Network Aware Scheduling in Grids

B. Volckaert, P. Thysebaert, M. De Leenheer
F. De Turck, B. Dhoedt, P. Demeester

Abstract

Computational Grids consist of an aggregation of data and computing resources, which can be co-allocated to execute large data-intensive tasks which cannot be handled by an individual resource. The vast amounts of (possibly remote) data these tasks process give rise to the need for a high-capacity network interconnecting the various resources. In this paper, it is argued that, in order to obtain quality job schedules resulting in low throughput times while making efficient use of available resources, network- and service-aware algorithms need to be used. We present several heuristic algorithms, which we simulate extensively in order to gain insight in the quality of the generated schedules. In particular, in a Grid with heterogeneous Computational Resources and link capacities, we show how the average job response time can be improved by distinguishing between data-intensive and compute-intensive jobs, and scheduling these jobs based upon both Computational Resource and network load. We show that the heuristics presented perform well when compared to an Integer Linear Programming approach modelling a periodic on-line scheduler.

1 Introduction

In parallel and distributed computing communities, a lot of attention has recently been paid to Computational Grids. These Grids are conceived as a collection of resources of various types (computational, storage, etc.), distributed over multiple geographical locations. The underlying principle of an operational Grid is that several resources can be co-allocated to the same resource-intensive task, overcoming each resource's individual limitations. These resource-intensive tasks will typically operate on large amounts of data, which must be transferred from its storage site to the processing site. This implies that, when scheduling jobs on this Grid, gathering information on data location and network bandwidth availability is essential in order to generate quality schedules.

Scheduling algorithms for parallel computing systems have been studied extensively in literature [1]. Most approaches assume a discrete workload (e.g. jobs) and use a stochastic [2, 3] or linear programming model [4]. Divisible load theory [5], on the other hand, assumes a continuously divisible load, and seems well suited to model a Grid, as it leads to fairly easy systems of equations, even for large problem sizes. However, in order to remain analytically tractable, these techniques trade in some of the characteristic properties of a Grid by simplifying network and/or job models or focusing on a particular topology.

More accurate grid models require the use of simulation tools in order to be evaluated. In [6] and [7], network-awareness is introduced in the scheduling decisions by replicating data to suitable sites; jobs can then be scheduled to run at a site holding all the necessary data. Our in-house developed NS2-based Grid simulator NSGrid [8, 9] extends this model by allowing data to be transferred to the execution site while the job has already been started, and this extension is used in the simulations described here.

In this paper, we use this simulator to investigate the generation of schedules for a Grid in which the core network uses high capacity (possibly optical) core links. While this approach seemingly provides for abundant bandwidth in the network, it follows from the expected Grid job size in the near future (the expected data size generated yearly in the European DataGrid project is in the order of 12-14 PB/year [10]) that high-bandwidth (typically found in optical networks) interconnections are a necessity, not a luxury. This implies that naive scheduling heuristics, which assume "infinite" (or at least "sufficient") network bandwidth at all times or ignore the network altogether are unable to generate quality schedules, both in terms of average job response time and in terms of efficient resource utilization.
We verify this through simulation of different scheduling heuristics and calculation of an ILP solution to the problem, operating on a suitable workload (consisting of a mix of different types of Grid jobs requiring sufficiently high computational power and network bandwidth).
The remaining sections of this paper are organized as follows: Section 2 explains how the various Grid components in our simulation environment are modelled. A concise description of the scheduling problem addressed here, the ILP model of a periodic, on-line scheduler and the heuristics used in our simulations are presented in section 3. Our results are presented in section 4, and the main conclusions drawn from them are summarized in section 5.

2 Grid Model

For the simulations described in this paper, a Grid has been modelled as a set of interconnected, geographically dispersed *Grid sites*. At each site, one or more resources are present. The status and properties of these resources are stored in an *Information Service* repository. The idea is that a single site groups "geographically close" resources, connecting them through a high-bandwidth, low-latency network. Each site is connected to the backbone network and all simulations use static shortest-path routing for inter-site traffic.

2.1 Grid Resources

The resources modelled in the simulation framework include Computational Resources (CR) and Storage/Information Resources (SR/IR). Computational Resources represent either a cluster or multi-computer, and are described by the number of (time-shared) processors provided, the amount of memory and temporary disk space. In order to schedule a job, a reservation can be made with a CR, consisting of a time-share of a processor dedicated to that job. Storage/Information Resources hold the data written/read by the job. We use the notation $\mathcal{C}$ for the set of Computational Resources, with elements $c \in \mathcal{C}$.
Jobs transfer data from and to remote resources using *connections*, which act as point-to-point bandwidth pipes between a pair of resources and are allocated with the network management system by the grid scheduler. This allows the network to be viewed as a resource (with limited renewable capacity a.k.a. bandwidth, of which a portion can be reserved for a connection), giving the scheduler the possibility to generate accurate network-aware schedules, and balance computation vs. communication bottlenecks. We use the notation $\mathcal{E}$ for the collection of network links, where each $e \in \mathcal{E}$ has a bandwidth B_e.

2.2 Jobs

We reserve the term *job* for atomic units of work. Jobs $j \in \mathcal{J}$ are characterized by their computational and communication (I/O) complexity. The first property is captured using the *reference running time* t_j (time to execute on a reference processor when not experiencing any communication bottlenecks). The communication complexity is specified by a number

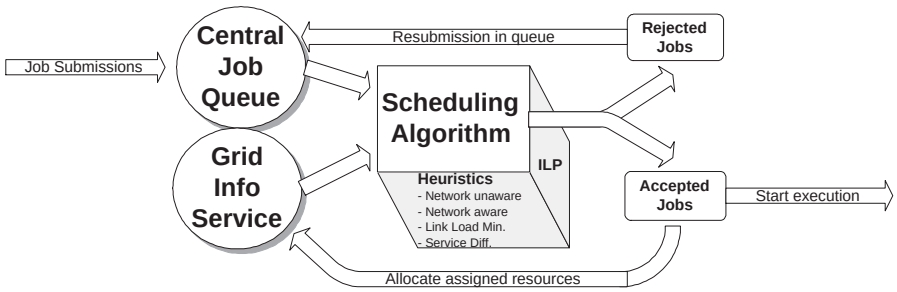


Figure 1: General overview of scheduling algorithm

of parameters: the data sets the job needs as input (total size d_j^r), the size of the outputs it produces (total size d_j^w), whether or not I/O are processed block-per-block (the opposite case is when all input data is pre-staged to the CR the job is running on *prior* to execution) and the amount of separate blocks I/O data is divided into (if applicable). This allows to simulate data *streams* with arbitrary precision. The fundamental relation between computation and communication in our model is that a data block can only be processed when it has arrived entirely; if this is not the case, the job's computational progress is *blocked*.

3 Scheduling Algorithms

The scheduling algorithms we use are *queueing* algorithms (see figure 1) that attempt to schedule jobs in a FCFS fashion [11], and, once a job is scheduled, the scheduler does not attempt to pre-empt jobs. Since this introduces the risk of allocating minimal leftover computational resource time shares to certain jobs, we have demanded that a processor can be time-shared by at most a specified number of concurrent jobs N_c. Our scheduling algorithms are executed periodically; after a specified amount of time T, all job requests are taken into consideration, and, based on the state of the grid's resources at that time, a schedule is calculated.

3.1 ILP

In this section we present an Integer Linear Programming (ILP) model, which captures all important parameters in the scheduling problem. It is well known that ILP is a computationally intensive technique. This, together with the fact that the proposed grid and job models are both very generic, obliges us to impose additional restrictions to the model. We model a grid as a collection $\mathcal{C}$ of resources, where each $c \in \mathcal{C}$ has a processing capacity P_c and unlimited storage capacity. We assume a job can only be executed at its reference processing speed, and thus the execution length is given by the reference running time t_j. This also implies the job is not held up by communication bottlenecks. We further assume a job has only one input and one output block, which is to be transferred at a constant datarate during the job's execution. It follows that a job requires $b_j^r = \frac{d_j^r}{t_j}$ of input bandwidth, and $b_j^w = \frac{d_j^w}{t_j}$ of output bandwidth. Two additional parameters for the ILP model are the budget w_j which a job pays if it is accepted for execution, and the arrival site s_j. We introduce the following binary variables:

- $x_j = 1 \iff$ job j is accepted for execution.
- $y_{jc}^p = 1 \iff$ job j uses processing power of resource c.
  $y_{jc}^s = 1 \iff$ job j uses storage capacity of resource c.

- $z^r_{je} = 1 \iff$ job j uses network link e for input.
  $z^w_{je} = 1 \iff$ job j uses network link e for output.

These variables allow us to express the cost to execute a job, which is given by a weighted sum of the processing cost and the transfer cost:

$$C_j = \alpha \times k_j \times \sum_{c \in \mathcal{C}} (C^p_c \times p_j \times y^p_{jc}) + \beta \times k_j \times \sum_{e \in \mathcal{E}} (C^b_e \times (b^r_j \times z^r_{je} + b^w_j \times z^w_{je})) \quad (1)$$

where $k_j = \lceil \frac{t_j}{T} \rceil$ is the number of periods a task will run, and C^p_c and C^b_e are the costs for using resource c and network link e during one time unit. Parameters α and β allow us to give priority to CPU-bound or network-bound jobs. We investigate their influence on the scheduler in section 4.3.2.
Our objective function, stated below, expresses the *profit* our scheduler makes by choosing which tasks to execute where. Clearly our goal is to maximize this function, as it strikes a balance between a user's perceived satisfaction (high acceptance rate) and the provider's optimal resource utilization (keeping the costs low means jobs are executed close to where they are submitted).

$$f = \sum_{j \in \mathcal{J}} (w_j \times x_j - C_j) \quad (2)$$

We now introduce the constraints which need to be satisfied. Recall that all variables are binary-valued. We first state that an accepted job uses computational power of exactly one resource, uses its submission site as storage site, and possibly uses multiple network links:

$$\forall j \cdot \sum_{c \in \mathcal{C}} y^p_{jc} = x_j \quad ; \quad \forall j \cdot y^s_{jc} = \begin{cases} x_j & \text{if } c = s_j \\ 0 & \text{otherwise} \end{cases} \quad ; \quad \forall j \cdot \forall e \cdot z^{r,w}_{je} \leq x_j \quad (3)$$

The following constraints limit the usage of the individual resources:

$$\forall c \cdot \sum_{j \in \mathcal{J}} y^p_{jc} \leq N_c \quad ; \quad \forall c \cdot \sum_{j \in \mathcal{J}} (p_j \times y^p_{jc}) \leq P_c \quad ; \quad \forall e \cdot \sum_{j \in \mathcal{J}} (b^r_j \times z^r_{je} + b^w_j \times z^w_{je}) \leq B_e \quad (4)$$

The model is completed by expressing the balance of flows in the network (analogous constraints are necessary for variables z^w_{je}):

$$\forall j \cdot \forall u \cdot \sum_{\substack{v \in \mathcal{V} \\ (u,v) \in \mathcal{E}}} z^r_{j(u,v)} - \sum_{\substack{v \in \mathcal{V} \\ (u,v) \in \mathcal{E}}} z^r_{j(v,u)} = y^s_{ju} - y^p_{ju} \quad (5)$$

3.2 Heuristics

The ILP model described above can only be used to solve an offline scheduling problem because of its computational intensiveness (i.e. it is unsuitable as an actual scheduling algorithm implementation). Below we give an overview of the scheduling heuristics that were used in our study and compare their results with the ILP solution.

3.2.1 Network Unaware

We have used network unaware scheduling as a naive heuristic for comparison. This heuristic will compute job schedules based on CR/IR/SR status. It does not take into account information concerning the status of resource interconnections. Because of this, job processing

can block on I/O operations: computational progress is no longer determined by the CR's processor fraction that has been allocated to a job (which, together with the job's length and the CR's relative speed determines its earliest end time *if all input and output transfers complete on time i.e. before the start of the appropriate instruction block*), but rather by the limited bandwidth available to its I/O streams.

3.2.2 Network Aware

Network aware algorithms will not only query the Information Services (for resources adhering to the job's requirements), but also the network management for information about the status of the interconnecting network links. Based on both the Information Services' and Connection Manager's answer, the heuristic is able to calculate job execution speed and end time more accurately, taking into account the speed at which data can be delivered to (or sent from) each available CR. For jobs with one input and one output stream, the best resource (CR/IR/SR) triplet is the one that minimizes the expected completion time of the job. This value is determined by the available processing power to that job on the Computational Resource, the job's length, the job's total I/O size and the residual bandwidth on the links from IR to CR and from CR to SR.

3.2.3 Minimum Hop Count

This heuristic attempts to minimize network usage when scheduling jobs. In order to achieve this, the scheduler evaluates the network load each CR/IR/SR triplet selection would generate for a given job, and chooses to schedule the job on the triplet that minimizes this network usage. Typically, when jobs require that their I/O be streamed from/to the originating site, this heuristic will first attempt to schedule the job on local resources, and, when this is infeasible, will try to submit the job to a CR *as close as possible to the job's originating site*, minimizing the amount of network links that data has to be sent over.

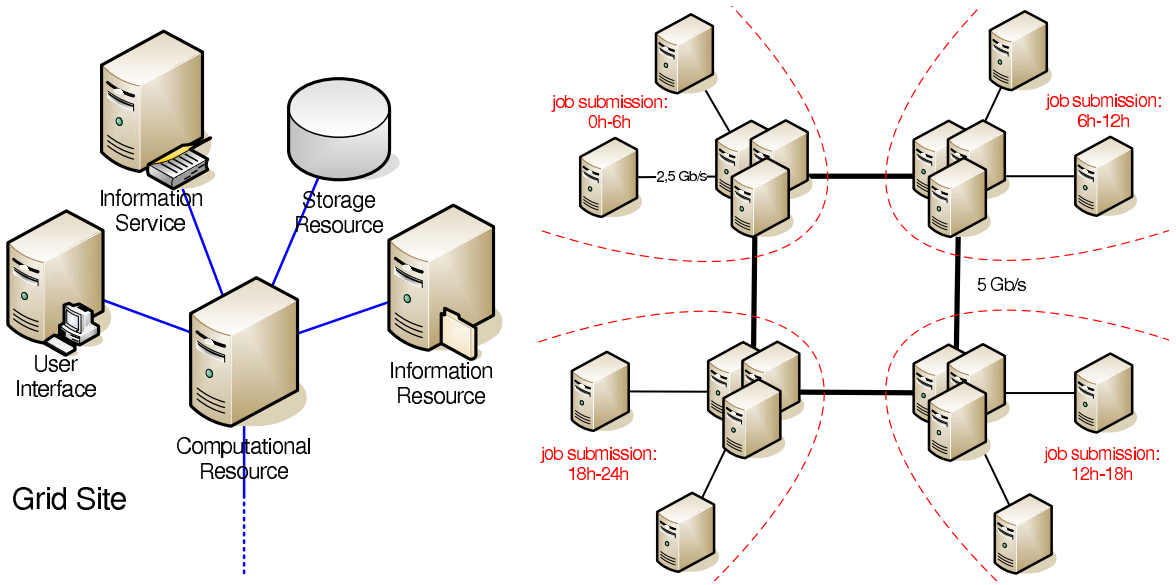
3.2.4 Service Differentiation

The Service Differentiation heuristic will compute the data intensity of a job, based on the job's I/O and processing requirements, and, based on this metric, will classify the job as either a data intensive or a computational intensive job. Data intensive jobs will only be scheduled on resources local to the job's originating site (if these resources adhere to the job's requirements). If no local processing slots are available, the data intensive jobs are queued for local processing. Jobs in the other service class will be scheduled on remote processing resources (in a network aware manner), in order to maximize the chance of having local processing slots available for data intensive jobs.

4 Simulation results

4.1 Simulated Topology

The Grid topology used in our simulations is shown in figure 2. The topology is symmetric and consists of 12 Grid sites (8 edge and 4 core sites) interconnected by means of bidirectional optical links ($2, 5Gb/s$ links between edge and core sites, and a $5Gb/s$ ring network between core sites). Furthermore, each core site connects two edge sites to the core ring network. Each site contains an IR/SR/CR and an Information Service management component responsible for tracking resource properties. Jobs are submitted to the site through that site's user



(a) Grid Site (b) Grid Topology

Figure 2: Simulated Topology

interface. Core sites differ from edge sites in terms of the offered Computational Resource; both CR types have 4 processors, capable of running 8 jobs simultaneously, but a core CR offers three times the processing speed of its edge counterpart. A site's local interconnections are modelled as fiber channel links capable of transmitting data at $10Gb/s$.

4.2 Grid Jobs

The user interface at each Grid site submits two different job types: data-intensive and cpu-intensive jobs. Both job types have the same reference run time, but they differ in the amount of I/O data they need/generate. Jobs are instantiated at specified time intervals, according to the interarrival time (IAT) distribution of the job's class. Furthermore, all jobs *stream* their I/O from/to the originating site's local IR/SR. Average job parameters have been summarized in table 1. In each simulation, the job load consists of 405 jobs. We chose to use a fixed interval of $1000s$ between consecutive scheduling rounds.

	CPU-Job	**Data-Job**
Input(GB)	15.6	156
Output(GB)	15.6	156
IAT(s)	1350	5400
Ref. run time(s)	10000	10000

Table 1: Job Properties

4.3 Results

4.3.1 Bandwidth of Core Network

The job throughput for different core network bandwidth values is shown in the left-hand side of figure 3. We clearly see that, for the given job load, a "sufficient" bandwidth exists; going above this value has little to no effect on the job throughput and thus implies an overdimensioned core network.

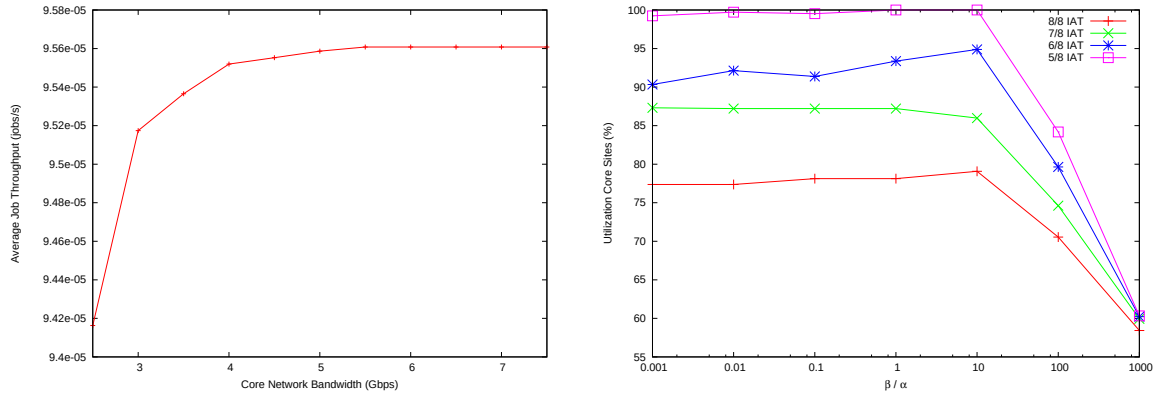


Figure 3: Evaluation of ILP model

4.3.2 Parameters of ILP

The right-hand side of figure 3 depicts the utilization of the core sites for various values of the quotient $\frac{\beta}{\alpha}$. This quotient expresses the relative cost between the usage of a network link and a computational resource. Clearly, higher values imply that using the network becomes more and more expensive, and thus local execution becomes preferable. We performed this experiment for different values of the job interarrival times (IAT); higher job arrival rates generate higher site utilization as long as the network is cheap enough. When the usage of the network becomes too expensive, the site utilization drops notably.

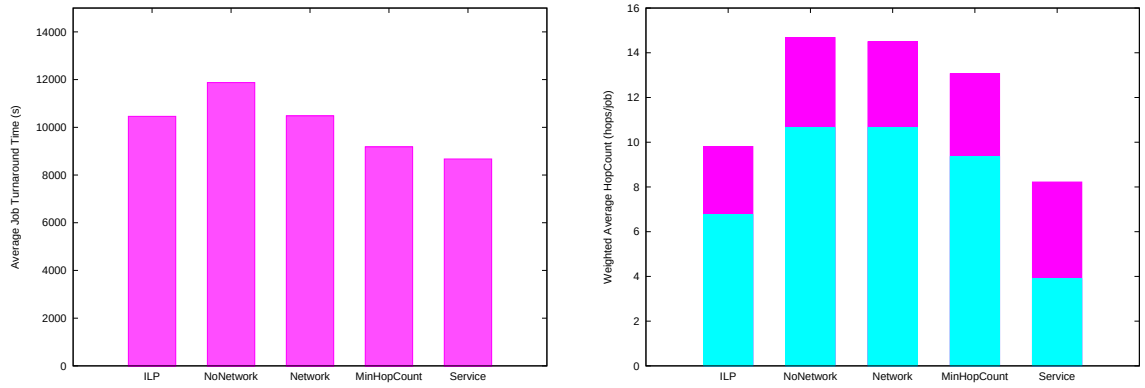


Figure 4: Schedule Quality Comparison

4.3.3 Turnaround Time

The left-hand side of figure 4 shows the average job turnaround time (time between arrival of a job and the time it finishes) for the different scheduling heuristics discussed above. The "Service" heuristic clearly provides the best performance, by pushing cpu-intensive jobs to remote processing sites, and, in doing so, reserving the local processing slots for data-intensive jobs. It is exactly because remote processing a lot of data-intensive jobs causes a network bottleneck (and thereby causes processing to be blocked), that the network aware "Minimum Hop Count" and "Service" heuristics perform well.

4.3.4 Network Utilization

The difference in network utilization resulting from deploying different scheduling heuristics is shown in the right-hand side of figure 4 (the lower part of each bar represents network utilization of data-intensive jobs). The metric used here is called weighted hopcount, as we took into account the fact that data-intensive jobs use on average 10 times more data than other jobs. The results show that the least network traffic flows when we make use of the service differentiation heuristic, maximizing the chance that data-intensive jobs obtain a local processing slot.

5 Conclusion

Our main observations are that due to the expected size of the data transferred by a Grid job in the future, even with high-capacity network links, naive (non-network aware) heuristics are outperformed by network-aware scheduling strategies both from the end user's viewpoint (job turnaround time) and from the provider's viewpoint (network usage efficiency). Furthermore, if multiple applications with different bandwidth requirements are run on the Grid, further improvements are possible by deploying different service-oriented scheduling strategies.

References

[1] D. G. Feitelson, L. Rudolph, U. Schwiegelshohn, K. C. Sevcik, P .Wong, "*Theory and Practice in Parallel Job Scheduling*", Proc. of JSSPP3, pp. 1-34

[2] A.I.D. Bucur, D.H.J. Epema, "*An Evaluation of Processor Co-Allocation for Different System Configurations and Job Struxctures*", Proc. of SBAC-PAD 2002, pp. 195-203.

[3] A.I.D. Bucur, D.H.J. Epema, "*The Influence of the Structure and Sizes of Jobs on the Performance of Co-Allocation*", Proc. of JSSPP6, pp. 154-173.

[4] L. A. Hall, A. S. Schulz, D. B. Shmoys, J. Wein, "*Scheduling to Minimize Average Completion Time: Off-line and On-line Approximation Algorithms*", Mathematics of Operations Research, Vol. 22, Issue 3 (Aug 1997), pp. 513-544

[5] L. Marchal, Y. Yang, H. Casanova, Y. Robert, "*A realistic network/application model for scheduling divisible loads on large-scale platforms*", Ecole Normale Superieure de Lyon, Research Report 2004-21, April 2004

[6] K. Ranganathan, I. Foster, "*Simulation Studies of Computation and Data Scheduling Algorithms for Data Grids*", Journal of Grid Computing, Vol. 1, Issue 1, pp. 53-62

[7] D.G. Cameron, R. Carvajal-Schiaffino, A.P. Millar, C. Nicholson, K. Stockinger, F. Zini, "*Evaluating Scheduling and Replica Optimisation strategies in OptorSim*", Grid2003

[8] "*The Network Simulator - NS2*", website, `http://www.isi.edu/nsnam/ns`

[9] B. Volckaert, P. Thysebaert, F. De Turck, P. Demeester, B. Dhoedt, "*Evaluation of Grid Scheduling Strategies through a Network-aware Grid Simulator*", PDPTA 2003, pp. 31-35

[10] "*The DataGrid Project*", website, `http://eu-datagrid.web.cern.ch/eu-datagrid`

[11] M. Hovestadt, O. Kao, A. Keller, A. Streit, "*Scheduling in HPC Resource Management Systems: Queueing vs. Planning*", Proc. of JSSPP9

Contact Details

Bruno.Volckaert@intec.UGent.be

Department of Information Technology (INTEC - IMEC)

Ghent University

St.-Pietersnieuwstraat 41, B-9000 Gent, Belgium

Tel. ++32 9 264 9965