Cluster Computing 7, 39–49, 2004

A Synchronous Co-Allocation Mechanism for Grid Computing Systems

FARAG AZZEDIN *
Computer Science Department, University of Manitoba, 516 Machray Hall, Winnipeg, MB R3T 2N2, Canada

MUTHUCUMARU MAHESWARAN
School of Computer Science, McGill University, 3480 University Street, Montreal, QC H3A 2A7, Canada

NEIL ARNASON
Computer Science Department, University of Manitoba, 527 Machray Hall, Winnipeg, MB R3T 2N2, Canada

Abstract. Grid computing systems are emerging as a computing infrastructure that will enable the use of wide-area network computing systems for a variety of challenging applications. One of these is the ever increasing demand for multimedia from users engaging in a wide range of activities such as scientific research, education, commerce, and entertainment. To provide an adequate level of service to multimedia applications, it is often necessary to simultaneously allocate resources including predetermined capacities from 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 formally define the co-allocation problem and 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. This enables an SQ-based approach to over subscribe the resources and hence improve resource utilization. The simulation studies performed to evaluate SQ indicate that it outperforms an QoS-based scheme with strict admission control by a significant margin.

Keywords: co-allocation, Grid computing, multi-scheduling, synchronous queuing

1. Introduction

Motivated by the successes of network computing, researchers have started examining a more generalized resource sharing infrastructure called the Grid [4,6]. The Grid is a generalized large-scale computing and data-handling virtual system that is formed by aggregating the services provided by several distributed resources. This computing infrastructure enables the use of wide-area network computing systems for a variety of challenging applications. One of these challenging applications is the ever-increasing demand for multimedia from users engaging in a wide range of activities such as scientific research, education, commerce, and entertainment. Multimedia applications (e.g., digital audio or video) impose real-time requirements on the underlying computing and communication systems [9]. To provide an adequate level of service to multimedia applications, it is often necessary to allocate different resources including predetermined capacities from the interconnecting networks simultaneously to specific applications. Examples of applications that require simultaneous allocation of resources include multimedia conferencing, virtual-reality-based distributed interactive simulation, and distance learning. The simultaneous allocation of resources is often referred to as *co-allocation* in Grid literature [4,5].

Co-allocation in a Grid environment is a much more general problem than in traditional distributed computing environments [10]. This is due to various issues including: (a) location-independent access and management of resources, (b) resource heterogeneity both in terms of capability and policy, and (c) geographically distributed location of the resources. These issues call for a resource management model with a hierarchical scheduling structure. The hierarchical scheduling structure introduces "hidden" scheduling [15] problems rendering the overall resource management, and particularly the co-allocation of resources, a challenging task.

Most existing approaches [3,5] to co-allocation in wide-area distributed systems depend on the ability of the resources to support advance reservations. While performing co-allocation via advance reservation simplifies the problem, this approach has several drawbacks. One of the drawbacks is that this model does not allow the over subscription of the resources and thus leads to under utilization of the overall system. Another drawback is that the advance reservation-based approach imposes strict timing constraints on the client side.

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 *quality of service* (QoS) constraints, i.e., it is

* Corresponding author.
E-mail: fazzedin@cs.umanitoba.ca

possible to perform co-allocation with hard QoS guarantees as well as soft QoS guarantees and best effort.

The SQ algorithm is designed for applications that have long life times. The core idea of SQ is to divide the execution time into schedule cycles and monitor the different applications at each schedule cycle to determine the progress made towards their execution. The scheduling process of SQ has two major properties: (a) aggregate-sensitiveness and (b) environment-awareness. The aggregate-sensitivity refers to the property of SQ which ensures that the total work accomplished by a subtask at all previous schedule cycles it was runnable up to the current schedule cycle is considered in determining the next allocation. The environment-awareness refers to the property of SQ which ensures that the aggregate work accomplished by a subtask does not fall behind or exceed the aggregate work accomplished by another subtask belonging to the same task that is possibly executing on a different machine.

The SQ QoS guarantee differs from the traditional admission-control-based QoS guarantee in the following ways. First, traditional admission-control-based QoS guarantee is an instantaneous guarantee and requires the application to be adaptive and sense its own progress; whereas the SQ QoS guarantee is an aggregated-based guarantee that is monitored by the scheduler for the entire life of the application. Second, traditional admission-control-based QoS guarantee is probabilistic in the sense that a subtask 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; whereas the SQ QoS guarantee is deterministic in the sense that if the subtask got less than m in the current schedule cycle, then this will be taken into consideration in the next schedule cycle and the scheduler will compensate the subtask for its loss in the previous schedule cycle. Third, traditional admission-control-based QoS guarantee is environment-unaware in that it does not know about other subtasks' progress to assure that all subtasks of the application can make satisfactory progress with their execution.

In section 2 of the paper, we present the notation and mathematically define the co-allocation problem. In section 3, we discuss a Grid resource management model while in section 4 we describe the SQ algorithm. The simulation results and a discussion of them are presented in section 5. Section 6 examines related work.

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 $s_0 \ldots s_{n-1},\ n \geqslant 1$. 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 subtasks belonging to different tasks. These subtasks are further scheduled by the local scheduler that controls the machine according to local policies. Some of these tasks may have co-allocation requirements and others may not.

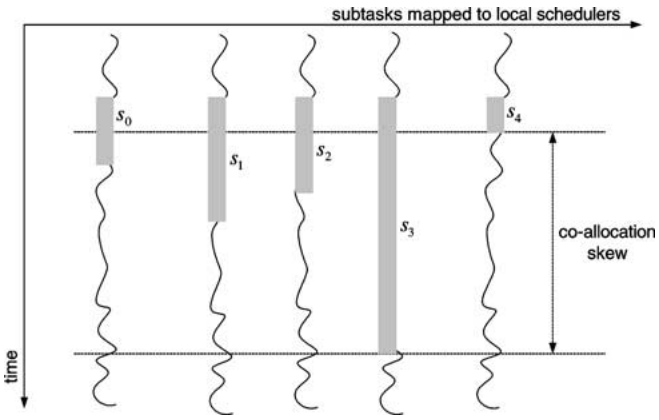

Figure 1. Five subtasks belonging to one task: An example scenario that causes a co-allocation skew. A shaded bar represents the progression of the task based on the work done given to it by its local machine.

Once the subtasks 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 a different mix of tasks, subtasks and scheduling policies, their behavior will be different. For tasks with co-allocation requirements, their subtasks must proceed with their execution in a coordinated manner. Some of these subtasks may be delayed because they are not allocated sufficient resources. The time difference between the fastest running and the slowest running subtasks of a task is called the *co-allocation skew* of the task. Figure 1 illustrates the co-allocation skew for a scenario where five subtasks belonging to a single task are allocated on different machines. The goal of the SQ algorithm is to minimize the co-allocation skew of all tasks that require co-allocation.

Let s_i and s_j be subtasks of task t that became runnable at schedule cycle q_0. 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 schedule cycle q_k. The weight of a subtask is a pre-defined relative cost or priority and accounts for differences in importance among the subtasks. Subtasks s_i and s_j are said to be synchronized at any given schedule cycle q_k if the normalized aggregate work done on the two subtasks s_i and s_j since they became runnable are identical. The synchronized condition is defined as follows:

$$\frac{1}{r_{s_i}} \times \sum_{m=0}^{k-1} W_m^{s_i} - \frac{1}{r_{s_j}} \times \sum_{m=0}^{k-1} W_m^{s_j} = 0. \tag{1}$$

This is an idealized definition of synchronization that assumes infinitely divisible subtasks. Let task t have l number of subtasks. The co-allocation skew of task t given by $\Phi_k(t)$ at schedule cycle q_k is given by:

$$\Phi_k(t) = \max_{i \in [0..l-1]} \left[\sum_{m=0}^{k-1} \frac{W_m^{s_i}}{r_{s_i}} \right] - \min_{i \in [0..l-1]} \left[\sum_{m=0}^{k-1} \frac{W_m^{s_j}}{r_{s_j}} \right]. \tag{2}$$

As mentioned previously, the objective of SQ is to minimize the co-allocation skew of all tasks requiring co-allocation. Let

Γ be the set of tasks that require co-allocation, then SQ minimizes $\sum_{t\in\Gamma} \Phi(t)$.

3. Grid resource management model

Figure 2 illustrates a Grid resource management hierarchy with two levels: the *local resource management system* (LRMS) and the *global Grid resource management system* (GGRMS). The GGRMS is responsible for assigning the subtasks to the different resources. Therefore, the GGRMS should be cognizant of the local resource status and capabilities which can be affected by the resource management policies implemented by the LRMS. In a Grid environment, in general, different resources utilize different load management systems and have their resources scheduling policies. In this model, the local resource is segmented into three partitions by the LRMS. The site autonomy comes from the fact that the local resource is free to choose any load management strategy for the local partition and also the partition sizes are governed by the local policies. However, the native scheduler of a resource is replaced with the multilevel local scheduler that is introduced here. The queuing hierarchy within a local scheduler is shown in figure 3.

Our Grid model assumes that applications are pre-decomposed into subtasks and the relative weights of the subtasks are known. These weights may be provided by the applica-

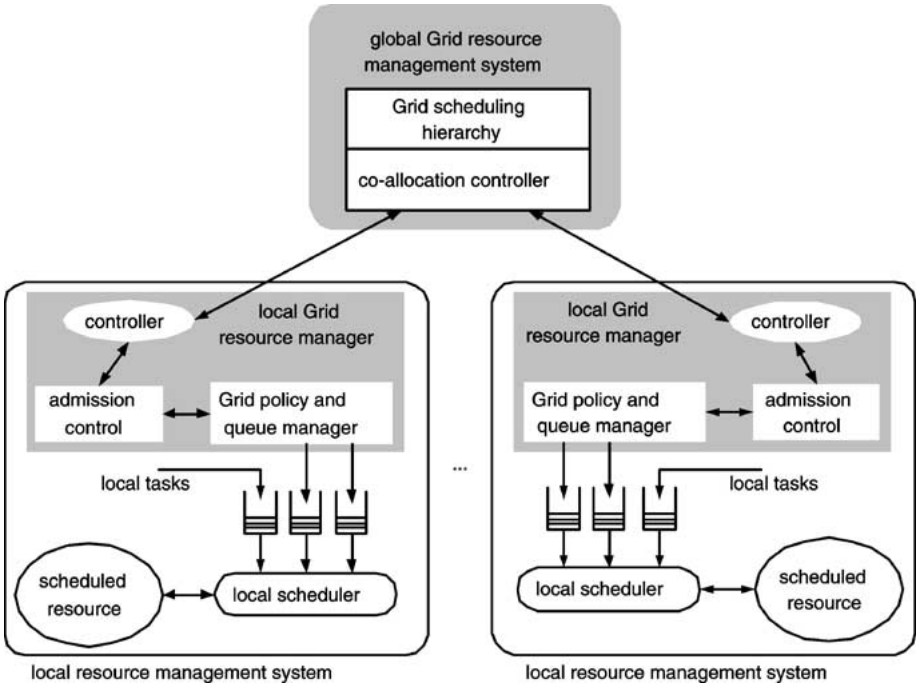

Figure 2. Overall Grid resource management hierarchy.

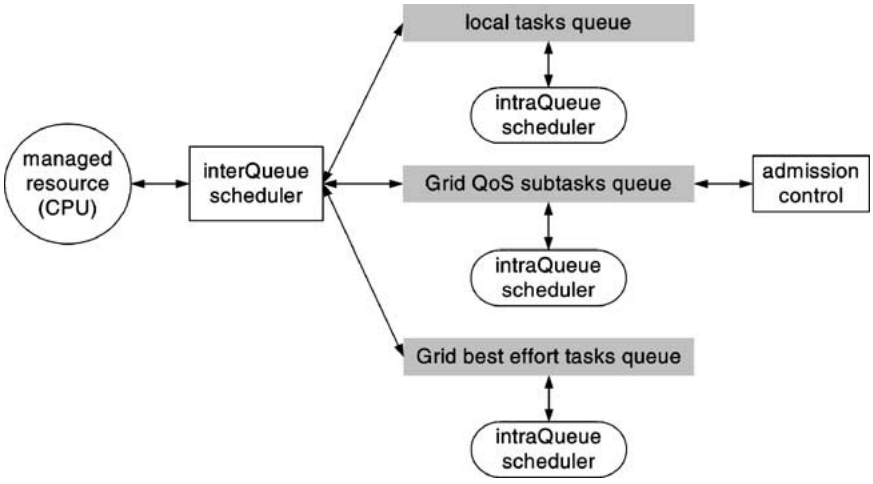

Figure 3. Queuing hierarchy within a local scheduler.

tions. Further, we do not consider any data or precedence constraints among the subtasks of a task.

4. Synchronous queuing

4.1. Overview

The co-allocation problem is concerned with ensuring that a task with several subtasks will be allocated sufficient resources so that all subtasks of the task can make satisfactory progress with their execution. These subtasks will be running in a heterogeneous environment (i.e., on different local resources) as illustrated in figure 2.

A local machine's load is due to three types of task and subtask flows: Grid QoS, Grid best effort, and local tasks. Grid QoS tasks can be further divided into hard and soft QoS tasks. The Grid QoS and the Grid best effort tasks are assigned to an LRMS by the GGRMS. Tasks or subtasks belonging to the different types are assigned to their respective queues as illustrated in figure 3. The *Grid policy and queue manager* decides how much of the local resource (e.g., CPU) is assigned to each of the three queues in each schedule cycle. Figure 3 shows the hierarchy of schedulers that are used within each local resource manager. The inter-Queue scheduler determines which queue should be selected whereas the intra-Queue scheduler decides which task or subtask should be scheduled from the selected queue.

The local resource monitors the progress made by each of the subtasks that is mapped onto it. This monitoring process is performed at an interval greater than or equal to a scheduling cycle. If the progress deviates from the expected value by more than a given threshold, the local resource notifies the Grid co-allocation controller about the violation and the value of the deviation. The co-allocation controller receives the violation reports from the different resources and makes the decisions with respect to the necessary corrective action.

The monitoring process at each local resource is made efficient by using real time (RT) and virtual time (VT) clocks [16]. The RT clock is maintained at each resource by an independent process and it has a globally consistent value. The skew among the real time clocks of the different resources are assumed to be computable and the drift is assumed to be negligible.

Suppose subtask s_j of task t becomes runnable for the first time when RT clock has value τ_0. Then $RT = pRT = VT = \tau_0$, where pRT is the value of the RT at the end of the previous scheduling cycle. Let x be the CPU quantum allocated for s_j during a schedule cycle and $\bar{x}$ be the actual value allocated for s_j. At a schedule cycle of duration q_i, the clock values are incremented as follows:

$$RT = RT + q_i, \tag{3}$$

$$VT = VT + 1/x \times (RT - pRT) \times \bar{x}, \tag{4}$$

$$pRT = RT. \tag{5}$$

If the local resource satisfies s_j's agreement (i.e., delivers x quantum of CPU cycles) for every schedule cycle up to the current one, then $RT = VT$ and thus s_j is making satisfactory progress. As can be noted from equations (3) to (5), if $RT < VT$, then s_j is progressing ahead of schedule and $RT > VT$ indicates that s_j is behind its execution schedule.

Table 1
Values of different clocks for an example subtask.

	Initial parameter value	Schedule cycle number			
		0	1	2	3
		$\bar{x} = 2$	$\bar{x} = 3$	$\bar{x} = 1$	$\bar{x} = 1$
RT	0	10	20	30	40
VT	0	10	25	30	35
pRT	0	10	20	30	40

Table 1 shows an example scenario where the different clock values are computed for a subtask that became runnable at $\tau_0 = 0$ for four schedule cycles. The duration of the schedule cycles are equal and is 10. Further, let the agreed quantum x be 2. The actual quantum $\bar{x}$ delivered to the subtask changes as shown in table 1.

4.2. Tasks flow within synchronous queuing

4.2.1. Hierarchy of local schedulers

As illustrated in figure 3, a local resource has a scheduling hierarchy containing inter-Queue as well as intra-Queue schedulers. The inter-Queue scheduler determines which queue should be selected whereas the intra-Queue scheduler decides which task or subtask should be scheduled from the selected queue. The selection process among the three different queues and the selection process among the different tasks and subtasks within each queue are based on weights assigned to the queues as well as the subtasks.

The local resource manager partitions a resource (e.g., CPU) among the three different task and subtask flows: local, Grid QoS, and Grid best effort. This partitioning is static (invariant at runtime) and is achieved by selecting three weights r_{local}, r_{GQoS}, and r_{QBE}, respectively for the three queues corresponding to the flows. The local resource is fully partitioned among the flows, i.e., $r_{\text{local}} + r_{\text{GQoS}} + r_{\text{GBE}} = 1$ (this can be expressed in terms of percentage weights as $m_{\text{local}} + m_{\text{GQoS}} + m_{\text{GBE}} = 100$). The inter-Queue scheduler uses *Start-time Fair Queuing* (SFQ) [8] to fairly allocate the local resource among the queues. For the local and the Grid best effort queues, a *round robin* (RR) scheduler is used as the intra-Queue scheduler. Thus, no weights are necessary for these two types of task flows. SFQ is used as the intra-Queue scheduler for the Grid QoS subtasks and hence, weights are associated for the Grid QoS subtasks as explained below.

4.2.2. Grid QoS subtask weight assignment

A Grid QoS task submitted at the Grid level has a CPU requirement attached with it. This requirement is expressed with respect to a standard CPU running at one GHz. A Grid QoS task contains several subtasks and the subtasks have relative weights associated with them.

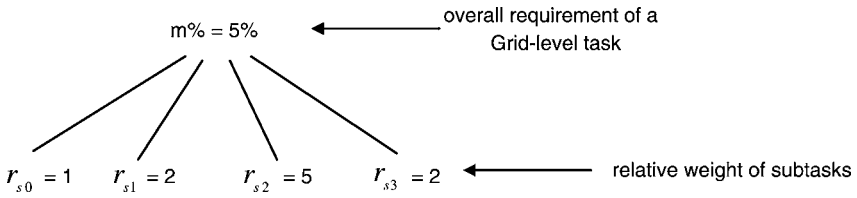

Figure 4. Assigning weights for a Grid-level QoS task and its four subtasks.

Suppose a Grid QoS task t that for each schedule cycle requires $m\%$ of the resource (i.e., CPU in this study) is mapped by the Grid resource manager onto the local resources. The length of each schedule cycle is set to 100 quanta. Let t have n subtasks $s_0 \ldots s_{n-1}$, $n \geqslant 1$. Subtask s_i has weight r_{s_i} associated with it that signifies its relative importance and we assume that r_{s_i} is provided by the application. Let d_i be the slowdown of the CPU in the local resource compared to the standard CPU, i.e., $d_i = 1/\alpha_i$, where α_i is the speed of the local machine in GHz. Subtask s_i requires $m \times r_{s_i} / \sum r_{s_i}\%$ of the standard machine. Therefore, s_i requires $m \times d \times r_{s_i} / \sum r_{s_i}\%$ of the local machine. Because the local machine dedicates only $m_{\mathrm{GQoS}}\%$ for Grid QoS tasks, the subtask should have a weight given by equation (6). It should be noted that the mapping of the subtask to local resource is infeasible if $\bar{r}_{s_i} > 1$,

$$\bar{r}_{s_i} = \left[m \times d \times r_{s_i} \Big/ \sum r_{s_i} \right] \Big/ m_{\mathrm{GQoS}}. \tag{6}$$

To illustrate the weight assignment process, consider an example scenario as depicted in figure 4. A task with four subtasks is to be mapped on the local resources. Let the task require 5% of standard machine resource. Let the machine selected for mapping be running at 100 MHz, i.e., slowdown is 10. If the relative weights of the subtasks are as given in figure 4, we can use equation (6) to compute the weights of the subtasks at the local queues. For instance, when s_0 is assigned to the local resource it requires $5 \times 10 \times 1/10\%$ of the resource assuming that the resource dedicates 100% for the Grid QoS task flow. If the resource allocates only 40% for the Grid QoS task flow (i.e., $m_{\mathrm{GQoS}} = 40\%$), the subtask s_0 should be assigned the following weight:

$$\bar{r}_{s_0} = [5 \times 10 \times 1/10]/40 = 5/40 = 0.125.$$

4.3. Basic SQ co-allocation algorithm

4.3.1. 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 $pp = 1/n \times \sum_{i=0}^{n-1} VT_i$, where n is the number of subtasks belonging to task t, and VT_i is the virtual time for subtask s_i. Therefore, pp is essentially the average of the virtual times for the n subtasks belonging to task t.

4.3.2. Detection of co-allocation skew

For each Grid hard QoS task, the client provides two QoS attributes: *asynchrony* and *overall deviation. Asynchrony* is the acceptable co-allocation skew that a task t can tolerate and is computed 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 *overall deviation* is the acceptable retardation or acceleration that a task t can tolerate for its subtasks.

(1) pp ;; pivotal point
(2) RT_t ;; real time of task t
(3) OV_t ;; overall deviation of task t
(4) SY_t ;; asynchrony of task t
(5) VT_f ;; virtual time of the fastest subtask of task t
(6) VT_s ;; virtual time of the slowest subtask of task t
(7) after all the subtasks belonging to task t reported
to the Grid co-allocation controller **do**
(8) calculate (pp)
(9) determine whether the pp is within the overall deviation window
if $pp \leqslant (RT_t - OV_t)$ **and** $pp \geqslant (RT_t + OV_t)$
the task confirms to the overall deviation window
(10) find the fastest and the slowest subtasks of task t
(11) determine whether task t violates its asynchrony window
if $(VT_f - VT_s > SY_t)$
(12) if the fastest subtask contributes to the asynchrony window more than the slowest subtask
(13) slowdown(fastest)
(14) else
(15) speedup(slowest)
(16) **else** the overall deviation window of task t is violated
(17) synchronize(t)

Figure 5. Corrective action algorithm.

4.3.3. Corrective action

The overall deviation window enforces the QoS constraints on the overall progress of the subtasks that belong to a task. That is, this window is used to prevent an overall speedup or slowdown of the task (without asynchrony) that is beyond the acceptance limits as expressed by this window. The asynchrony window is used to enforce the acceptable limit on the asynchrony.

The corrective action routine shown in figure 5 is periodically examined after the receipt of the reports sent by all subtasks of a task t at the Grid co-allocation controller. For each t, the pp is computed and checked whether it falls within the overall deviation window. If the overall deviation window is violated by the pp lying outside it, then all subtasks of task t are synchronized by speeding them or slowing them down. If pp is within the overall deviation window, we check for the asynchrony window. The asynchrony window is violated if the slowest and fastest subtasks lie outside the window. If the asynchrony window is violated, we need to slow down the fastest subtask of speed up the slowest subtask.

To slow down a subtask s_i, the slow down factor SF_i^d is determined as shown in equation (7),

$$SF_i^d = [VT_{s_i} - RT_{s_i}] \times r_{s_i}/100. \quad (7)$$

The subtask s_i transfers the weight SF_i^d to a dummy subtask that is already instantiated at the node. The weight of s_i, given by r_{s_i} is decreased by SF_i^d. Similarly, to speed up a subtask a speed up factor (SF_j^u) is computed for a subtask s_j. In the speed up case, the weight is transfered to the subtask from the dummy subtask provided that the dummy subtask is sufficiently large.

When slowing down, if there is no dummy subtask, a dummy subtask with weight equals to SF_i^d is created. The dummy subtask is inserted into the Grid QoS queue to soak the extra SF_i^d weight that s_i consumed.

The Grid co-allocation controller signals the local machine for a corrective action which may be to speedup or slowdown a subtask. The local machine might succeed or fail in carrying out the corrective action. 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 subtasks. In this case, the local machine reports back to the Grid co-allocation controller for a global corrective action to take place. On the other this code segment is executed at the Grid co-allocation controller 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 is under loaded (i.e., CPU cycles can be borrowed easily).

4.4. Properties of SQ

As mentioned in section 4.2.1, the SFQ algorithm is used to implement the inter-Queue scheduler and the intra-Queue scheduler for Grid QoS tasks and subtasks. The SFQ is shown to implement the property of "fair allocation of the CPU regardless of the variation in available processing bandwidth" with minimal error [7]. In practice, this property means that when two subtasks contend for the CPU, they will be provided the CPU in proportion to their weights. In general, if $r_1, r_2, \ldots, r_n$ denote the weights of n tasks and if B denotes the total processor bandwidth, then the ith task receives CPU bandwidth B_i given by $B_i = [r_i / \sum_{j=1}^{n} r_j] \times B$ [7].

One of the key properties of SQ that follows from the above SFQ property is that SQ ensures that each class of tasks receive at least the proportion designated by the weights of the corresponding queues. The inter-Queue scheduler which is implemented by an SFQ algorithm supports this property. Because the intra-Queue scheduler used for Grid QoS tasks and subtasks is based on SFQ, it provides another very useful property that is a subtask can speed up or slow down without affecting other subtasks within the queue subject to certain restrictions. Let $s_1, \ldots, s_n$ be the n subtasks at a Grid QoS queue. We add a dummy subtask s_0 to the set to facilitate the slowing down of a subtask s_i by moving a fraction of its weight (r_i) to s_0. Similarly, subtask s_i can be sped up by transferring some of r_0's weight to r_i provided r_0 is sufficiently large.

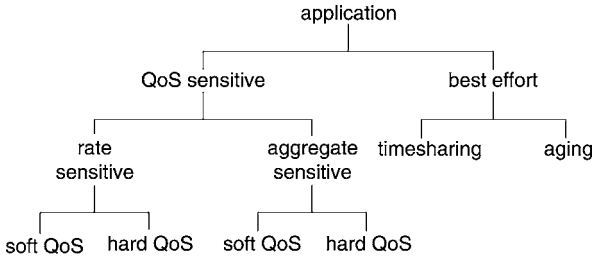

Figure 6. Classification of applications based on synchronization requirements.

Another property of the SQ is the over subscription. QoS provisioning schemes based on admission-control assure the required level of performance by limiting the number of tasks and subtasks such that the capacity is not exceeded at the local resource. SQ, relaxes the capacity constraint by allowing over subscription by an overload factor β, i.e., $\sum \bar{r}_{s_i} < (1 + \beta)$, where $\bar{r}_{s_i}$ is the normalized relative weight of subtask s_i assigned to the local resource and $\beta < 1$.

4.5. Applications suitable for SQ

Traditionally, the Internet was used for running best-effort applications. With the rise of multimedia applications, supporting various classes of QoS is becoming essential. Figure 6 classifies the wide variety of applications that might co-exist in a Grid computing system. This co-existence of the various applications is a key issue to be handled in solving the co-allocation problem. *Rate-sensitive* applications depend on accomplishing a consistent amount of work per unit time throughout their lifetime. *Aggregate-sensitive* applications, on the other hand, are not sensitive to the rate at which progress is made. However they are sensitive to the total progress made at given time intervals.

As illustrated in figure 6, the rate-sensitive and aggregate-sensitive applications can be further classified into two classes namely soft and hard QoS. Hard QoS applications have stringent progress constraints [14] and missing constraints such as a deadline can lead to catastrophic failures. Thus, these applications require deterministic guarantee for their QoS parameters. On the other hand, missing a deadline for soft QoS applications only diminishes the quality of the results and does not lead to catastrophic failures. Thus, these applications require a statistical or probabilistic guarantee for their QoS parameters.

The SQ is flexible in supporting QoS constraints according to the application's needs which are expressed in the two QoS attributes provided, namely *asynchrony* and *overall deviation.* The flexibility of these two QoS attributes make it possible for SQ to accommodate both types, rate-sensitive as well as aggregate-sensitive applications.

5. Simulation results and discussion

5.1. Goals of the simulation

The primary goal of the simulation study was to evaluate the effectiveness of SQ in limiting the *co-allocation skew* among subtasks belonging to a single task that executes in a Grid environment. We compare the performance of SQ with a scheme that is based on strict admission control. Because the purpose is to examine the effectiveness of SQ in reducing the co-allocation skew, we scale the workload with increasing number of machines. The scaling is performed by increasing the number of subtasks per task in direct proportion to the number of machines. As the number of subtasks per task increase, the complexity of the co-allocation problem increases.

In this paper, we present a study where different metrics were used to evaluate the performance of the two algorithms while various parameters were varied. Although *co-allocation skew* is the primary performance metric the SQ algorithm is trying to minimize, our simulation study uses several other performance metrics as well. These metrics include: (a) *acceptance ratio*: the number of requests accepted divided by the total number of submitted requests, (b) *effective cycles delivered*: number of cycles delivered to the applications when they are in the valid QoS-space, and (c) *QoS conformance*: the conformation to the *overall deviation* window provided by the client.

5.2. Simulation setup

In our simulation studies, we partition the CPU of each local machine into three partitions weighted 0.4, 0.4, and 0.2 for local, Grid QoS, and Grid best effort flows, respectively. In the strict admission-control-based scheme, an admission controller at the Grid QoS queue prevents the admission of further tasks once the capacity is reached. Thus preventing over subscription. Whereas, the SQ allows up to 5% over subscription.

The Grid topology was simulated by a set of nodes without any consideration on the inter-connectivity. The size of the Grid topology was varied as (5, 10, 15, 25). The task arrivals were modeled by three task generators corresponding to the three types of task flows. The task generators inject a random number of tasks in the range (1000, 2000) using a Poisson arrival process with inter arrival times (λ) from (10, 100, 200, 500) seconds. For each hard QoS task, the Grid QoS generator associates two values randomly chosen from the uniform distribution (100, 500) seconds as allowable *asynchrony* value and *overall deviation*. Further, a Grid task is composed of several subtasks. The number of subtasks are randomly chosen from the uniform distribution (0, number of local resources) and the execution times of the subtasks are chosen from the uniform distribution (1500, 2000) seconds. The CPU bandwidth of a local machine (CPU_i) is randomly generated from the range (100, 600) MHz and a Grid level standard CPU (CPU_g) is assumed to have a 1 GHz bandwidth. The resource requirement of each task is expressed by choosing a value for the overall task weight m and relative weight r_{s_i} of subtask s_i in the range (1, 5).

5.3. Mean value utilization analysis

In this section, we analyze the different simulation scenarios using the mean value utilization (ρ). The computation of ρ for different simulation scenarios show that the simulation study covers overload as well as underload conditions. It should be noted that a resource is segmented into three partitions by the inter-Queue scheduler and the objective here is to examine the loading situations on the Grid QoS partitions of the resources. Therefore, the analysis below does not consider the loading on the other partitions.

Because the tasks or subtask do not communicate with each other in our model, the network loading is not considered here. We model the Grid as a collection of nodes. For simplicity, this analysis considers homogeneous processing nodes.

We select an arbitrary time interval (100 s in this study) and determine *the total CPU demand* (Υ) and the *total CPU cycles available* (Ψ) during this interval. Then, the mean value utilization is defined as:

$$\rho = \Upsilon / \Psi. \tag{8}$$

Each resource dedicates m_{GQoS} percent of its resource to Grid QoS tasks. Therefore, over the given time interval of 100 s, $m_{\text{GQoS}} \times N_{\text{CPUs}}$ worth of CPU cycles are available, where N_{CPUs} is the number of CPUs available in the whole system. Hence, Υ is given by the product of the total number of tasks and the average CPU demand per task. Because the inter arrival time is λ, the total number of tasks in the given interval of 100 s is $100/\lambda$. As mentioned previously, the CPU requirement of a task is expressed with respect to a standard CPU running at 1 GHz. Let $M(x)$ denote the mean value of variable x. Then average CPU requirement, with respect to the standard CPU, of a task is given by $M(m) \times M(\tau)$, where τ is the duration of the task.

The CPU requirement expressed with respect to a local CPU is given by $M(m) \times M(\tau) \times M(d)$. Therefore, the total average CPU demand is given by:

$$\Upsilon = \frac{100}{\lambda} \times M(m) \times M(\tau) \times M(d). \tag{9}$$

With the above values for Υ and Ψ,

$$\begin{aligned}\rho &= \frac{100 \times M(m) \times M(\tau) \times M(d)}{\lambda \times m_{\text{GQoS}} \times N_{\text{CPUs}}} \\ &= \frac{100 \times (3/100) \times 1750 \times (100/35)}{\lambda \times 40 \times N_{\text{CPUs}}} \\ &= \frac{375}{\lambda \times N_{\text{CPUs}}}.\end{aligned}$$

From table 2, it can be observed that the simulation performed for $\lambda = 10$ s overloads the system and the rest of the simulations under loads the system. This result is expected because at $\lambda = 10$ s the tasks are arriving at a much faster

Table 2
Mean value utilization of the different number of machines as λ increases.

λ (s)	Number of CPUs (N_{CPUs})			
	5	10	15	25
10	$\rho = 750\%$	$\rho = 375\%$	$\rho = 250\%$	$\rho = 150\%$
100	$\rho = 75\%$	$\rho = 37.5\%$	$\rho = 25\%$	$\rho = 15\%$
200	$\rho = 37.5\%$	$\rho = 18.75\%$	$\rho = 12.5\%$	$\rho = 7.5\%$
500	$\rho = 15\%$	$\rho = 7.5\%$	$\rho = 5\%$	$\rho = 3\%$

rate and the system will experience an increased level of demand for the fixed amount of resources. As λ increases to 500, the task arrivals get sparser with a corresponding lessening of the imposed load. However, it should be noted that the mean value utilization numbers provided here indicate the demand versus arbitrarily ratio before the admission control. Table 2 shows that when $\lambda = 10$ s, ρ is over 100% and that ρ decrease as the number of machines increase. As explained in section 4.2.2, a task t's weight equivalent is distributed among its subtasks. In the simulation, we increase the number of subtasks with the increase of the number of machines. Hence, as the number of machines increase, task t's weight is dissipated among larger number of subtasks and the load on the individual machines decreases.

5.4. Results and discussion

The SQ is compared with a QoS 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 conforming to the overall deviation and asynchrony windows, respectively.

Figures 7 (a) and (b) show the variation of the average co-allocation skew with the number of machines. Figure 7(a) shows the variation for the overloaded situation where $\lambda = 10$ s and figure 7(b) shows the variation for the underloaded situation where $\lambda = 500$ s. In these experiments, we increase the number of subtasks per task in direct proportion to the number of machines, i.e., as the number of machines increase, the co-allocation workload also increases. This is done because the intention of this study is to examine the effectiveness of SQ in reducing the co-allocation skew. From the results, we can observe that SQ significantly outperforms the admission-control-based QoS scheme. One interesting observation from the results is the increase in average co-allocation skew as λ increases. This can be understood by noting (a) both schemes use admission control and SQ relaxes the admission control to increase the number of admitted tasks and (b) the average co-allocation skew is computed over the tasks successfully completing their execution.

The admission control process shapes the request stream such that the request stream that reaches the actual system is within the system's capacity (i.e., the system is subjected

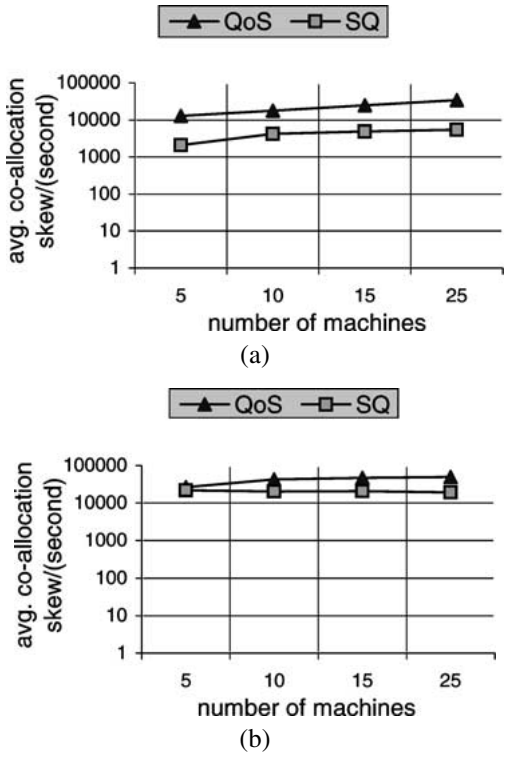

Figure 7. Variation of co-allocation skew with number of machines for λ (a) 10 and (b) 500 s.

to overload). However, compared to the underloaded condition, the system will complete a significantly larger number of tasks in the overloaded condition. As λ increases, the co-allocation skew of the QoS-based approach increases. The major contributor of this increase is the under loading of different resources that causes certain subtasks of a given task to speed away. Thus increasing the co-allocation skew. For the SQ algorithm too the co-allocation skew increases as λ increases. This can be explained by examining the operation of SQ. The SQ reduces the co-allocation skew by readjusting the weights of the different subtasks and possibly launching a "dummy" subtask to soak up additional resources that can increase co-allocation skew if they are delivered to actual subtasks. However, this is a feedback process and takes several scheduling cycles to stabilize at an operating point. When $\lambda = 10$ s, subtasks arrive in a burst and the system reconfigures once for that "batch" of subtasks. When $\lambda = 500$ s, the subtasks arrive in a sparse fashion and the system is in continuous reconfiguration. The co-allocation skew is minimal only when the system has reached a stable state. This explains why the SQ is not performing well for the underloaded situation although there exists room for adjustments.

Figures 8 and 9 show that QoS conformance and effective cycles delivered are much higher for SQ. As λ 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 λ. As the co-allocation skew increases, the conformance to the *asynchrony* and the *overall deviation* is violated. This violation causes QoS conformance

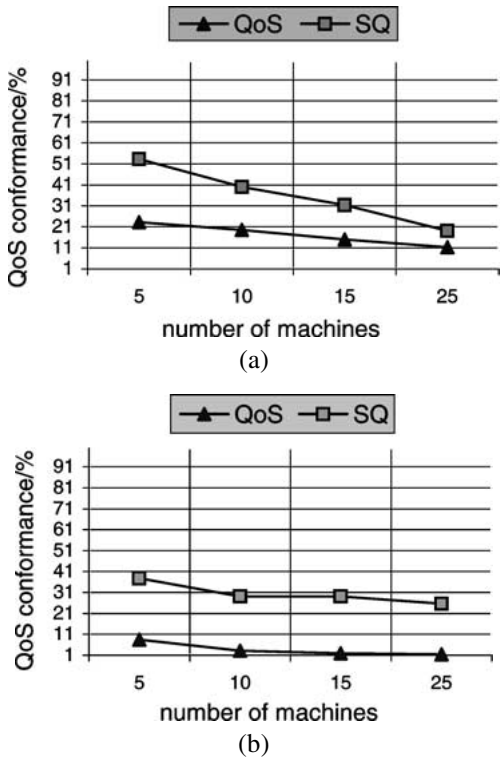

Figure 8. QoS conformance with number of machines for λ (a) 10 and (b) 500 s.

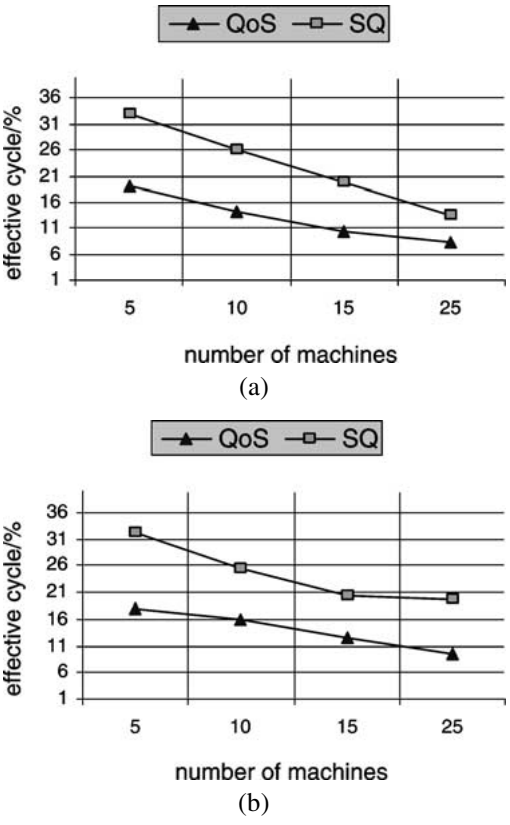

Figure 9. Effective cycles delivered for number of machines for λ (a) 10 and (b) 500 s.

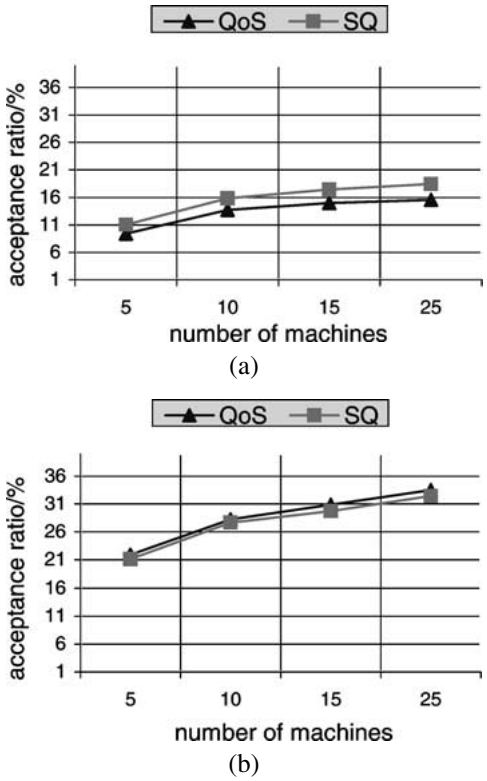

Figure 10. Acceptance ratio with number of machines for λ (a) 10 and (b) 500 s.

and effective cycles delivered to decrease as the co-allocation skew increases. Because SQ uses a relaxed admission control, it can be expected to have a higher acceptance ratio. This is supported by the acceptance ratio shown in figure 10(a) for $\lambda = 10$ s.

However, figure 10(b) shows a seemingly contradictory result that can be explained as follows. When $\lambda = 500$ s, as shown in table 2, the system is underloaded and this provides the different subtasks of a task the opportunity to speed away from the others. With the SQ, the subtasks are forcefully synchronized such that they proceed at the speed of the slowest subtask. This results in a subtask taking longer to complete its execution at the assigned machine. Consequently, less number of new tasks can be admitted. When $\lambda = 10$ s, the system is overloaded and there are very few opportunities for subtasks to speed away. As a result, SQ is less intrusive in slowing down the fast subtasks. Consequently, the over subscription allowed by the SQ dominates the overall acceptance ratio.

6. Related work

Advance reservation is one mechanism that is widely used for supporting co-allocation. The *Globus Architecture for Reservation and Allocation* (GARA) [5] is one system supports co-allocation using advance reservations. Another system using the same approach is Tenet real-time protocol suite [3]. The SQ is different from GARA and Tenet in several ways. Our approach addresses the co-allocation without the need for ad-

Table 3
Summary of co-allocation schemes.

Properties of example co-allocation schemes						Example schemes
Supported applications	Supported node	Supported resource(s)	Schedule scheme used	Problem addressed	Advanced reservation	
Best-effort and QoS	distributed	diverse	hierarchical	co-allocation	required	Tenet suite 2
Best-effort	distributed	CPU	application based	co-allocation	not required	Implicit co-scheduling
Best-effort	distributed	diverse	central	co-allocation	not required	Ensemble scheduling
Best-effort and QoS	distributed	diverse	hierarchical	co-allocation	required	GARA

vance 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 another 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. Unlike SQ, the implicit co-scheduling is targeted towards message passing sub-tasks. 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.

Ensemble scheduling [13] is a mechanism that favors jobs that require multi-site (ensemble) execution. The ensemble-aware scheduler tracks the execution of ensemble applications and from prior executions determines which resources are favored by such applications. It uses this information to increase priority of ensemble applications. This boost in priority essentially soft reserves the resources for ensemble applications by pushing the other applications away from these resources.

To conclude this section, we summarize example co-allocation schemes that address the co-allocation of resources. These schemes are presented along with their properties which are classified into 6 categories as shown in table 3.

7. Conclusions

This paper addresses the co-allocation issue in Grid computing environments. 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 such as the Grid. In such a system, 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 autonomy makes co-allocation a challenging problem. This paper proposes a novel scheme for co-allocation in Grid computing systems called the SQ algorithm.

Furthermore, 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 in the previous schedule cycles, (b) environment-aware QoS capability, where SQ assures that the aggregated work accomplished by a subtask does not fall behind the other subtasks belonging to the same task. These other subtasks may be running on different 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.

Several aspects of SQ need further examination. Some of these include (a) analysis of the impact of sensing delay on the stability of the scheme, (b) decentralizing the Grid-level co-allocation controller so that scalability can be improved, and (c) devising fair schemes for imposing corrective actions when there are conflicting requirements.

In summary, this paper presents an architecture and algorithm for performing co-allocation without the need for ad-

vance reservations. This can be implemented without modifying the local operating systems, i.e., the proposed queuing mechanisms can be implemented at the application level with pseudo real-time task dispatchers. Simulations studies indicate the effectiveness of the approach as compared to strict admission control-based approach.

Acknowledgments

This research is partially supported by a Natural Sciences and Engineering Research Council of Canada Research Grant RGP220278 and equipment used was supported by a Canada Foundation for Innovation Grant. A preliminary version of this paper appeared in the *13th IASTED International Conference on Parallel and Distributed Computing Systems (PDCS '01)*.

References

[1] A.C. Arpaci-Dusseau, D.E. Culler and A.M. Mainwaring, Scheduling with implicit information in distributed systems, in: *ACM SIGMETRICS Conference on 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, in: *4th Workshop on Job Scheduling Strategies for Parallel Processing* (March 1998) pp. 62–82.

[3] D. Ferrari, A. Gupta and G. Ventre, Distributed advance reservation of real-time connections, ACM/Springer-Verlag Journal on Multimedia Systems 5(3) (1997) 187–198.

[4] I. Foster and C. Kesselman, eds., *The Grid: Blueprint for a Future Computing Infrastructure* (Morgan Kaufmann, San Francisco, USA, 1999).

[5] 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, in: *International Workshop on Quality of Service* (June 1999) pp. 27–36.

[6] I. Foster, C. Kesselman and S. Tuecke, The anatomy of the Grid: Enabling scalable virtual organizations, International Journal on Supercomputer Applications 15(3) (2001) 200–222.

[7] P. Goyal, X. Guo and H. Vin, A hierarchical CPU scheduler for multimedia operating systems, in: *2nd USENIX Symposium on Operating Systems Design and Implementation (OSDI '96)* (October 1996) pp. 107–122.

[8] P. Goyal, H.M. Vin and H. Cheng, Start time fair queuing: A scheduling algorithm for integrated services packet switching networks, in: *ACM SIGCOMM '96* (August 1996) pp. 157–168.

[9] J. Nieh and M. Lam, The design, implementation, and evaluation of SMART: A scheduler for multimedia applications, in: *16th ACM Symposium on Operating Systems Principles (SOSP '97)* (October 1997) pp. 184–197.

[10] P.K. Sinha, *Distributed Operating Systems: Concepts and Design* (IEEE Press, New York, 1997).

[11] C.A. Waldspurger and W.E. Weihl, Lottery scheduling: Flexible proportional-share resource management, in: *1st USENIX Symposium on Operating System Design and Implementation (OSDI '94)* (November 1994) pp. 1–12.

[12] C.A. Waldspurger and W.E. Weihl, Stride scheduling: Deterministic proportional-share resource management, Technical Report, MIT Laboratory for Computer Science, MIT, Cambridge (June 1995).

[13] J. Weissman and P. Srinivasan, Ensemble scheduling: Resource co-allocation on the computational Grid, in: *2nd International Workshop on Grid Computing (Grid '01)* (November 2001) pp. 87–98.

[14] D. Yau, ARC-H: Uniform CPU scheduling for heterogeneous services, in: *International Conference on Multimedia Computing and Systems*, Vol. 2 (June 1999) pp. 127–132.

[15] D. Yau and S.S. Lam, Adaptive rate controlled scheduling for multimedia applications, IEEE/ACM Transactions on Networking 5(4) (August 1997) 475–488.

[16] L. Zhang, A new traffic control algorithm for packet switching networks, IEEE Transactions on Computer Systems 9(2) (May 1991) 101–124.

Farag Azzedin is a Ph.D. candidate at the Computer Science Department at the University of Manitoba, Canada. From January 1986 to August 1991, he was attending the University of Victoria, Canada where he received a B.Sc. degree in computer science. From 1991 to the end of 1998 he worked with the Ministry of Health, B.C. Canada and the city of Vancouver, B.C. Canada as a computer programmer/analyst and a data analyst, respectively. He received an M.S. degree in computer science in 2001 from the University of Manitoba, Canada. His research is supported by a fellowship from the University of Manitoba as well as a fellowship from TR-Labs, a Canadian research consortium in information and communications technology. His research interests include Grid computing, trust modeling and its application in peer-to-peer computing systems, and resource management in distributed systems. He has coauthored more than 10 technical papers in these and related areas.
E-mail: fazzedin@cs.umanitoba.ca

Muthucumaru Maheswaran is a joint Assistant Professor in the School of Computer Science and the Department of Electrical and Computer Engineering at McGill University, Canada. From August 1998 to December 2002, he was an assistant professor in the Department of Computer Science at the University of Manitoba, Canada. In 1990, he received a B.Sc. degree in electrical and electronic engineering from the University of Peradeniya, Sri Lanka. He received an M.S. degree in electrical engineering in 1994 and a Ph.D. degree in electrical and computer engineering in 1998, both from the School of Electrical and Computer Engineering at Purdue University. He held a Fulbright scholarship during his tenure as an MSEE student at Purdue University. His research interests include autonomic computing, Grid computing, peer-to-peer computing, trust modeling and management in large-scale networked systems, and scalable resource management systems. He has authored or coauthored more than 50 technical papers in these and related areas.
E-mail: maheswar@cs.mcgill.ca

Neil Arnason is a Full Professor in the Computer Science Department at the University of Manitoba. His Ph.D. (Edinburgh, 1971) was in population modeling and estimation. His research interests are mainly in animal population models and survey techniques and in performance analysis and simulation methods applied to computer and network systems.
E-mail: arnason@cs.umanitoba.ca