Offer-based Scheduling of Deadline-Constrained Bag-of-Tasks Applications for Utility Computing Systems

Marco A. S. Netto and Rajkumar Buyya
Grid Computing and Distributed Systems Laboratory
Department of Computer Science and Software Engineering
The University of Melbourne, Australia
{netto, raj}@csse.unimelb.edu.au

Abstract

Metaschedulers can distribute parts of a Bag-of-Tasks (BoT) application among various resource providers in order to speed up its execution. When providers cannot disclose private information such as their load and computing power, which are usually heterogeneous, the metascheduler needs to make blind scheduling decisions. We propose three policies for composing resource offers to schedule deadline-constrained BoT applications. Offers act as a mechanism in which resource providers expose their interest in executing an entire BoT or only part of it without revealing their load and total computing power. We also evaluate the amount of information resource providers need to expose to the metascheduler and its impact on the scheduling. Our main findings are: (i) offer-based scheduling produces less delay for jobs that cannot meet deadlines in comparison to scheduling based on load availability (i.e. free time slots); thus it is possible to keep providers' load private when scheduling multi-site BoTs; and (ii) if providers publish their total computing power they can have more local jobs meeting deadlines.

1. Introduction

Bags-of-Tasks (BoTs) are parallel applications with no inter-task communication. A variety of problems in several fields, including computational biology [18], image processing [23], and massive searches [3], have been modeled as BoT applications. In comparison to the message passing model, BoT applications can be easily executed on multiple resource providers to meet a user deadline or reduce the user response time. Although BoT applications comprise independent tasks, the results produced by all tasks constitute the solution of a single problem. In most cases, users need the whole set of tasks executed to be able to post-process or analyze the results. Therefore, the optimization of the aggregate set of results is important, and not the optimization of a particular task or group of tasks [4].

Large-scale parallel applications have been deployed in computing facilities such as TeraGrid, DAS-3, Grid'5000, and NAREGI. However, the academic and industry communities have also been considering the utility computing paradigm for executing these applications on environments such as Sun Grid, IBM On-Demand Computing, and Amazon Elastic Compute Cloud (EC2) [9]. These environments provide access to resources by charging users according to the application's demand, which is specified in contracts called Service Level Agreements (SLAs).

The execution of a BoT application on multiple utility computing facilities is an attractive solution to meet user deadlines. This is because more tasks of a single BoT application can execute in parallel and these facilities have to deliver a certain QoS level, otherwise the providers are penalized. A service provider containing a metascheduler is responsible for distributing the tasks among resource providers according to their load and system configuration. However, allocating resources from multiple providers is challenging because these resource providers cannot disclose much information about their local load to the metascheduler. Workload is private information that companies do not disclose easily since it may affect the business strategy of competitors.

Much work has been done on scheduling BoTs [7, 15, 16]. However, little effort has been devoted to schedule these applications with deadline requirements [6, 14, 24], in particular considering *limited load information available* from resource providers. Therefore, we extend the existing solutions and contribute to the research field in the following ways:

- We introduce three policies for composing offers to schedule BoT applications (Section 4). Offers are a mechanism in which resource providers expose their

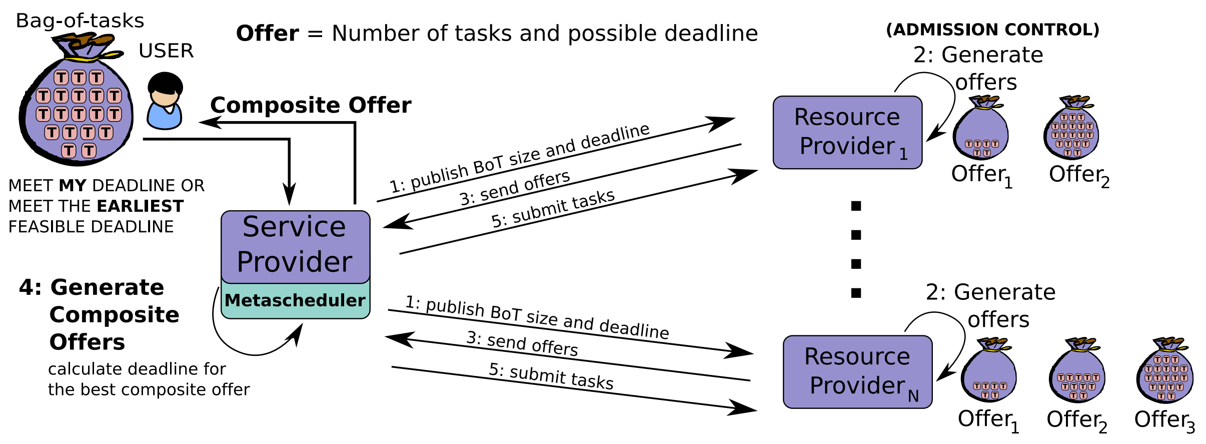

Bag-of-tasks
USER
Offer = Number of tasks and possible deadline
Composite Offer
MEET MY DEADLINE OR MEET THE EARLIEST FEASIBLE DEADLINE
Service Provider
Metascheduler
4: Generate Composite Offers
calculate deadline for the best composite offer
1: publish BoT size and deadline
3: send offers
5: submit tasks
(ADMISSION CONTROL)
2: Generate offers
Resource Provider 1
Offer1
Offer2
Resource Provider N
Offer1
Offer2
Offer3

Figure 1. Components interaction for scheduling a bag-of-tasks on multiple resources with offers.

interest in executing an entire BoT or only part of it without revealing their local load and system capabilities. Whenever providers cannot meet a deadline, they generate offers with another feasible deadline. For the offer generation within resource providers (Section 3), we leverage the work developed by Islam et al. [12,13], whereas the concept of combining offers for executing an application on multiple resource providers is inspired by the provisioning model of Singh et al. [22] and the capacity planning of Siddiqui et al. [21].

- We investigate the amount of information resource providers need to expose to the metascheduler and its impact on the scheduling of jobs local and external to the resource providers (Section 5).

Even though the main motivation of this work is scheduling deadline-constrained BoTs on utility computing facilities, the policies can also be used in other scenarios. Therefore, we have not included pricing and economic models in this work. Note that our study can be used for other applications rather than Bag-of-Tasks. Services running on multiple resource providers also require a coordinated allocation based on the load information access and services' demands. Another example is the scheduling of parallel phases of workflows, in which a group of tasks have to finish, so as the workflow can proceed to the next stage.

2. Architecture and Scheduling Policies

A metascheduler receives user requests to schedule BoTs on multiple autonomous resource providers in on-line mode. Users provide the number of tasks in the bag, their estimated required time, and a deadline to execute the entire BoT. The resources considered are space-shared machines such as clusters and massively parallel processing machines. Resource providers are responsible for scheduling both local and external jobs using the Earliest Deadline First. The local jobs can be both sequential and message passing parallel applications, whereas the external jobs are the BoT applications. The metascheduler has no access to the scheduling queues.

As illustrated in Figure 1, the scheduling of a BoT application consists of 5 steps. In step 1, the metascheduler is responsible for exposing the application requirements to the resource providers. In step 2, the resource providers generate a list of offers that can serve the entire BoT or only part of it. Once the resource providers generate the offers, they send them to the metascheduler (step 3), which composes them according to the user requirements (step 4), and submits the tasks to resource providers (step 5).

Scheduler's Goal. Our goal is to meet users' deadlines, and when not possible, try to schedule jobs as close as possible to these deadlines. The challenge from the metascheduler's point of view is to know how much work to submit to each resource provider such that it meets the BoT user's deadline, whereas from the resource providers' point of view is to know how much work they can admit without violating the deadlines of already accepted requests.

3. Offer Generation

3.1. Deadline-aware offer generation

An offer consists of a number of tasks, and the maximum time the resource provider can complete the work. These offers are to execute part or the entire BoT. The resource provider could follow different policies to generate a list of offers. For instance, a resource provider could generate offers that (i) are more profitable [11,19]; (ii) provide some slack in case of resource failures or to increase the chances of admitting more jobs in future; or (iii) do not violate the deadline of already scheduled tasks and the new task. In this work, we support the third approach and will leave the other two for future work.

The offer generation uses the BoT information provided by the metascheduler, which includes estimated execution time of each task, number of tasks, and deadline. Different from scheduling based on FIFO with conservative backfilling for example, where it is possible to identify time slots by simply calculating the start and completion time of tasks, an Earliest Deadline First based queue cannot follow such an approach. The reason is that the offer generation involves the rescheduling of the already accepted jobs, and hence the free time slots depend on the new submitted job.

In order to generate the list of offers Φ, the resource provider:

1. Defines a set of possible number of tasks Δ it is willing to accept. It creates this list by calculating a percentage of the total number of tasks in the BoT application (e.g. $\Delta \leftarrow \{BoT^s, 0.75 * BoT^s, 0.50 * BoT^s, 0.25 * BoT^s, 0.10 * BoT^s\}$, where s is the BoT size, i.e. total number of tasks).

2. A procedure *genOffer* generates an offer for each BoT size. To generate an offer, the resource provider creates a temporary job j_k that has the BoT specifications, which include deadline and estimated execution time, but with the different number of tasks, defined in Δ. The algorithm used is the Earliest Deadline First.

3. The *genOffer* procedure returns the completion time of that offer. This time can be the deadline provided by the user (in the case of a successful schedule), or a completion time that is longer than the deadline (when the scheduler cannot meet the user deadline).

4. The algorithm compares the current offer with the previous offer in Φ. If the completion time is the same, the previous offer simply has its number of tasks increased. If the completion time is longer, the resource provider includes the current offer in the list Φ.

FEEDBACK. In order to provide users with feedback when it is not possible to meet their deadlines, we use the approach proposed by Islam et al. [13]. The idea is to use a binary search that has as its first point the deadline defined by the user and its last point as the longest feasible deadline, i.e. the one when the job is placed in the last position of the scheduling queue.

3.2. Load-aware offer generation

As base for comparison in our evaluations, we will use a policy based on free time slots presented by Singh et al. [22] (**FreeTimeSlots** policy). In this policy, the resource provider uses the current schedule containing running and pending jobs. The resource provider generates windows for each processor that represent their time availability, also known as *time slots*. Different from Singh et al. [22], we analyze and provide the time slots for the entire scheduling queue, not only part of it.

4. Offer Composition

The metascheduler is responsible for composing the offers from resource providers and giving the user a single offer with a feasible deadline that can be met. The offer composition determines how much work the metascheduler should send to each resource provider; and when these tasks should receive the resources. The goal of the metascheduler is to meet their users' deadlines, or get as close as possible to the deadlines.

Once the metascheduler receives the offers from resource providers, it analyses whether it is possible to meet user's deadline. We have developed one policy for when the user's deadline cannot be met (*OffersNoLB*), and two policies when it is possible to meet the user's deadline (*OffersWithPLB* and *OffersWithDBLP*). These last two policies try to balance the load distributed among the resource providers according to the information available to the metascheduler.

4.1. When user's deadline cannot be met

For the **OfferNoLB** policy, after collecting the offers from resource providers, the metascheduler:

1. Creates a list with all the offers.

2. Sorts the list such that all offers that meet the user's deadline come before those that do not meet. For those that meet, the offers are sorted in decreasing order of number of tasks. For those that do not meet, the offers are sorted by the ascending order of completion time.

3. Removes all offers after the first offer that is able to execute the entire BoT.

4. Creates a list L that is dynamically updated with possible composite offers. The creation of L is based on the order of the offers. For each offer analyzed, the algorithm updates the number of remaining requested tasks and the last completion time of each list that uses that offer. Note that what makes this algorithm simple is the list pre-processing, i.e. sorting and filtering.

5. Returns the first composite offer in L that provides the earliest possible deadline.

Table 1. Example of offer composition with the *OfferNoLB* policy for a BoT with number of tasks = 512 and deadline = 40 time units.

Operation	Offers (size, deadline)$_{\text{provider}}$
Original Offers	(256, **40**)$_1$, (512, 100)$_1$, (128, **40**)$_2$, (512, 200)$_2$, (64, **40**)$_3$, (512, 200)$_3$
Sorted Offers	(256, **40**)$_1$, (128, **40**)$_2$, (64, **40**)$_3$, (512, 100)$_1$, (512, 200)$_2$, (512, 200)$_3$
Filterd Offers	(256, **40**)$_1$, (128, **40**)$_2$, (64, **40**)$_3$, (512, 100)$_1$
Composite Offers (List L)	{(128, **40**)$_2$, (64, **40**)$_3$, (320, 100)$_1$} or {(256, **40**)$_1$, (128, **40**)$_2$, (64, **40**)$_3$ (not enough tasks)}
Selected Composite Offer	(128, **40**)$_2$, (64, **40**)$_3$, (320, 100)$_1$

Table 1 illustrates an example on how the metascheduler composes offers. The example considers a list of offers $\Phi = \{(s_1, d_1)_{rid_1}, ..., (s_n, d_n)_{rid_n}\}$ where rid is the resource provider id, s is the BoT size, and d is the offer's deadline, and a BoT with deadline = 40 time units and number of tasks = 512. As we can observe, the choice of the offers is not greedy. From the example, the metascheduler does not use the offer *(256, 40)*$_1$, which is the best offer, because the remaining resource providers would end up completing the BoT by 200 time units rather than 100 time units, which is the next best offer from resource provider 3 that can accept enough tasks ((512, 200)$_3$).

4.2. Balancing the load when possible to meet user's deadline

When it is possible to meet the user's deadline, the metascheduler tries to balance the number of tasks to be submitted to each resource provider. This balance allows jobs local to resource providers to meet more deadlines. The policy for load balancing depends on the amount of information the metascheduler has about the resource providers.

We have developed two policies for load balancing: (i) **OffersWithPLB** (Proportional Load Balancing) balances the load according to the size of the offers; and (ii) **OffersWithDPLB** (Double Proportional Load Balancing) balances the load according to the size of the offers *and* the total computing power of resource providers. These policies work only with offers that meet user's deadlines; all the other offers are discarted. Therefore, each resource provider has only one offer that meets the deadline.

The *OffersWithPLB* policy uses the offer size in order to balance the number of tasks submitted to each resource provider. The proportional parameter P is calculated per offer as follows: $P \leftarrow$ *OfferSize / totalOfferedTasks*, where *OfferSize* is the number of tasks in an offer, and *totalOfferedTasks* is the total number of tasks from all offers. For each offer, the number of tasks is multiplied by P. As the metascheduler does not know the load and the total computing power of resource providers, the offer size serves as an indicator of how much work a resource provider should receive in relation to the others.

Algorithm 1: Pseudo-code for composing offers using the *OffersWithDPLB* policy.

```
1  Sort offers by decreasing order of their size (offers of same
   size are sorted by decreasing order of resource provider's
   total computing power)
2  for each offer size in offers list do
3      P ← totalTasksOfferedThisSize / totalOfferedTasks
4      remainingNTasksSameSize ← P * BoTSize
5      totalCPower ← total RPs' computing power of offers of
       this size
6      for all offers with this size do
7          if last offer from this group then
8              nTasks ← remainingNTasksSameSize
9          else
10             nTasks ← P * BoTSize
11             DP ← offer.RPCPower / totalCPower
12             nTasks ← DP * nTasks
13             if nTasks > offer.nTasks then
14                 nTasks ← offer.nTasks
15         offer.setNTasks(nTasks)
16         remainingNTasksSameSize.decrement(offer.nTasks)
17         compositeoffer.add(offer)
```

For the *OffersWithDPLB* policy (Algorithm 1), the parameter P is calculated by the group of offers with the same number of tasks (Line 3). The additional parameter *Double Proportional (DP)* is used to distributed the load to a given group of resource providers that contains offers with the same number of tasks (offer size) according to their capabilities (e.g. total number of resources a provider hosts) (Line 11). It is not always possible to distributed the tasks of a given offer size exactly proportionally to the resource capabilities. For this reason, we sort the offers of the same size in a decreasing order of providers' total computing power (Line 1) and we adjust the number of tasks to the resource provider when necessary (Lines 13-14).

Table 2. Summary of workloads used to perform the experiments.

Location	Trace	Procs	Jobs	Load	Job Req Procs	Job Req Time
Cluster 1	CTC SP2v2.1	430	3,478	49%	$1 \leq$ procs ≤ 306	1h $\leq$ time $\leq$ 18h
Cluster 2	HPC2N v1.1	240	959	56%	$1 \leq$ procs ≤ 128	1h $\leq$ time $\leq$ 120h
Cluster 3	HPC2N v1.1	240	4,913	48%	$1 \leq$ procs ≤ 128	1h $\leq$ time $\leq$ 120h
Cluster 4	SDSC SP2 v3.1	128	1,088	54%	$1 \leq$ procs ≤ 115	1h $\leq$ time $\leq$ 18h
Cluster 5	LPC-EGEE v1.2	140	6,574	52%	$1 \leq$ procs ≤ 1	2h $\leq$ time $\leq$ 72h
External	SDSC BLUE v3.1	1178	969	54%	$64 \leq$ procs ≤ 632	1h $\leq$ time $\leq$ 36h

5. Evaluation

We have evaluated the scheduling policies by means of simulations to observe their effects in a long-term usage. Simulations have allowed us to perform repeatable and controllable experiments. We have used our event-driven simulator, named PaJFit (Parallel Job Fit) [17], which we have extended to support BoTs on multi-site environments. We have used real traces from supercomputers available at the Parallel Workloads Archive[1] and extended them according to our needs.

We have evaluated the following scheduling policies:

- **FreeTimeSlots:** scheduling based on free time slots, i.e. the metascheduler has a detailed access to the load available (Section 3.2);
- **OffersWithPLB:** scheduling based on offers. The scheduler composes offers based on their sizes (Section 4.2);
- **OffersWithDPLB:** the metascheduler considers offer sizes and the total computing power of resource providers (Section 4.2);
- **OffersWithDPLBV2:** an extension of *OffersWithDPLB* in which the metascheduler has access to the resource providers' load to be processed (not the free time slots). This information is used in the same way to calculate the parameter DP described in Section 4.2;

For the offer-based policies, i.e. *OffersWithPLB*, *OffersWithDPLB*, and *OffersWithDPLBV2*, when no offer can meet the user deadline, the metascheduler follows the **OffersWithNoLB** policy described in Section 4.1.

5.1. Experimental configuration

TRACES. We have modeled an environment composed of five clusters with their own schedulers and loads, and one metascheduler that receives external (BoT) jobs that can be executed in either a single or multiple clusters. For the local jobs, we have used the traces: 430-node IBM SP2 from The Cornell Theory Center (CTC SP2v2.1), 240-procs AMD Athlon MP2000+ from High-Performance Computing Center North (HPC2N v1.1) in Sweden, the 128-node IBM SP2 from The San Diego Supercomputer Center (SDSC SP2 v3.1), and the 70 dual 3GHz Pentium-IV Xeons from LPC Clermont-Ferrand in France (LPC-EGEE v1.2). We have used two parts of the trace HPC2N, from different years, to simulate two clusters. For the external jobs, we have used the trace of a bigger machine from the San Diego Supercomputer Center Blue Horizon with 1,152 processors: 144-node IBM SP, with 8 processors per node, considering jobs requiring at least 64 processors (SDSC BLUE v3.1). We have simulated 60 days of these traces.

LOAD. Regarding the load used in the resource providers, approximately 50% comes from the multi-site BoTs (*external load*), which could be executed in any cluster or multiple clusters, and approximately 50% comes from users submitting parallel or sequential jobs directly to a particular cluster (*local load*). The *global load* is therefore the external load plus the local load submitted to the clusters. We have chosen the same load for local and external loads in order to be able to compare the impact of the scheduling policies on local and external jobs in a fair manner. We were able to vary the loads using a strategy similar to that described by Shmueli and Feitelson to evaluate their backfilling strategy [20], in which they modify the jobs' arrival time. However, we fixed the simulation time interval and modified the number of jobs in the traces. Table 2 summarizes the workload characteristics. More details on the workloads can be found at the Parallel Workloads Archive.

DEADLINES. To the best of our knowledge, there are no traces available with deadlines. Therefore, we have incorporated deadlines in the existing traces using the following function: T_j^s+T_j^r+k, where T_j^s is the job submission time, T_j^r is the job estimated runtime, and k is a parameter that assumes three values according to two **Deadline Schemas**. For *Deadline Schema 1*, k assumes the values 18 hours, 36 hours, and 10 days, and for *Deadline Schema 2*, k assumes the values 12 hours, 1 day, and 1 week. Therefore, Deadline

[1]Parallel Workloads Archive:
http://www.cs.huji.ac.il/labs/parallel/workload

Schema 2 has more jobs with tighter deadlines than Deadline Schema 1. We have used a uniform distribution for the values of k for all jobs in each workload. Note that k is not a function of job size. Modeling k independently of job size allowed the environment to have both small and big jobs with relaxed and tight deadlines. We have generated 30 workloads for each original trace varying the seed for the deadlines. By having 60 days of simulated time, 30 workloads with different deadlines, and 2 deadline schemas, we believe that we have been able to evaluate the policies under various conditions.

METRICS. We have assessed five metrics:

1. *Jobs Delayed:* number of jobs that were not able to meet their deadlines;
2. *Work Delayed:* amount of work (processors x execution time) of the jobs that were not able to meet their deadlines;
3. *Total Weighted Delay:* weighted difference between jobs' deadlines and their new deadline given by the system;

$$TWDelay = \sum_{D_j^N > D_j} R_j * \left(\left(\frac{D_j^N - T_j^s}{D_j - T_j^s} \right) - 1 \right) * 100 \quad (1)$$

   Where D_j is the job deadline, D_j^N is the new job deadline provided by the system, R_j is the number of tasks of the job, and T_j^s is the job submitted time. We used the value R_j as the weight for the percentage difference between the time of the original and new deadline.

4. *Clusters per BoT:* number of clusters used by BoTs;
5. *System Utilization:* global system utilization.

For utility computing environments, the first two metrics represent the loss in revenue due to possible rejections, whereas the third metric could represent penalties for not meeting user's demand. The last two metrics allow us to verify how jobs are spread across the clusters and the impact of the policies on the system utilization.

GOAL. Assess the impact of the information available to the metascheduler for scheduling deadline-constrained jobs.

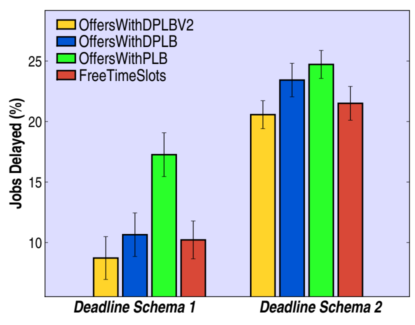
5.2. Results and analysis

The graphs we show in this section contain the averages of 30 simulation runs, each with different workloads, along with their standard deviations. We show the results for the external load, local load and both together since external and local load have different characteristics. We first analyze the jobs that are not able to meet their deadlines when submitted to the system. Figures 2 and 3 represent number of jobs and their respective amount of work (time x number of tasks) delayed in relation to the total number of jobs in the system and total amount of work respectively.

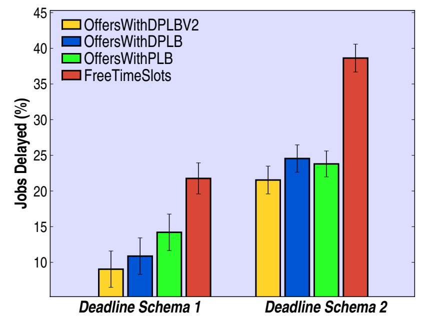
Deadline Tightness. We observe that the difference in the results among the offer-based policies increases when jobs have more relaxed deadlines (Schema 1 has more relaxed deadline jobs than Schema 2). That is because as jobs have more relaxed deadlines in Schema 1, resource providers can reschedule more jobs in order to generate better offers. Therefore, the metascheduler has more options to schedule BoT applications. When it is not possible to meet a user deadline, which is the more frequent in Schema 2, the load balancing policies cannot be used. The metascheduler has to use the *OffersWithNoLB* policy (Section 4.1) for most of the jobs.

Information access. In relation to the differences between the *FreeTimeSlots* policy and offer-based policies we observe that the former policy handles local jobs better or similar to offer-based policies. That is because the metascheduler has more detailed load information using the *FreeTimeSlots* policy, and hence it can better distribute the load among resource providers. As local jobs do not have the option to choose the resource providers, they enjoy more benefits using this policy. We observe that the *OffersWithDPLBV2* policy can generate similar results as *FreeTimeSlots* for local jobs. This happens because in *OffersWithDPLBV2*, the metascheduler has rough access to the local loads, which is enough to balance the load and give local jobs equal opportunity. Finally, we observe that for these two metrics, having access to the providers' total computing power is enough to provide as good results as the *FreeTimeSlots* policy, in which the metascheduler has a detailed access to the resource providers' loads, i.e. the free time slots. If rough load information is also available (*OffersWithDPLBV2*), it is possible to get even better results.

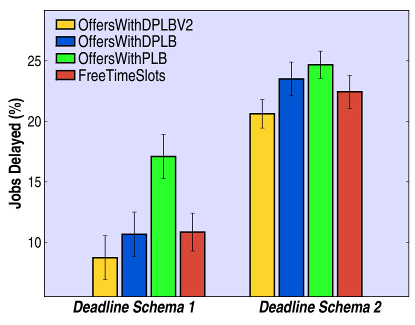
Job delays. Figure 4 illustrates total weighted delay for not meeting the deadlines of local, external, and global load. This metric is interesting because it shows the difference between what users asked and what the system was able to provide. We observe that the behavior of this metric is similar to the previous metrics, except that the delayed external jobs suffer much more in the *FreeTimeSlots* policy. In this policy, the providers disclose their free time slots to the metascheduler, which has no knowledge of the deadlines of the already accepted jobs. Therefore, the metascheduler makes blind decisions in terms of deadlines, which have a considerable impact on external jobs. Thus, even though resource providers disclose detailed information of their local



(a) Local Load.

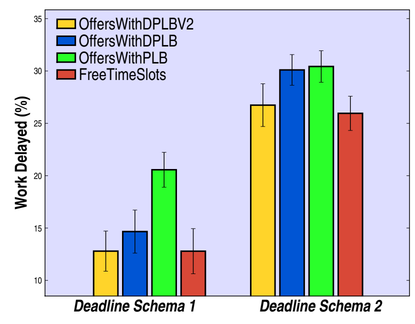


(b) External Load.

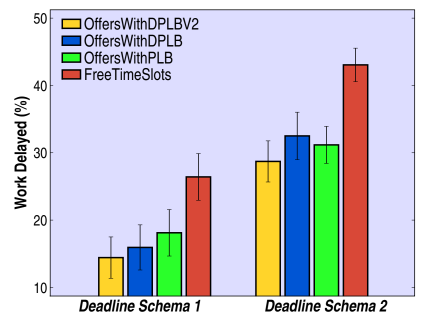


(c) Global Load.

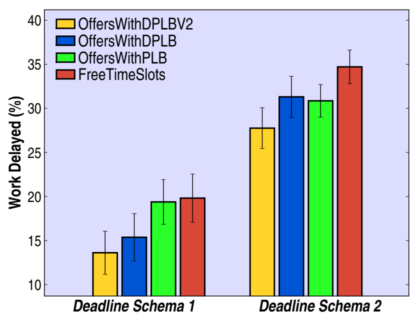
Figure 2. Number of jobs delayed for local, external, and global load.



(a) Local Load.

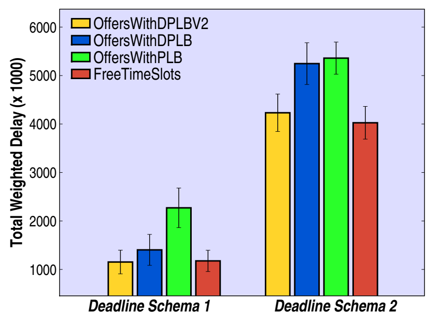


(b) External Load.

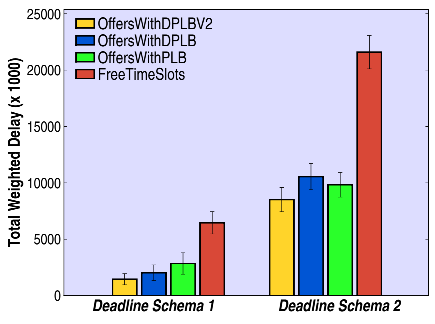


(c) Global Load.

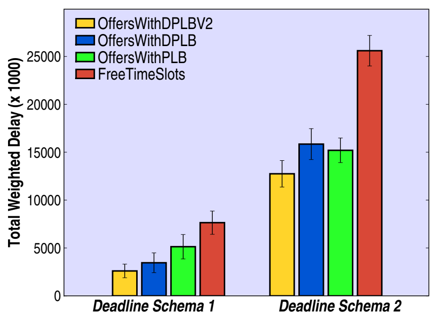
Figure 3. Amount of work delayed for local, external, and global load.



(a) Local Load.



(b) External Load.



(c) Global Load.

Figure 4. Total Weighted Delay for local, external, and global load.

load to the metascheduler using the *FreeTimeSlots* policy, such a policy produces much worse results than the simplest offer-based policy, i.e. *OffersWithPLB*. In this offer-based policy, the metascheduler uses only the offers, without knowing the resource providers' total computing power and load. This reveals that indeed, the offer sizes are a good indicator to balance load among resource providers without accessing their private information.

Load and System utilization. Another important factor is the *Number of Clusters* used by the BoT applications. Figure 5 shows the number of clusters used by BoT applications for each policy. We observe that the tighter the deadlines, the fewer options the metascheduler has to distribute the tasks of BoT applications among resource providers. That is because there is more unbalance in the load when jobs have tighter deadlines, and hence the metascheduler

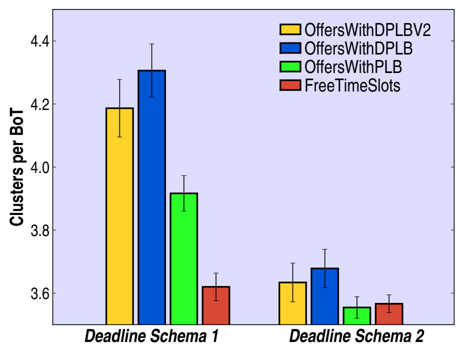


Figure 5. Number of clusters per BoT.

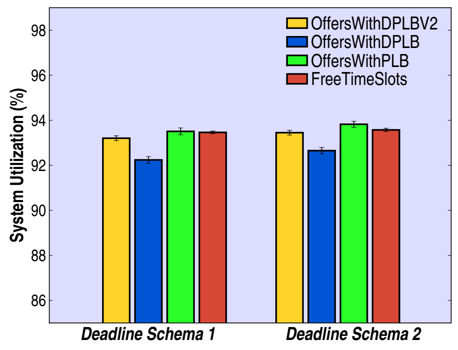


Figure 6. Global system utilization.

tends to use fewer options in offer-based policies, and fewer time slots in the *FreeTimeSlots* policy. In addition, the offer-based policies with double proportional load balancing tend to distribute the load better than the other two policies. Therefore, they allow more scheduling options for the next jobs arriving into the system. Regarding the *System Utilization* (Figure 6), we observe that the difference between the policies is minimal, i.e. less than 1 percent. *OffersWithDPLB* has a minimal decrease in relation to the other policies because it considers the total computing power of providers without their actual loads. Therefore, a provider that has less computing power than the others may receive fewer tasks even if the other big providers have higher load. This situation happens only when all offers that meet the deadline have the same size.

6. Related Work

Iosup et al. [10] have performed an extensive analysis of BoT scheduling in large-scale distributed systems. To evaluate the scheduling algorithms, they have considered different resource management architectures, task selection policies, and task scheduling policies. Among all the policies they have presented, three support a certain level of Quality-of-Service. These policies mainly work with priorities, without considering the expected completion time of the tasks. Casanova et al. [8] have also performed an extensive evaluation of policies for scheduling BoTs, but they have assumed more detailed information on the characteristics and load of all resources in the system is available. Different from these projects, our work aims at providing users with expected completion times that are based on user requirements, in particular user deadlines, and the restricted access of the scheduling queues.

Beaumont et al. [5] have investigated centralized and decentralized policies for scheduling BoTs considering both CPU and network. Different from our work, their goal is to maximize the throughput of user applications in a fair way. Viswanathan et al. [24] have proposed scheduling strategies for large compute intensive loads that are arbitrarily divisible and have deadline constraints. They use a pull-based scheduling strategy with an admission control to ensure the applications' deadlines are satisfied. Different from our approach, their work assumes a coordinator node that knows all the tasks in the system. Users do not receive any feedback when they submit their tasks, since the resource providers ask for tasks from the coordinator node according to their available capacity. In addition, users do not receive a new estimation of completion time when deadlines cannot be satisfied.

Benoit et al. [6] have presented scheduling techniques for multiple BoTs. One of the strategies presented in their work uses the Earliest Deadline First policy for the resources to choose the next task to be executed among those they have received. As in previous work, the deadlines have been used as a priority to select tasks, without giving any feedback and completion time estimations to the users.

Kim et al. [14] have proposed and compared eight heuristics that consider priorities and deadlines for independent tasks. Different from our work, tasks submitted to the system are not part of BoTs, therefore their deadlines are independent. Moreover, users submit the tasks directly to a centralized entity, i.e. resource providers have no local load. Abramson et al. [1, 2] proposed a broker to schedule *parameter sweep applications* (a subclass of BoT applications) with deadline constraints. Similar to the work developed by Kim et al., the deadlines of the tasks are independent, whereas in our case, the tasks in a BoT application share the same deadline, thus requiring a coordinated allocation strategy. Yeo and Buyya [25] have also investigated scheduling of tasks with deadline constraints, but focusing on single cluster environments and with no feedback when deadlines are not possible to be satisfied.

Much closer to our work, Islam et al. [12, 13] have investigated policies for admission control that consider jobs with deadline constraints and response time guarantees. Moreover, they provide a feedback for earliest feasible completion for non-admitted jobs. The main difference between their work and ours lies in the fact that they consider parallel jobs submitted to a single site, whereas we consider BoTs submitted to a multi-site environment.

Siddiqui et al. [21] have introduced a mechanism for capacity planning to optimize user QoS requirements in a Grid environment. Their mechanism supports negotiation and is based on advance reservations. A co-allocation request contains sub-requests that are submitted to the resource providers, which in turn send counter-offers when users and resource providers cannot establish an agreement. Although they use the concept of offers, which give a feedback to the users, these offers are for negotiations and therefore have a different role from the offers in our work. Moreover, our work specifically investigates the scheduling of BoTs.

Singh et al. [22] have developed a multi-objective Genetic Algorithm mechanism for provisioning resources. The resource providers publish their available slots, or offers, so that applications can use them according to their requirements (e.g. number of processors, time, and cost). The use of offers presented in our work is similar. However, in their work, resource providers generate offers for queues based on FIFO with conservative backfilling, and jobs do not have deadline constraints.

7. Conclusions

We introduced three policies for composing resource offers from multiple providers to schedule deadline-constrained BoT applications. These offers express the interest of resource providers in executing an entire BoT or only part of it without revealing their local load and total system capabilities. When the metascheduler receives enough offers to meet user deadlines, it can decide how to balance the tasks among the resource providers according to the information it has access, such as resource providers' total computing power and their local loads.

From our experiments, we observed that by using the free time slots of resource providers, BoT applications cannot access resources in short term even when local jobs could be rescheduled without violating their deadlines. The only benefit of publishing the free time slots to the metascheduler is that it can balance the load among resource providers, which makes more local jobs meet deadlines. However, when using offer-based policies, more BoTs can meet deadlines and the delays between the user deadline and the new deadline assigned by the system is much lower (in some cases 50% lower) in comparison to the policy that uses free time slots (*FreeTimeSlots*).

We also observed that the simplest offer-based policy (*OffersWithPLB*) produces schedules that delay fewer jobs in comparison to the *FreeTimeSlots* policy. However, *OffersWithPLB* rejects more local jobs than *FreeTimeSlots*. This happens because *OffersWithPLB* cannot balance the load among resource providers. If resource providers also publish the total computing power (the *OffersWithDPLB* policy), the metascheduler can balance the load and have similar acceptance rates as the *FreeTimeSlots* policy for local jobs. If the resource providers can make their load available (*OffersWithDPLBV2*), the metascheduler can reduce even more the number of jobs delayed; however the benefit is not significant. Therefore, our main conclusions are: (i) offer-based scheduling produces less delay for jobs that cannot meet deadlines in comparison to scheduling based on load availability (i.e. free time slots); thus it is possible to keep providers' load private when scheduling multi-site BoTs; and (ii) if providers publish their total computing power they can have more local jobs meeting deadlines.

Acknowledgments

We would like to thank Marcos Dias de Assunção, Chee Shin Yeo, and the anonymous reviewers for their comments on the paper. We also would like to thank Dr. Dror Feitelson for maintaining the Parallel Workload Archive, and all organizations and researchers who made their workload logs available. This work is partially supported by research grants from the Australian Research Council (ARC) and Australian Department of Innovation, Industry, Science and Research (DIISR).

References

[1] D. Abramson, R. Buyya, and J. Giddy. A computational economy for grid computing and its implementation in the Nimrod-G resource broker. *Future Generation Computer Systems.*, 18(8):1061–1074, 2002.

[2] D. Abramson, J. Giddy, and L. Kotler. High performance parametric modeling with Nimrod/G: Killer application for the global grid? In *Proceedings of the 14th International Parallel and Distributed Processing Symposium (IPDPS'00)*, Cancun, Mexico, May 1-5 2000. IEEE Computer Society.

[3] S. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403–410, 1990.

[4] A. Auyoung, L. Grit, J. Wiener, and J. Wilkes. Service contracts and aggregate utility functions. In *Proceedings of the 15th International Symposium on High Performance Distributed Computing (HPDC'06)*, Paris, France, June 19-23 2006. IEEE.

[5] O. Beaumont, L. Carter, J. Ferrante, A. Legrand, L. Marchal, and Y. Robert. Centralized versus distributed schedulers for bag-of-tasks applications. *IEEE Transactions on Parallel and Distributed Systems*, 19(5):698–709, 2008.

[6] A. Benoit, L. Marchal, J.-F. Pineau, Y. Robert, and F. Vivien. Offline and online master-worker scheduling of concurrent bags-of-tasks on heterogeneous platforms. In *Proceedings of the 22nd IEEE International Symposium on Parallel and*

Distributed Processing (IPDPS'00), Miami, USA, April 14-18 2008. IEEE Computer Society.

[7] T. D. Braun, H. J. Siegel, N. Beck, L. Bölöni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. A. Hensgen, and R. F. Freund. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. *Journal of Parallel and Distributed Computing*, 61(6):810–837, 2001.

[8] H. Casanova, A. Legrand, D. Zagorodnov, and F. Berman. Heuristics for scheduling parameter sweep applications in grid environments. In *Proceedings of the Heterogeneous Computing Workshop (HCW'00)*, pages 349–363, Cancun, Mexico, 1 May 2000. IEEE Computer Society.

[9] E. Deelman, G. Singh, M. Livny, J. B. Berriman, and J. Good. The cost of doing science on the cloud: the montage example. In *Proceedings of the ACM/IEEE Conference on High Performance Computing (SC'08)*, Austin, USA, November 15-21 2008. IEEE/ACM.

[10] A. Iosup, O. O. Sonmez, S. Anoep, and D. H. J. Epema. The performance of bags-of-tasks in large-scale distributed systems. In *Proceedings of the 17th International Symposium on High-Performance Distributed Computing (HPDC'08)*, pages 97–108, Boston, USA, 23-27 June 2008. ACM.

[11] M. Islam, P. Balaji, G. Sabin, and P. Sadayappan. Analyzing and minimizing the impact of opportunity cost in QoS-aware job scheduling. In *Proceedings of the International Conference on Parallel Processing (ICPP'07)*, page 42, Xi-An, China, Sep. 10–14 2007. IEEE Computer Society.

[12] M. Islam, P. Balaji, P. Sadayappan, and D. K. Panda. QoPS: A QoS Based Scheme for Parallel Job Scheduling. In *Proceedings of the 9th International Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP'03)*, volume 2862 of *Lecture Notes in Computer Science*, pages 252–268, Seattle, USA, June 24 2003. Springer.

[13] M. Islam, P. Balaji, P. Sadayappan, and D. K. Panda. Towards provision of quality of service guarantees in job scheduling. In *Proceedings of the IEEE International Conference on Cluster Computing (CLUSTER'04)*, pages 245–254, San Diego, USA, Sept. 20-23 2004. IEEE Computer Society.

[14] J.-K. Kim, S. Shivle, H. J. Siegel, A. A. Maciejewski, T. D. Braun, M. Schneider, S. Tideman, R. Chitta, R. B. Dilmaghani, and R. Joshi. Dynamically mapping tasks with priorities and multiple deadlines in a heterogeneous environment. *Journal of Parallel and Distributed Computing*, 67(2):154–169, 2007.

[15] Y. C. Lee and A. Y. Zomaya. Practical scheduling of bag-of-tasks applications on grids with dynamic resilience. *IEEE Transactions on Computers*, 56(6):815–825, 2007.

[16] M. Maheswaran, S. Ali, H. J. Siegel, D. A. Hensgen, and R. F. Freund. Dynamic mapping of a class of independent tasks onto heterogeneous computing systems. *Journal of Parallel and Distributed Computing*, 59(2):107–131, 1999.

[17] M. A. S. Netto and R. Buyya. Rescheduling co-allocation requests based on flexible advance reservations and processor remapping. In *Proceedings of 9th IEEE/ACM International Conference on Grid Computing (GRID'08)*, Tsukuba, Japan, 2008. IEEE Computer Society.

[18] V. S. Pande, I. Baker, J. Chapman, S. Elmer, S. M. Larson, Y. M. Rhee, M. R. Shirts, C. D. Snow, E. J. Sorin, and B. Zagrovic. Atomistic protein folding simulations on the submillisecond time scale using worldwide distributed computing. *Peter Kollman Memorial Issue, Biopolymers*, 68(1):91–109, 2003.

[19] F. I. Popovici and J. Wilkes. Profitable services in an uncertain world. In *Proceedings of the ACM/IEEE Conference on High Performance Networking and Computing (SC'05)*, Seattle, USA, November 12-18 2005. IEEE Computer Society.

[20] E. Shmueli and D. G. Feitelson. Backfilling with lookahead to optimize the packing of parallel jobs. *Journal of Parallel Distributed Computing*, 65(9):1090–1107, 2005.

[21] M. Siddiqui, A. Villazón, and T. Fahringer. Grid capacity planning with negotiation-based advance reservation for optimized QoS. In *Proceedings of the ACM/IEEE Conference on High Performance Networking and Computing (SC'06)*, Tampa, USA, Nov. 11–17 2006. ACM Press.

[22] G. Singh, C. Kesselman, and E. Deelman. A provisioning model and its comparison with best-effort for performance-cost optimization in grids. In *Proceedings of the 16th International Symposium on High-Performance Distributed Computing (HPDC'07)*, pages 117–126, Monterey, USA, 25-29 June 2007. ACM.

[23] S. Smallen, H. Casanova, and F. Berman. Applying scheduling and tuning to on-line parallel tomography. In *Proceedings of the ACM/IEEE Conference on Supercomputing (SC'01)*, Denver, USA, November 10-16 2001. ACM.

[24] S. Viswanathan, B. Veeravalli, and T. G. Robertazzi. Resource-aware distributed scheduling strategies for large-scale computational cluster/grid systems. *IEEE Transactions on Parallel and Distributed Systems*, 18(10):1450–1461, 2007.

[25] C. S. Yeo and R. Buyya. Integrated risk analysis for a commercial computing service. In *Proceedings of the 21th International Parallel and Distributed Processing Symposium (IPDPS'07)*, pages 1–10, Long Beach, USA, Mar. 26–30 2007. IEEE.

Biographies

Marco A. S. Netto is a PhD student in Computer Science at the University of Melbourne, Australia, and member of GRIDS laboratory. Marco has a Bachelor's (2002) and Master's degree (2004) in Computer Science, both from the Pontifical Catholic University of Rio Grande do Sul (PUCRS), Brazil. He has been working with resource management and job scheduling for high performance computing environments since 2000. Marco has also worked with structural Bioinformatics and Desktop Grids. Marco's current research effort is on co-allocation of space-shared resources for parallel applications. His work considers Quality-of-Service in terms of job's completion time guarantees, and rescheduling aspects, mainly due to inaccurate runtime estimates.

Rajkumar Buyya is an Associate Professor and Reader of Computer Science and Software Engineering; and Director of the Grid Computing and Distributed Systems (GRIDS) Laboratory at the University of Melbourne, Australia. He is also serving as the founding CEO of Manjrasoft Pty Ltd. He received his B.E and M.E in computer science and engineering from Mysore and Bangalore Universities in 1992 and 1995, respectively; and his Ph.D. in computer science and software engineering from Monash University, Melbourne, Australia, in April 2002. Dr. Buyya has authored/co-authored over 250 publications. He has co-authored three books: Microprocessor x86 Programming, BPB Press, New Delhi, 1995, Mastering C++, Tata McGraw Hill Press, New Delhi, 1997, and Design of PARAS Microkernel. The books on emerging topics that he edited include, High Performance Cluster Computing (Prentice Hall, USA, 1999), High Performance Mass Storage and Parallel I/O (IEEE and Wiley Press, USA, 2001), Content Delivery Networks (Springer, Germany, 2008), and Market Oriented Grid and Utility Computing (Wiley Press, USA, 2009). Dr. Buyya served as a speaker in the IEEE Computer Society Chapter Tutorials Program (from 1999-2001), Founding Co-Chair of the IEEE Task Force on Cluster Computing (TFCC) from 1999-2004, Interim Co-Chair of the IEEE Technical Committee on Scalable Computing (TCSC) from 2004-Sept 2005, and member of the Executive Committee of the IEEE Technical Committee on Parallel Processing (TCPP) from 2003-2009. He served as the first elected Chair of the IEEE Technical Committee on Scalable Computing (TCSC) during 2005-2007 and played a prominent role in the creation and execution of several innovative community programs that propelled TCSC into one of the most successful TCs within the IEEE Computer Society. In recognition of these dedicated services to computing community over a decade, President of the IEEE Computer Society presented Dr. Buyya a Distinguished Service Award in 2008.