Application-Specific Hints in Reconfigurable Grid Scheduling Algorithms

Bruno Volckaert[1], Pieter Thysebaert[2], Filip De Turck[3], Bart Dhoedt[1], and Piet Demeester[1]

[1] Department of Information Technology, Ghent University - IMEC
Sint-Pietersnieuwstraat 41, B-9000 Gent, Belgium
Tel.: +32 9 267 3587, Fax.: +32 9 267 35 99
{bruno.volckaert,pieter.thysebaert}@intec.ugent.be
[2] Research Assistant of the Fund of Scientific Research - Flanders (F.W.O.-V.)
[3] Postdoctoral Fellow of the Fund of Scientific Research - Flanders (F.W.O.-V.)

Abstract. In this paper, we investigate the use of application-specific hints when scheduling jobs on a Computational Grid, as these jobs can expose widely differing characteristics regarding CPU and I/O requirements. Specifically, we consider hints that specify the relative importance of network and computational resources w.r.t. their influence on the associated application's performance. Using our ns-2 based Grid Simulator (NSGrid), we compare schedules that were produced by taking application-specific hints into account to schedules produced by applying the same strategy for all jobs. The results show that better schedules can be obtained when using these scheduling hints intelligently.

1 Introduction

Computational Grids consist of a multitude of heterogeneous resources (such as Computational, Storage and Network resources) which can be co-allocated for the execution of applications or jobs. The allocation of resources to particular jobs and the order in which these jobs are processed on the Grid are determined by the Grid's management infrastructure through the application of a *scheduling algorithm*. Most jobs will need access to different resource types during their execution, meaning job execution progress depends on the quality of service delivered to that job by *every* resource involved. The exact sensitivity of a job's computational progress w.r.t. the individual resources' performance depends on the "nature" of that job: jobs that require huge amounts of CPU power, but perform (relatively) little I/O operations, will only suffer lightly from a temporary degradation of e.g. available network bandwidth, but cannot withstand a sudden loss of CPU power. Conversely, the computational progress made by an I/O-bound job is influenced dramatically by the network bandwidth available to that job, and to a lesser extent by the variation in available computing power. This leads us to the observation that:

1. algorithms that schedule jobs on a Computational Grid ought to take into account the status of multiple different resource types instead of solely relying on e.g. the available computational power.
2. using the same scheduling algorithm with rigid constraints for all job types can be outperformed by applying different scheduling algorithms for different job types; each job-specific algorithm only performs rigid resource reservation with *critical* resources, but allows for relaxed resource availability constraints when dealing with non-critical resources.

This indicates that programmable architectures, where the job scheduling mechanism is provided (at least partly) by the application (and where the algorithms could even be adapted on the fly), are a promising avenue towards grids offering a wide variety of services (each having their specific service metrics and quality classes). In this approach, the grid infrastructure is coarsely managed by cross-service components, supplemented by on-the-fly configurable service specific management components. The latter components manage the resources allocated to the service on a fine grained level, optimizing job throughput and service quality simultaneously according to service specific attributes and metrics. In this paper we show how *scheduling hints* can be incorporated into job descriptions. The goal of these hints is to enable a Grid scheduler to estimate the critical level of the different resource types w.r.t. that job. Because hints are contained in the job description, they are available at each scheduler in the Grid to which the job is submitted or forwarded.

This paper continues as follows: Sect. 2 starts with a short description of the related work. In Sect. 3, we give an overview of the relevant simulation models used: the Grid, Resource, VPN, Job and Scheduling Hint models are explained in detail. In Sect. 4, we discuss the various algorithms that we compared; they differ from each other in (i) the types of resources they take into account and (ii) whether or not they treat all jobs equally. Our simulated scenario and corresponding results are presented in Sect. 5, leading to the conclusions in Sect. 6.

2 Related Work

Well-known Grid Simulation toolkits include *GridSim* [1] and *SimGrid* [2]. The key difference with *NSGrid* [3] is that NSGrid makes use of a network simulator which allows for accurate simulation down to the network packet level (ns-2 [4]).

Scheduling jobs over multiple processing units has been studied extensively in literature. Machine scheduling [5][6] is concerned with producing optimal schedules for tasks on a set of tightly-coupled processors, and provides analytical results for certain objective functions. Jobs are commonly modelled as task graphs, or as continuously divisible work entities. As these models do not deal with "network connections" or "data transfers", they do not capture all the Grid-specific ingredients described in the previous section. Grid scheduling strategies which take both computational resource load and data locality into account are extensively discussed in [7]. The use of Application-specific scheduling hints is not considered however.

The *Metacomputing Adaptive Runtime System* (MARS) [8] is a framework for utilizing a heterogeneous WAN-connected metacomputer as a distributed computing platform. When scheduling tasks, the MARS system takes into account Computational Resource and Network load, and statistical performance data gathered from previous runs of the tasks. As such, the MARS approach differs from the scheduling model simulated by NSGrid, as NSGrid allows for *user preferences* to be taken into account as well.

Application-level scheduling agents, interoperable with existing resource management systems have been implemented in the *AppLeS* [9] work. Essentially, one separate scheduler needs to be constructed per application type. Our simulation environment allows the simulation of multiple scheduling scenarios, including those using a single centralized schedule as well as those having multiple competing schedulers (not necessarily one per application type).

3 Simulation Model

3.1 Grid Model

Grids are modelled as a collection of interconnected and geographically dispersed *Grid sites.* Each Grid Site can contain multiple *resources* of different kinds such as Computational Resources (CRs) and Storage Resources (SRs) interconnected by VPN links. At each Grid Site, resource properties and status information are collected in a local *Information Service.* Jobs are submitted through a *Grid Portal* and are scheduled on some collection of resources by a *Scheduler.* To this end, the scheduler makes *reservations* with the appropriate *Resource Managers.*

3.2 Grid Resource Models

Each Grid site can offer one or more CRs and/or SRs. A CR is a monolithic entity, described by its total processing power in MIPS, the maximum number of jobs that it can handle simultaneously and the maximum slice of processing power (in MIPS) that can be reserved for a single job. An SR on the other hand, serves the purpose of providing disk space to store input and output data. In our model, their basic properties include the total available storage space, the input data sets currently stored at the resource and the speed at which the resource can read and write data. While a SR does not perform computational work, it can be attached to the same network node as some CR. Interconnections between local resources are modelled as a collection of point-to-point VPN links, each offering a guaranteed total bandwidth available to Grid jobs. Of course, these VPN links can only be set up if, in the underlying network, a route (with sufficient bandwidth capacity) exists between the nodes to which these resources are attached. Different Grid Sites can be interconnected by a VPN link. These models are covered in more detail in [3].

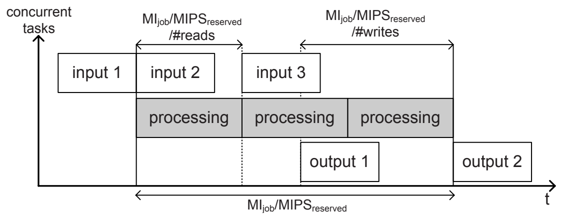


Fig. 1. Simulated job lifespan with indication of start-of-I/O events; non-blocking job

3.3 Job Model

The atomic (i.e. that which cannot be parallelized) unit of work used throughout this paper is coined with the term *job*. Each job is characterized by its length (measured in *instructions*), its required *input data sets*, its need for *storage*, and the *burstiness* with which these data streams are read or written. During a job's execution, a certain minimal computational progress is to be guaranteed at all times (i.e. a deadline relative to the starting time is to be met).

Knowing the job's total length (in million instructions, MI) and the frequency at which each input (output) stream is read (written), the total execution length of a job can be seen as a concatenation of instruction "blocks". The block of input data to be processed in such an instruction block is to be present before the start of the instruction block; that data is therefore transferred from the input source at the start of the previous instruction block. In a similar way, the output data produced by each instruction block is sent out at the beginning of the next instruction block. We assume these input and output transfers occur in parallel with the execution of an instruction block. Only when input data is not available at the beginning of an instruction block or previous output data has not been completely transferred yet, a job is suspended until the blocking operation completes. The presented model allows us to mimic both *streaming* data (high read or write frequency) and *data staging* approaches (read frequency set to 1). A typical job execution cycle (one input stream and one output stream) is shown in Fig. 1.

3.4 Scheduling Hints Model

From the job model described in the previous section, it is clear that the computational progress made by a job is determined by both the computational power and network bandwidth available to that job. As such, scheduling hints (distributed together with the job description) describe

– the resource types that are to be taken into account when scheduling this job; any subset of `{Computational / Network Resource}` can be specified.
– for each of the above resource types, the size of an acceptable (not preventing the job from being scheduled on that resource) deviation from the resource's performance delivered to that job (described in the job requirements).

It is not desirable to have critical resources deliver a less-than-minimal performance to the job, while this may not matter much for non-critical resources.

4 Algorithms

When jobs are submitted to a Grid Portal, a Scheduler needs to decide where to place the job for execution. As has been mentioned previously, we discriminate between algorithms using two criteria: the type of resources they take into account and whether or not they take into account scheduling hints. If the scheduler is unable to allocate the needed resources for a job, the job gets queued for rescheduling in the next scheduling round. The time between two scheduling rounds can be fixed, but it is also possible to set a threshold which triggers the next scheduling round. During each scheduling round, every algorithm processes submitted yet unscheduled jobs in a greedy fashion, attempting to minimize job completion time. Once scheduled, our scheduler does not pre-empt jobs.

4.1 Algorithm "NoNetwork"

As the name implies, this algorithm does not take into account the status of Network Resources when scheduling jobs. Rather, it assumes that only CRs are critical for each job. Furthermore, it will treat minimal job requirements as hard constraints; it disregards hints that might propose a softer approach. At first, "NoNetwork" will attempt to place a job on a site's local CRs, only using remote resources when strictly necessary (we believe this to be a plausible approach from an economic viewpoint). If this is impossible, and at least one remote CR is available, that job will be scheduled on the remote CR offering most processing power. It is therefore expected that this algorithm will perform badly when dealing with a significant amount of "I/O-bound" jobs: due to "blocking", these jobs will finish considerably later than predicted by the scheduling algorithm.

4.2 Algorithm "PreferLocal"

Similar to the "NoNetwork" algorithm, "PreferLocal" will a priori attempt to place a job on a site's local CRs. If this turns out to be impossible, remote CRs will be considered. While looking for the best resources for a particular job, however, "PreferLocal" not only considers the status of CRs, but also the residual bandwidth on network links connecting Computational and Storage Resources. The best resource combination is the one that maximizes the job's computational progress. For a job requiring one CR and one SR (in different Grid Sites, connected through a VPN link), the maximal computational progress (expressed in MIPS) that can be delivered to that job is given by

$$MIPS_{\mathit{eff}} = \min_{CR,VPN}(MIPS_{CR}, \frac{MI * BW_{VPN}}{8 * DATASIZE})$$

It is easily verified that it makes no sense to allocate a bigger portion of the best CR's power to this job, as due to network limitations, the job cannot be processed at a higher rate. In a similar way, due to CR limitations, it makes no sense to allocate more bandwidth to the job than $\frac{8*DATASIZE*MIPS_{eff}}{MI}$. Like "NoNetwork", "PreferLocal" does not adapt its strategy using job-specific hints.

4.3 Algorithm "Service"

Algorithm "Service", like "PreferLocal", considers both Computational and Network Resources when scheduling jobs. However, instead of immediately rejecting those resource combinations that would not allow some of the job's minimal requirements to be met, it can still select those resources (if none better are found) if this is declared "acceptable" by the appropriate job hint. For instance, jobs can specify that their available network bandwidth requirements are less important than their computational requirements (and/or quantify the relative importance), or that there is no gain in finishing the job before its deadline (i.e. no gain in attempting to maximize that job's $MIPS_{eff}$). Using these hints, jobs can be divided into different classes, where all jobs in one class have similar associated hints. The algorithm can then be seen as delivering the same service to each of those jobs in a single class.

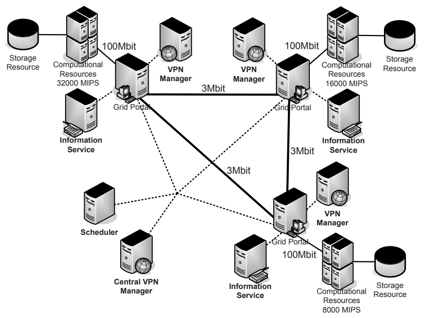
5 Simulation Results

5.1 Simulated Grid

A fixed Grid topology was used for all simulations presented here. This topology is depicted in Table 1. Grid control components are interconnected by means of dedicated network links providing for out-of-band Grid control traffic (as shown by the dotted network links).

Table 1. Sketch of simulated scenario

Service type	I/O size	MI
I/O-bound	6100 MB	12500000
CPU-bound	0.4 MB	25000000



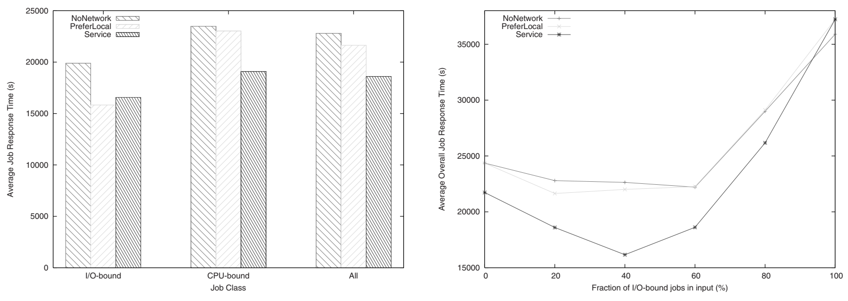
5.2 Simulated Jobs

In our simulations, two types of hints were used (i.e. two service types). The first type of hint is supplied with CPU-bound jobs, and specifies that the job should be scheduled on the fastest CR, even if this means scheduling the job on a remote resource when it could have been scheduled locally. The second type of hint is distributed with I/O-bound jobs, stating that these jobs are better off being scheduled using only local resources, as this offers better chances of allocating sufficient network bandwidth. In both cases, however, resource loads are not ignored; rather, the *preferred* execution rate for a job is no longer treated as a rigid minimum.

We have compared the "Service" algorithm (which understands and uses the hints as specified) for this job set with the "NoNetwork" and "PreferLocal" algorithms, both disregarding the hints.

5.3 Per-Class Response Times

The average job response time for each algorithm is shown in the left part of Fig. 2. The figure shows both the overall response rate and the per-class response rates when 20% I/O jobs and 80% CPU intensive jobs are submitted to the Grid. As expected, "NoNetwork" fails to produce good schedules for I/O-bound jobs, as it ignores network loads (which are, of course, of particular importance to this type of job). In addition, notice that the use of hints (algorithm "Service") improves the average response time for CPU-bound jobs (which make up most of the jobs in this simulation). Indeed, some jobs are now processed at a rate (slightly) lower than the preferred one (the goal of the hints is exactly to specify that this is allowed), but finish sooner than if they were delayed in time.



(a) average job response time (b) average job response time as function of input job set

Fig. 2. NoNetwork, PreferLocal and Service schedule performance

5.4 Response Times with Varying Class Representation

In these simulations we stressed the three Grid sites by submitting a heavy job load (parameterized by the percentage of I/O-bound jobs in the total job load). The resulting average job response time for the three algorithms (as a function of the percentage of I/O-bound jobs) is shown in Fig. 2 (right side).

When only CPU-bound jobs are submitted, “NoNetwork” performs like “PreferLocal”, as we remarked previously. Note that “Service” performs slightly better, as this algorithm does not prefer local resources over remote. When the amount of I/O-bound jobs is increased, “NoNetwork” performance degrades as network load status gains importance when scheduling jobs. Since “PreferLocal” always tries local resources first (instead of the best resources), it will schedule more I/O-bound jobs remotely (i.e. using lower-bandwidth links) as CPU-bound jobs (the majority) use up these local resources; this accounts for the difference with “Service”. Once the fraction of I/O-bound jobs passes 60%, the network links of our simulated Grid saturate, and the three algorithms’ performance converges.

6 Conclusions

In this paper we have shown the benefits of using active scheduling mechanisms in a service oriented Grid environment. In particular, we used NSGrid to compare the efficiency of schedules produced by algorithms that do not take into account the service specific needs of a job, to schedules produced by algorithms that use service scheduling hints. It was shown that when using the latter algorithms, average job response times in our simulated scenario improved significantly (up to 30% in some cases).

References

1. Buyya, R., Murshed, M.: GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing, The Journal of Concurrency and Computation: Practice and Experience (CCPE), Wiley Press, (2002)
2. Legrand, A., Marchal, L., Casanova, H.: Scheduling Distributed Applications: the SimGrid Simulation Framework, Proc. of CCGrid 2003 (2003) 138–145
3. Volckaert, B., Thysebaert, P., De Turck, F., Dhoedt, B., Demeester, P.: Evaluation of Grid Scheduling Strategies through a Network-aware Grid Simulator, Proc. of PDPTA 2003 (2003) 30–35
4. “*The Network Simulator - NS2*”, website, `http://www.isi.edu/nsnam/ns`
5. Feitelson, D. G., Rudolph, L., Schwiegelshohn, U., Sevcik, K. C., Wong, P.: Theory and Practice in Parallel Job Scheduling, Job Scheduling Strategies for Parallel Processing, (1997) 1–34
6. Hall, L. A., Schulz, A. S., Shmoys, D. B., Wein, J.: Scheduling to Minimize Average Completion Time: Off-line and On-line Approximation Algorithms, Mathematics of Operations Research, **22-3** (1997) 513–544

7. Ranganathan, K., Foster, I.: Simulation Studies of Computation and Data Scheduling Algorithms for Data Grids, Journal of Grid Computing, Kluwer Academic Publishers, **1-1** (2003) 53–62
8. Gehring, J., Reinfeld, A.: Mars - a framework for minimizing the job execution time in a metacomputing environment, Proc. of Future General Computer Systems '96 (1996)
9. Berman, F., Wolski, R., Figueira, S., Schopf, J., Shao, G.: Application-Level Scheduling on Distributed Heterogeneous Networks, Proc. of SuperComputing 96 (1996)