effective cycles delivered, and QoS conformance. *Average co-allocation skew* is defined as the average of the difference in the finish time of all subtasks belonging to a task. *Acceptance ratio* is defined as the ratio between QoS tasks accepted and the QoS tasks generated. *QoS-conformance*, and *effective cycles delivered* are defined as the ratio of hard QoS tasks confirming to overall deviation and asynchrony windows, respectively. From Figure 5, it can be observed that SQ outperforms QoS by a significant margin. The increase of the co-allocation skew as the machine numbers increase is due to the increase in the subtasks belonging to a task. Since the co-allocation complexity is known to increase with the increase of the imposed load, we intended to increase the imposed load with increasing number of machines to test SQ's effectiveness in addressing the co-allocation problem over various degrees of complexity.

Figure 6 and 7 show that QoS conformance and effective cycles delivered are much higher for SQ. As $\lambda$ increases, the drop in QoS conformance and effective cycles delivered stays higher for QoS than SQ. This shows the effectiveness of SQ even with the variation of $\lambda$. As the co-allocation skew increases, the conformance to the *asynchrony* and the *overall deviation* is violated. This violation causes QoS conformance and effective cycles delivered to decrease as the co-allocation skew increases. Since SQ uses relaxed admission control, more QoS tasks will be accepted as shown in Figure 8. While minimizing the co-allocation skew, SQ Accommodate more QoS tasks and still guarantees total isolation between the three different queues as well as between the hard QoS tasks in the Grid QoS tasks queue.

## 6. CONCLUSIONS

This paper addressed the co-allocation issue in MEGs. Primarily, the co-allocation issue is concerned with allocating sufficient resources to all the subtasks of an application such that the different subtasks can make satisfactory progress with their execution. Co-allocation is an essential feature for several important classes of multimedia applications and it is an important consideration when the applications are mapped onto a distributed system.

The MEGs is a generalized form of distributed system that is based on the Grid concept. In a typical MEG environment, resources will be managed by a hierarchy of schedulers, i.e., Grid-level schedulers and local schedulers. These scheduling hierarchies coupled with the enforcement of site autonomies makes co-allocation a challenging problem.

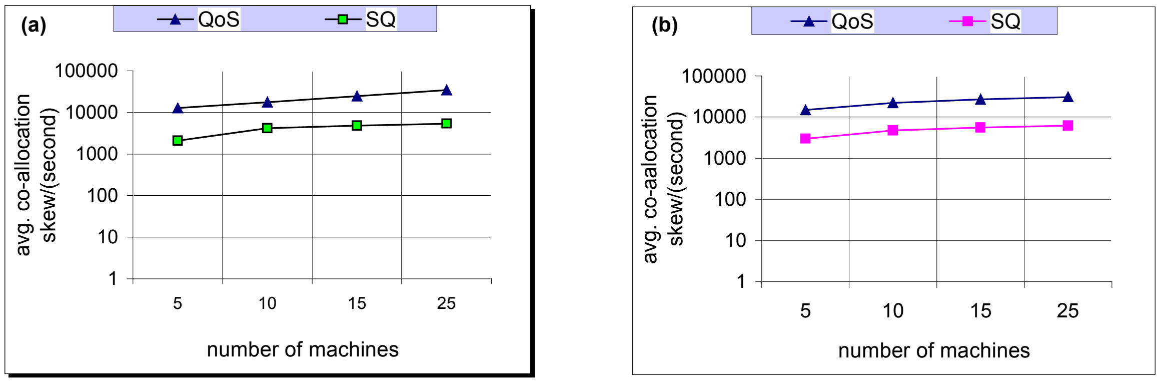


**Figure 5:** Variation of co-allocation skew with number of machines for $\lambda$ equals (a) 10 and (b) 200 seconds.

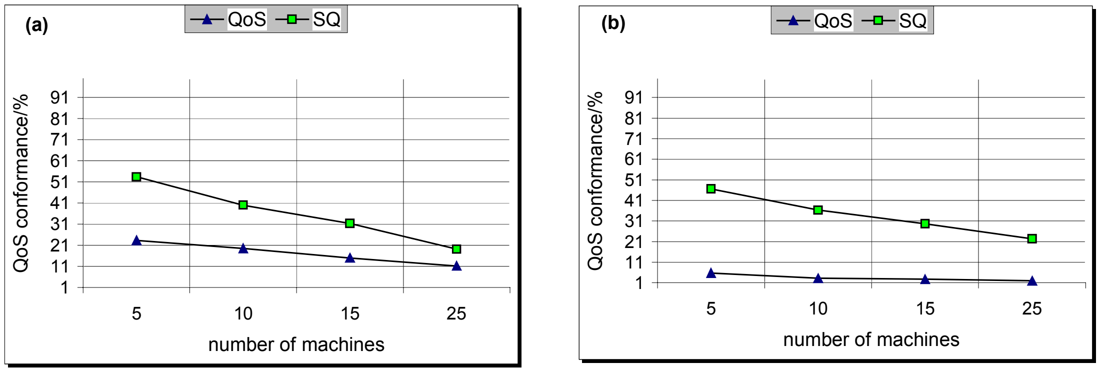


**Figure 6:** QoS conformance with different number of machines for $\lambda$ equals (a) 10 and (b) 200 seconds.

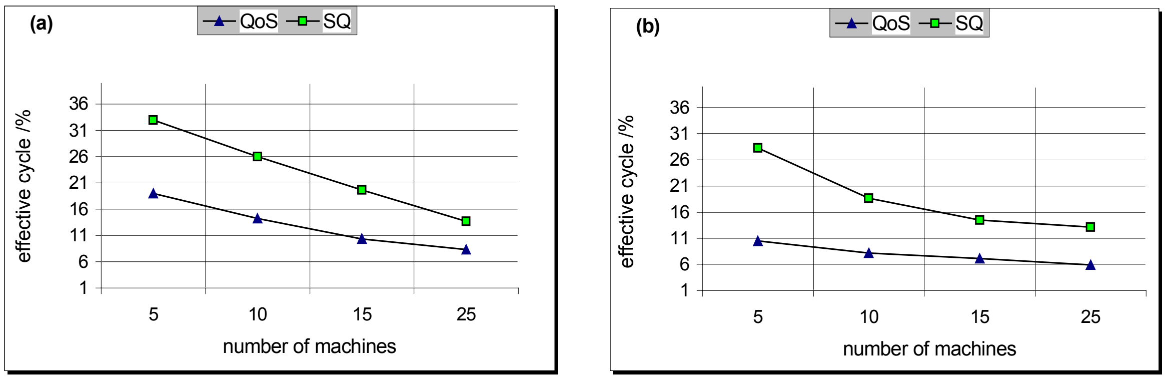
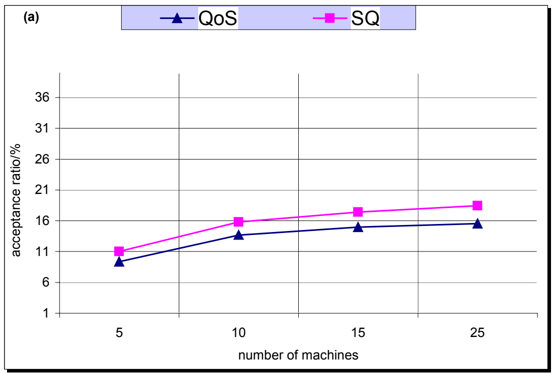


**Figure 7:** Effective cycles delivered for number of machines for $\lambda$ equals (a) 10 and (b) 200 seconds.



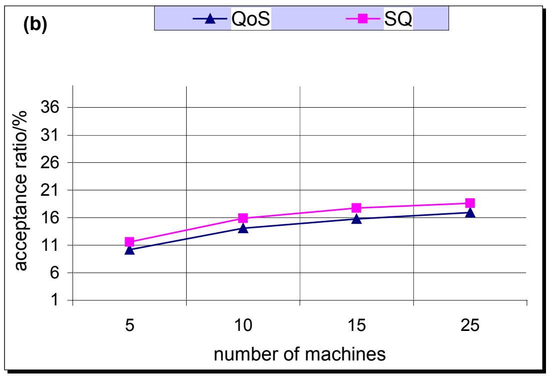


**Figure 8:** Acceptance ratio with number of machines for $\lambda$ equals (a) 10 and (b) 200 seconds.

This paper proposes a novel scheme for co-allocation in MEGs called the SQ algorithm. Unlike existing approaches for co-allocation, the SQ does not require advance reservation capabilities at the target resources. The SQ has the following key attributes: (a) memory-oriented QoS capability, where SQ remembers the total work accomplished by each subtask $s_i$ in the previous schedule cycles, (b) environment-aware QoS capability, where SQ assures that the aggregated work accomplished by each subtask $s_i$ does not fall behind the other subtasks belonging to task $t$. Other subtasks may be running on different local machines and thus it is important for SQ to have an environment-aware QoS capability, (c) framework for co-allocation without the need for advance reservation, and (d) framework for co-allocation with the ability to over subscribe resources. The algorithm and architecture for implementing SQ are presented. Simulation studies performed to evaluate SQ indicate that it outperforms an admission control-based scheme by a significant margin. The simulation studies were performed for various numbers of machines and inter-arrival times.

## ACKNOWLEDGMENTS

This research is supported by a National Sciences and Engineering Research Council of Canada Research Grant RGP220278 and equipment used was supported by a Canada Foundation for Innovation Grant.

## References

[1] A. C. Arpaci-Dusseau, D. E. Culler, and A. M. Mainwaring, "Scheduling with implicit information in distributed systems," *SIGMETRICS Conference on the Measurement and Modeling of Computer Systems,* June 1998, pp. 233-243.

[2] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke, "A resource management architecture for metacomputing systems," *4th Workshop on Job Scheduling Strategies for Parallel Processing*, Springer-Verlag LNCS 1459, 1998, pp. 62-82.

[3] P. Goyal, H. M. Vin, and H. Cheng, "Start time fair queuing: A scheduling algorithm for integrated services packet switching networks," *ACM SIGCOMM'96*, Aug. 1996, pp. 157-168.

[4] D. Ferrari, A. Gupta, and G. Ventre, "Distributed advance reservation of real-time connections," *ACM/Springer-Verlag Journal on Multimedia Systems*, Vol. 5, No. 3, 1997.

[5] I. Foster, C. Kesselman, *The Grid: Blueprint for a New Computing Infrastructure*, Morgan-Kaufmann, July 1998.

[6] I. Foster, C. Kesselman, C. Lee, R. Lindell, K. Nahrstedt, and A. Roy, "A distributed resource management architecture that supports advance reservations and co-Allocation," *International Workshop on Quality of Service*, 1999, pp. 27-36.

[7] M. Maheswaran and K. Krauter, "A parametric-based approach to resource discovery in Grid computing systems," *1st IEEE/ACM International Workshop on Grid Computing (Grid 2000),* Dec. 2000.

[8] L. Zhang, "Virtual clock: A new traffic control algorithm for packet switching networks," *Transactions on Computer Systems*, Vol. 9, No. 2, 1991, pp. 101-124.

[9] D. Yau and S. S. Lam, "Adaptive rate controlled scheduling for multimedia applications," *ACM Multimedia Conference '96*, Nov. 1996.