Job-Scheduling Strategies for a Computational Grid

Collaboration project with: Andrei Tchernykh CICESE Research Center, Mexico and Ramin Yahyapour
(Computer Engineering Institute, University of Dortmund, Germany)

Get in contact with Andrei Tchernykh
http://usuario.cicese.mx/~chernykh/
Phone +52 (646) 175-0595 ext. 25434
FAX +52 (646) 175-0593
E-mail: chernykh@cicese.mx

1. Project Summary:

In recent years parallel computers and clusters have been deployed to support computation-intensive applications and become part of so called computational grids (C-GRIDs) or metacomputers. Such C-GRIDs are emerging as a new paradigm for solving large-scale problems in science, engineering, and commerce. Grids are becoming almost commonplace, with many projects using them for production runs. The initial challenges of Grid computing—how to run a job, how to transfer large files, how to manage multiple user accounts on different systems—have been resolved to first order, so researchers can now address the issues that will allow more efficient use of the resources. The use of Grid resource-management tools is far from ubiquitous because of the many open issues of the field, including the multiple layers of scheduling, dynamicity and scalability.

Resources are decentralized and geographically dispersed. However, it is unrealistic to build completely decentralized or centralized systems. Multilayer resource management is a natural model of such a system, so that a tradeoff between fully centralized and fully decentralized model can be found. Thus, we should extend existing scheduling strategies to these multilayer models of resource management, and resource selection problem should be addressed to cope with the local load fluctuations.

Many of the difficulties in multi-layer systems already occur in two-level systems, and so in this project we focus on two-level solutions. First level is responsible for selecting of a job from the set of all submitted jobs, and then selecting for each job of best suitable resource from all available resources taking into consideration job parameters, computer parameters, additional requirements of the C-GRID scheduling infrastructure, and optimization criteria. At the second level, local schedulers are used at each resource. The efficiency of scheduling policies is crucial to C-GRID performance. Therefore, the development of a theory of multilayer hierarchy scheduling, and the design of corresponding resource optimization C-GRID infrastructure for real world applications are becoming highly desirable.

We need also theoretical and experimental analysis to understand questions of scalability. Currently, many existing grids are relatively small. We expect future resources and workloads to be orders-of-magnitude larger. For large grids, these scalability issues come into play.

Intellectual Merit. We propose to develop solutions for GRID resource management and to investigate their efficiency and scalability. Our models will be based on scheduling mechanisms currently in use in production systems. We will analyze these mechanisms, determining their performance guarantees, scalability, and efficiency. Our strategies will be integrated with different C-GRID models and applied over a heterogeneous, cluster-based meta-computer.

Broader Impact. The purpose is to study the processes of resource management that involves many players and possibly several different layers of schedulers. At the highest layer, Grid schedulers may have a more general view of the resources but are very far away from the resources where the application will eventually run on. The lowest layer is a local resource management system that manages a specific resource or set of resources. Other layers may be in between these. Since introducing GRID computing in the early 90s and multilayer-hierarchy resource management in the last few years, a broad variety of selection and local scheduling strategies have been suggested and investigated with different assumptions such as a GRID model (centralized or decentralized), hierarchy, allocation (co-allocation), job models (computational o data intensive), etc. Many different classes of these strategies have been defined in the last decade. A main drawback of these approaches is that most of them are based on experimental analysis and theoretical analysis is weakly exploited. Our research will provide a theoretical foundation and performance evaluation that will include the following methodology aspects: (1) theoretical analysis of scheduling algorithms under different models and constrains; (2) experimental studies of proposed strategies
based on realistic GRID parameters and real workloads; (3) comparison of the behavior of such algorithms on a variety of existed implementations of C-GRIDs; (4) considering all steps of the development of Computational GRID from the first formulation of the requirements to its implementation in the computer network.
2. Hierarchical (Multilevel) Scheduling Strategies

This section describes the approach that we will take to address the development of multilayer hierarchy GRID resource management and distributed scheduling techniques for the specified complex problem space. First we describe the basic research, which will involve developing a model for resource management and scheduling, GRID architecture for the resource management, selection and local scheduling strategies. Second, we describe how we will evaluate the work through simulation and prototype development. We will disseminate our results through research publication, and open source software release.

2.1 Model

We present our preliminary model that we will use for developing GRID resource management and scheduling algorithms for the problem space of this proposal. This model will be adapted as our research progresses.

Each rigid job is described by a 2-tuple \((k_j, p_{k_j}^i)\), where \(k_j\) is a job size that is referred to as the job’s degree of parallelism or number of jobs’ tasks (processes, threads). \(p_{k_j}^i\) is the job \(J_j\) execution time on \(k_j\) processors. A processing time on the node \(N_i\) is calculated as \(p_{i,k_j}^j = p_{k_j}^i / b_i\). The job work \(w_j\) also called area, wait or cost of a job \(J_j\) is the product of the number of processors \(k_j\) assigned to the job and the execution time \(p_{k_j}^i\) of the job, \(w_j = p_{k_j}^i \cdot k_j\). Let \(r_j\) be a release time of the job \(J_j\), and \(c_j\) be its completion time.

On the first phase of the project we restrict our analysis to the scheduling systems where all the jobs are given at time 0 and are processed into the same batch. This means that a set of available ready jobs is executed up to the completion of the last one. All jobs which arrive in the system during this time will be processed in the next batch. A relation between this scheme and the scheme where jobs arrived over time, either at their release time, according to the precedence constraints, or released by different users is known and studied for different scheduling strategies for general or restricted cases. Using most basic results [SWW95], the performance guarantee of strategies which allows release times is 2-competitive of the batch style algorithms.

2.1.1. Two Levels Hierarchy Scheduling Strategies

The scheduling consists of two layers: the selection of the parallel node for jobs execution, and the local scheduling (Fig. 2.1).

We consider the following scenario. On the first stage, the broker analyzes the job request, current C-GRID resources’ characteristics, such as a load (number of jobs in each local queue), parallel load (sum of jobs’ sizes or jobs’ tasks), work (sum of jobs work), etc., and applies the selection strategy to select the best node for the job execution according to some optimization criterion.
2.1.2 Selection

In our model of the C-GRID environment, the selection is divided into two parts: the job selection from the set of jobs ready to be executed (available in the main broker queue) and resource selection from all available resources that can execute this job.

**Job Selection.** For the job selection parameters of the jobs can be used. In our model they are the job size, and execution time. If we have no knowledge about these parameters, FIFO strategy can be applied (Table 2.1). If we assume that the total number of jobs in the batch and the size of each job are available, then non-clairvoyant-batch strategies can be applied. If the total number of jobs in the batch and the job execution time of each job are available than clairvoyant-batch strategies are used. In the case of using both size and execution time of each job off-line-batch strategies can be applied.

<table>
<thead>
<tr>
<th>Job Selection strategies</th>
<th>Selection parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Job size</td>
<td>Batch job sizes</td>
</tr>
<tr>
<td>FIFO</td>
<td></td>
</tr>
<tr>
<td>Non-clairvoyant</td>
<td>Min SF (Min-Size-First, Narrowest-Job-First)</td>
</tr>
<tr>
<td></td>
<td>Max SF (Max-Size-First, Widest-Job-First)</td>
</tr>
<tr>
<td>Clairvoyant</td>
<td>Min TF (Min-Exec-Time-First, Shortest-Job-First)</td>
</tr>
<tr>
<td></td>
<td>Max TF (Max-exec-Time-First, Longest-Job-First)</td>
</tr>
<tr>
<td>Off-line</td>
<td>Min WF (Min-Work-First, Smallest-Job-First)</td>
</tr>
<tr>
<td></td>
<td>Max WF (Max-Work-First, Biggest-Job-First)</td>
</tr>
</tbody>
</table>

**Resource Selection.** We focus on the resource selections based on the model where the information about the size of each C-GRID node and its relative speed are known. We distinguish five levels of knowledge available for the resource selection.

**Level 1:** The length of the local C-GRID node queues (current local load) is available.
**Level 2:** The size of each job in the local queues is also available.
**Level 3:** The execution time (estimation of the execution time) of the jobs in the local queues is also available.
**Level 4:** The information about current local schedules (based on estimation of the job execution time) is also available.
**Level 5:** The information about current local schedules at runtime is also available.

On the Level 1 of knowledge we are only able to apply the following resource selection strategies: Min_L (Min-Load, Least Loaded Site, Shortest Queue), Min_LP (Min-Load-per-Proc, Site with Least Loaded Proc) (see Table 2).

On the Level 2 we are able to apply the following resource selection strategies: Min_PL (Min-Parallel-Load-per-Proc, Min-Threaded-Load-per-Proc).
On the Level 3, Min_LB (Min-Lower-Bound) strategy is available. On the Level 4 and 5 of knowledge we are able to apply the following resource selection strategies: Min_CT (Min-Completion-Time), Min_SWWT (Min-Sum of Weighted Waiting Time, where the weight is a job size or exec time), Min_SWCT (Min-Sum of Waited Completion-Time, where the wait is a job size or exec time), Min_WT (Min-Waiting-Time), Min_U (Min-Utilization), Min_ST (Min-Start-Time), Min_TA (Min-turnaround), Big_Free (Biggest Free), Min_BF (Min-Best-Fit), Min_FF (Min-First-Fit), BS (Best-Size), etc.

Levels 1-3 assumes that the resource selection is based on the information of job parameters that are available by broker, without their scheduling.

On the Level 4 we make the assumption that the broker builds preliminary local node schedules using estimation of job execution times the provided by users (selection schedule). It does not take into account possible jobs execution time fluctuation during their execution, and actual local scheduling policies. GRIDs are dynamic platforms. All these parameters can be unknown and change over time. Even local scheduling policies can be changed unpredictably because the GRID has no control over them. The parameters of the jobs already assigned to nodes, known by the broker, and one of possible local scheduling policies are used for selection only.

To motivate the introduction of the Levels 5 of knowledge available for the resource selection we can consider a model where the broker has information about actual local scheduling policies, and information from the node about current jobs execution, such as actual current schedule, node utilization, number of idle processors available to start the job immediately, actual mean waiting time, etc.

Table 2. Resource Selection Strategies

<table>
<thead>
<tr>
<th>Resource Selection Strategies</th>
<th>load</th>
<th>Job sizes</th>
<th>Job exec times</th>
<th>Cmax</th>
<th>Utilization</th>
<th>Waiting time</th>
<th>Turnaround time</th>
<th>Schedule</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Level 1</td>
<td>Min_L</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Min_LP</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Level 2</td>
<td>Min_PL</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Level 3</td>
<td>Min_LB</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Level 4</td>
<td>Min_CT</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Min_SWWT</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Min_SWCT</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Min_WT</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Min_U</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Level 5</td>
<td>Min_ST</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Min_TA</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Big_Free</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Min_BF</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Min_FF</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>BS</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Random** strategy chooses a node randomly (with uniform distribution) from a subset of nodes that are able to execute a job;
- **Min_L** strategy selects the node with the lowest load. This strategy is similar to static load balancing;
- **Min_LP** selects the node with the lowest load per processor (\( \min(n_i / m_j) \forall i = 1..m \)) where \( n_i \) is a number of jobs, and \( m_j \) is a number of processors in the node \( i \).
- **Min_PL** selects the node with the lowest parallel load per processor (\( \min(\sum_{j=1}^{n_i} k_j / n_i), i = 1..m \)).
− **Min_LB** strategy chooses the node with the least possible lower bound of completion time of assigned jobs that is the node with lowest work per processor \( \min_i (\max_j (\sum_j w_j / n_i, t_{\text{max}}^i)) \). Instead of the actual execution time of a job that is not available in non-clairvoyance scheduling, the value provided by the user at job submission, or estimated execution time is used. Notice that **Min_LB** strategy is based only on the parameters of already assigned jobs available to the broker, and does not need a construction of a partial schedule for each node.

− **Min_CT**. In contrast to **Min_LB**, the earliest possible completion time is determined based on a partial schedule of already assigned jobs [SKRS03, ZCAGGKPS04]. For instance, Moab [Clu] can estimate the completion time of all jobs in the local queue because jobs and reservations possess a start time and a wallclock limit.

− **Min_SWCT**. The node with the minimum of the sum of weighted job completion times \( \sum_j k_j c_j \) is chosen for selection strategy. By use of \( k_j \) as the weight wide size jobs may indirectly get a higher priority than others. This criterion has gained attention for the scheduling of parallel jobs. In the off-line batch scheduling Schwiegelshohn et. al. [SLWTY98], presented 8.53 approximation algorithm.

− **Min_WT**. The node with the minimum of the mean job waiting time is chosen.

− **Min_SWWT**. The node with the minimum of the mean job weighted waiting time is chosen where the weight is a job size). By use of \( k_j \) as the weight wide size jobs may indirectly get a higher priority than others to optimize mean job waiting time.

− **BS** (Best-Size) takes the node that leaves the least free resources of the node (least difference between the job size and node size). It prevents narrow jobs filling wide nodes causing wide jobs waiting for execution, but in the case of the specific job size predominance can cause overloading nodes with this size. In comparison to BestFit in [HSSY00], we do not use information about free resources available to start the job immediately;

etc

Parallel jobs scheduling problem for a given model of computational GRID can be considered as multiple strip packing problem, where number of strips is equal to the number of nodes with the sizes equal to the sizes of respective nodes. One of the issues of the resource selections and proper jobs allocation is to prevent narrow jobs filling wide nodes causing wide jobs waiting for execution. Several solutions are known.

### 3 Local Scheduling Algorithms

Several known scheduling policies are analyzed and compared in this project: **FCFS** (First-Come-First-Serve); **LSF** (Largest-Size-First); **SSF** (Smallest size first), **LETF** (Largest exec-time- first), **SETF** (Smallest exec-time-first), **LWF** (Largest work first), **SWF** (Smallest work first), **BackFill-FirstFit-Easy**, **BackFill-BestFit-Easy**, **BackFill-FirstSizeFit**, **BackFill-BestSizeFit**, etc.

**FCFS** (First-Come-First-Serve): It is one of the simplest strategies. The scheduler starts the jobs in the local list order. If not enough resources are currently available, the scheduler waits until the job can be started. The other jobs in the submission queue are stalled. This strategy is known to be inefficient for many workloads as wide jobs waiting for execution can result in unnecessary idle time of some resources.

**LSF** (Largest-Size-First) strategy in which jobs in the local queue are organized in non increasing order of their sizes. It is equivalent to the Bottom-Left-Decreasing (BLD).

**BackFill** is a scheduling optimization of FCFS which allows a scheduler to make better use of available resources by running jobs out of order. It has been implemented in several production schedulers. If the job at the head of the queue cannot be started due to a lack of available resources then system tries to find another job which can use the resources. Three types of backfilling are distinguished. Those that do not postpone the job at the head of the queue, those in which the job can be delayed on the acceptable length controlled by scheduling parameters, and those in which the job execution delay is acceptable.

In the first group there are two variants of backfilling, EASY backfilling and Conservative one, as described by Feitelson et. al. [FW98, SF03]. EASY or liberal backfilling guaranties do not postpone the job at the head of the queue only. Conservative backfilling does not increase the projected completion time of jobs submitted before the job used for backfilling, and requires more computational effort than EASY.
The relaxed backfilling [WMW02] belongs to the second group and explores the strategy in which the lower priority jobs can delay the highest priority jobs to the certain limit defined as a factor of the wait time of the highest priority job.

These first two groups are based on strategies in which the knowledge of the actual or estimated job execution time is used. The third one includes on-line strategies that do not use information about job execution time.

On the first phase of the project, the first group strategies BackFill-FirstFit-Easy, BackFill-BestFit-Easy, and the third group strategies BackFill-FirstSizeFit, BackFill-BestSizeFit are considered.

In the BackFill-FirstFit-Easy, if not enough resources are currently available for a job, the scheduler looks for the first job down in the queue that can use available resources, but not postpone the execution of the job in the queue head.

In the BackFill-BestFit-Easy, the scheduler looks for one or several jobs that can best use available resources (best job areas combination) under the premise that the head queue job is not delayed.

In the BackFill-FirstSizeFit, the scheduler looks for the next jobs down the queue that can use available resources, assuming that it can delay current unscheduled job execution.

In the BackFill-BestSizeFit, the scheduler looks for jobs in the queue that can best use available resources (best job sizes combination).

On the first phase of the project, for the worst case analysis we choose FCFS, LSF scheduling strategies and the following selection strategies: Min-L, Min-PL, Min-LB, and Min-CT, based on non-admissible and admissible (-a) algorithms.

4. Experimental Analysis

Often competitive analysis cannot be successfully applied to strategies which are based on complex algorithms, and real system/application parameters. Moreover, competitive factors are worst case factors that frequently do not occur in real systems. Simulation currently is the only feasible way many large-scale distributed systems of heterogeneous resources with different resource selection strategies can be evaluated. Simulation is effective when working with very large problems that involve a large number of resources and users. Simulation makes it possible to explore different types of systems operating under varying workloads and algorithms.

In the project proposals, an analysis of the worst-case behavior of proposed strategies is complemented by an analysis of their mean behavior. We plan to perform a detailed performance evaluation varying factors affecting hierarchy scheduling: selection strategy, local scheduling scheme, workload parameters, C-GRID configuration, and real workload traces. For performance evaluation of the different algorithms and admissible resource selection strategies we used simplified simulation environment named Simula. It allows the evaluation of different configurations by providing results for common evaluation criteria, like schedule-length (makespan), average response-time, resource utilization, and other performance metrics. Given a C-GRID setup and a set of real workloads, the system simulates what would happen in the Grid if the different job selection, resource selection and local scheduling strategies are in use. It provides us with a set of measurements used to quantify the effectiveness of the strategies.

4.1. Evaluation Criteria

For performance evaluation of the different strategies we used common evaluation criteria:

− Competitive ratio defined as ratio of the completion time over the lower bound of the optimal solution
  \[ \rho = \frac{C_{\text{max}}}{C_{\text{opt}}} \], where \( C_{\text{max}} \) is the completion time, and \( C_{\text{opt}} \geq \max(\sum_{j=1}^{n} W_{j}, W_{\text{idle}}) \), where \( p_{\text{max}} \) is the longest job execution time, and \( W_{\text{tot}} = \sum_{j=1}^{n} W_{j} \).

− Processor utilization \( U = \frac{W_{\text{tot}}}{(W_{\text{tot}} + W_{\text{idle}})} \), where \( W_{\text{tot}} \) is the total work of C-GRID, and \( W_{\text{idle}} \) total amount of resources that not used during jobs execution.

− Throughput: number of jobs completed in C-GRID per unit time: \( n/C_{\text{max}} \).
\(-\) Turnaround (response) time: mean time from submission to completion of jobs: \( \sum_{j=1}^{n} t_j^f / n \), where \( t_j^f = C_j^f - r_j^f \).

The criterion is the user-centric, since it is important from the user's viewpoint. Its minimization yields a minimization of the mean response time of the scheduled job set;

\(-\) Waiting time: mean time spent ready to run but not running job, so \( \sum_{j=1}^{n} t_j^w / n \), where \( t_j^w = t_j^f - r_j^f \).

\(-\) Response ratio is mean ratio over all jobs \( \frac{1}{n} \sum_{j=1}^{n} (t_j^w + p_j^f) / p_j^f \), where response ratio defined for each job as:

\( (t_j^w + p_j^f) / p_j^f \), where \( p_j^f \) is the execution time, and \( t_j^w \) is waiting time of job \( J_j \).

5. Future work

Research Problem. Preliminary results motivate finding approximation bounds of other two or several layer hierarchy scheduling strategies that can better predict algorithms behavior.

Research Problem. It is also important to study different algorithms for the more variety of local scheduling strategies.

Research Problem. Another important question is how fuzzy execution time affects the efficiency. Several factors impact on the execution time fluctuation: overestimation of time provided by users, variation of a job input, variety of processor speed, memory constraints, etc.

Research Problem. It is also important to study job scheduling including data/communication costs. Network topology, available bandwidth between the different sites, and estimation of the required transfer cost have to be taken into consideration when assigning jobs to Grid sites.

References


BDE00 J. Blažewicz, M. Drozdowski, K. Ecker, “Management of resources in parallel systems”, in the Handbook on Parallel and Distributed Processing, Springer Verlag, 2000, p.263-341


Clu www.clusterresources.com


PWA  Parallel Workloads Archive www.cs.huji.ac.il/labs/parallel/workload/


TT02  A. Tchernykh, D. Trystram. Adaptive Strategy for On-Line Scheduling of Real World Parallel Applications.. SOCAL’02 Southern California Workshop on Parallel and Distributed Processing and Architecture, Santa Barbara, USA, p. 7-8., 2002


WL03 Chuliang Weng, Xinda Lu: A Cost-Based Online Scheduling Algorithm for Job Assignment on Computational Grids. APPT 2003: 343-351


WLMFN96 K. Windisch, V. Lo, R. Moore, D. Feitelson, B. Nitzberg, A Comparison of Workload Traces From Two Production Parallel Machines

