Analysis of Grid Schedulers with Resource Allocation

Ovidiu Gherman, Oana Vultur
Ștefan cel Mare University
Suceava, Romania
ovidiuh@stud.usv.ro, vultur_oana@yahoo.com

Abstract: Scheduling algorithms for grid systems are important parts of the management of distributed systems. It allows using efficiently the available resources and ensures an adequate level of quality of service to its users. There is a constant need of better optimized algorithms, taking into account a plethora of parameters to implement an execution schedule that is both efficient for the system (minimizing the idle state of the resources) and provides computation services that are both easy to use and fast. We propose such an algorithm and implement it in a grid simulator for testing.

Keywords: scheduling algorithm, grid simulator, backfilling, QoS parameters.

1. Introduction

Scheduling jobs for execution on grid infrastructures represent a critical component of today’s distributed systems. This allows an efficient use of the hardware and software resources while offering non-trivial QoS to the users. The main purpose of a scheduler is to create a scheme for distribution of tasks towards the available resources and to ensure mapping, while using efficiently the resources at hand. The user employs transparently a resource broker that will discover the available resources and will negotiate certain services according to the resources cost and user’s necessities (and its status and access rights). The same broker (using specific services) is responsible to map the tasks on the scheduled resources, to deploy the application and to gather the results. The trace of the broker can be analyzed and used to further enhance the efficiency of the scheduler (in resource availability and scheduling optimization), but also to pinpoint potential problems in grid structure and architecture or even the modules that reduce performance and can be improved.

The article is structured as follows: Chapter 2 presents some background information about grid schedulers and QoS parameters, Chapter 3 presents a grid simulation engine and front-to-end that will be used to implement and test a grid scheduler, Chapter 4 contains experimental results and Chapter 5 presents the conclusions.

2. Scheduling with QoS parameters

2.1. Schedulers

Computational grids are highly dynamic systems that pose significant problems for job scheduling because of the heterogeneous and distributed nature. Usually a certain level of performance must be maintained, taking into consideration the QoS requirements and the reliability/robustness of the system (hardware and especially software).

Task scheduling can have - usually - one of the two goals: high performance computing (used for minimizing the execution time of discrete tasks in parallel program or HPC) or high throughput computing (to offer increased availability to various tasks on a long period of time in non-dedicated grids). Mapping tasks on resources according to a schedule can be done in the moment of receiving the job (on-line mode) or when a certain number of jobs have been received (batch mode). If in the first case the scheduler allows more flexibility and a decreased response time, in the second case a more efficient schedule can be created because it can be altered dynamically for new tasks. The scheduler itself can be centralized, hierarchical or decentralized (with certain advantages and disadvantages). It is also possible to create hybrid schedulers that use different characteristics of the mentioned types.
Most of the scheduling techniques used in computing grids are based on queues, easy to implement but not very effective in mapping jobs after well determined parameters. Algorithms such as FCFS (First Come, First Served), EASY Backfilling or Flexible Backfilling are routinely employed [1] in the grid systems. If a certain user needs a specific resource (for performance reasons or special requirements) usually s/he has to log on directly on specific resource and to shun the scheduling service.

A more advanced method is to have a schedule-based approach. This allow advanced reservation of resources, dynamically remapping jobs on more appropriate resources and optimization of the schedule based on optimization methods (such as local search-based methods). This approach is usually most effective when the submitted jobs are in batch mode (a new task will trigger either rescheduling, either will be placed in a temporary queue if it has a low priority, until a significant amount of jobs are queued, depending on the scheduling algorithm used).

Several policies are commonly employed in the schedule-based approach, for example Greedy techniques, EG-EDF (Earliest Gap - Earlier Deadline First) that fills the earliest gaps in the schedule with newly arrived jobs (if not found, the job is placed accordingly its deadline) and Tabu Search (it moves again certain jobs into suitable gaps - if found - optimizing a local solution without rebuilding an entire schedule). This approach it is usually more effective than a queue-based one [1].

After a schedule is created, the tasks are mapped on individual resources (in a processing pipeline). Their execution will start according to the start time of each task (and its priority in conflictual cases). The tasks will be processed by the local mechanism (local policies must be taken into account - the system has its own computational necessities).

A higher machine usage is beneficial for the resource provider (meaning less wasted resources and more available systems for unexpected loads and contingencies) while a higher non-delayed jobs means a better QoS experience for the users. The scheduler must take into consideration both positions.

2.2. QoS attributes in scheduling

A service provider usually will agree with a user on the terms of QoS through a Service Level Agreement (SLA) that will specify conditions of service and the QoS metrics that must be obeyed by the provider. Typically metrics are service-level (that is - the user will specify a certain goal, like a deadline, to be achieved without knowing the platform's physical characteristics) and the provider can allocate dynamically its resources [3].

But this is not always the case. In non-commercial systems the QoS specification often must account for users and job priority (certain users have authority to access more resources and preferential executions rights) and for special requirements (such as a specific platform like QS22 Cell-based [4]. In such cases, we propose that a tiered QoS can be used (users receive a certain level of QoS based on what class they belong to). Its provisions can determine even the dynamic rescheduling if a
job with higher priority (or from a user on a better tier) will be submitted. Of course, a mechanism that will prevent starvation - indefinite postponing of the job (such when a queue of high-priority jobs arriving constantly) based on lack of immediate available resources - must be implemented (for example a token - based on the deadline of the job and its initial priority - that expires after a certain time period or after a number of delays).

Another problem with the SLA's is that in certain systems where the metrics are based on physical characteristics (certain CPU cycles or performance are allocated or a certain level of availability is expected) a part of the resources are blocked through over-provisioning (resources put on hold either as a backup or underused) and cannot be used (usually through backfilling or dynamic rescheduling) by other jobs since it will broke the SLA on the provider's part. To avoid such disadvantages, the non-commercial systems can employ a reputation-based ranking for the resources and to classify the user in predefined tiers that will make easier to schedule the incoming jobs.

Moving from the classical staples of QoS (attributes like availability, reliability, throughput, latency, etc.) could be beneficial in non-commercial grids as it makes easier for a user to select the proper level of QoS to be expected and lowers the gap between the objective system quality and the subjective perception of quality.

In ensuring an adequate level of compliance with the QoS requirements, a method of detecting a (more) compliant service provider is to rank the providers according to their reputation. This attribute must take into consideration the user's perception of quality (unreliable by nature but the most important in its satisfaction) and the rating of the system (that determines objectively if the QoS specifications where met, although with certain provisions - this is usually a non-trivial task); it measures the trustworthiness of the service [5].

We propose that this method (employed especially in commercial infrastructures) can be extended in the realm of grid resources. A certain resource can be given a rating based on its past performance (this must include failures, availability, completion of jobs in the scheduled period). A scheduler that will use this reputation rating will have a faster access to relevant information on the probability of meeting the defined QoS parameters. Of course, this rating is dynamic and must be performed at certain intervals or after a number of concluded tasks (either failed, either solved), for every resource (this can be computed locally, to decentralize partially the scheduler). In this case, the scheduler will be the beneficiary of the reputation attribute, viewed as a tool to establish a better and reliable schedule.

Moreover, aside rating the system (from user's perspective or from operational logs), the users themselves can be rated by the system. Knowing the user's behavior in the system and evaluating it (the rating process will consider user's time table, it's effectiveness in evaluating the execution time for purpose to correct the overestimation or underestimation, how often its programs will fail because of bad programming or incompatibilities with system local dependencies, settings and conditions, etc.), the system can increase the probability of meeting the deadline and therefore the chances to abide by the defined QoS parameters. Of course, this rating is updated dynamically and is more precise with the number of jobs submitted by the said user (the probability of meeting the deadline can be expressed as a rapport of the successfully completed jobs and the failed ones; near-completed jobs can also be taken into account as signaling potential venues to improve or upgrade certain resources - an easy way to uplift the resources and to increase, even marginally, the local performance).

A critical step in modern scheduling techniques is the precision of estimating the job completion time. Usually this is done by the user itself (based on his analysis of the source code and the relative performance - or platform's equivalent performance - of target-machines). There are methods to compute automatically the completion time such the short term prediction model (employed in AppLeS) where the scheduler observes the execution times for the fastest jobs (in the span of minutes usually) and corrects its initial estimations, or the long term prediction model (GHS - Grid Harvest Service) [6]. The latter methods are not as widespread as the former.

To enforce the QoS expectations, a guided QoS component is added in the conventional scheduling algorithm and the most recent performance prediction (in non-dedicated grid systems) is used to estimate a more precise execution time.

The QoS provisions can be defined as loose guaranties and strict guaranties [7]. In the first case, the user can select a provider (offering a best-effort QoS) that can offer (statistically) a QoS level that meets the users' necessities. The QoS assurance component relies on monitoring
critical performance parameters over time and to act accordingly (on a statistical basis) through estimation. The nature of loose guarantees preclude the need for a SLA (simplifying the submit process) and permits future adjustment of the scheduling strategies for a better (always suboptimal) performance. This approach is used especially in non-commercial grids.

In the second case, an exact QoS profile is defined, based on user's expectations and provider's offerings. The profile is contracted as a SLA between the two parties. It is not adequate for most non-dedicated grids.

3. Grid simulators for scheduling

Experimenting with large numbers of grid scheduling scenarios is impractical in real world systems because of the limited number of various resources available for testing sessions. Moreover, the grid environment it is not controlled so multiple repetitive experiments are hard to deploy. This impact negatively the experimenter's capacity to evaluate the scheduling algorithms and strategies employed.

A testbed platform (a limited size prototype) can be used for testing, but is has visible disadvantages: it is small, quite expensive; it is time consuming to build and not too diverse. The possible tested scenarios are limited in scope and size. To avoid this, a framework for deterministic modeling and simulation of resources, applications, users and algorithms must be used for scheduling strategies evaluation. The most common way to achieve this is using a software simulator that will provide the necessary tools for modeling, simulation and visualization of the result data.

3.1. GridSim Toolkit simulator

Such a framework is the GridSim Toolkit [8] that is capable to simulate resources as single and multiprocessors, shared and distributed memory machines and various network topologies and connectivity, either as PCs, SPMs or clusters in various configurations (parallel and distributed -like computing grids and P2P networks) [9]. The simulator has functions to implement applications, to obtain information and to map tasks to resources (and subsequent management), but also tools for data visualizations and to trace the simulated environment's footprints. The resource broker simulates the design of Nimrod-G scheduler (cost and time optimization for budget constrains or with deadline jobs). The toolkit can use simulated and real workloads in common formats (.gwf) and can record the traces.

A scheduler must be able to discover resources, to evaluate their QoS potential and to select the appropriate ones for the active jobs. The schedule must be optimized to answer to the user's requirements but without wasting precious resources (energy, CPU cycles, memory space, availability, etc.). A well written schedule must also be dynamic (to respond to modification in system's status - new jobs, jobs unfinished, new resources, etc. - considering the QoS level of the user and its priority compared with the other users that submitted jobs).

Some features implemented in GridSim Toolkit are listed here [9]:

- the possibility to define the capability of resources (Processing Elements) as MIPS (Million Instructions per Second) with multiple PEs creating a machine; the resource (multi-machine) is the basic unit of the simulator (various resource types can be defined, with unique characteristics and performances);
- the possibility to define a detailed time table (calendar) for job submitting and for resources (time zones);
- advanced reservation of resources;
- the specification of task types (CPU, memory or I/O intensive);
- time-sharing of processes (modeling multiple simultaneous processes on the same resource);
- the possibility to define the interconnect network speed;
- the capability to work with static/dynamic and centralized/decentralized schedulers.

The most common scheduling algorithm used are Round Robin and FCFS (First Come, First Served) with backfilling [9]. Most schedulers take into account the QoS parameters of the user but also the specific characteristics of the said user: the type of job (usually an estimated execution time), the deadline, the budget constraints, the time-zone of the user, special requirements or its activity on certain periods of time. The brokers employed are instanced for every users and compete according to the user priority, requirements and job's parameters and can contact directly (using specific information services) the resources to obtain their static and dynamic properties.

The GridSim simulator regulates the information flow among its entities by input and output entities (via I/O channels and ports) in a parallel manner. The communications can be modeled with real-world provisions (delays, losses, etc.). All of the entities' interactions are
implemented by events (internal and external) that can be synchronous (if a certain response to the event is expected) or asynchronous (when no response is expected after the event).

GridSim has a multi-layered architecture [10] on top of the scalable Java interface and the JVM. A second layer is based on SimJava, an discrete event-driven engine that shapes resources, users and brokers as entities communicating via messaging events (synchronous and asynchronous). These entities runs in parallel, in their own threads, as described by three simulation rules (schedule - event messaging via ports; hold - holding the virtual resources in simulation; wait - waiting for event messages). The GridSim user must only write a Java program that will call the appropriate libraries to model the grid elements that require simulation and to collect the simulation statistics. The third layer is the GridSim Toolkit itself. The last ones are the grid scheduler layer and the user modeling layer (Figure 2).

Fig. 2. GridSim architecture [9].
3.2. Alea 2 simulator

Alea 2 is an event-based simulator used for testing scheduling algorithms under various conditions (different resources and job types, dynamic changes in the environment, etc.).

This grid simulator is based on GridSim Toolkit (presented above) and represents an extension that incorporates better tools for scheduling algorithm implementation (similar to real-world solutions such as PBSpro and LSF), visualization capability and a higher speed of simulations. The scalability is also improved.

The main component of the simulator is a scheduler entity that incorporates several predefined algorithms (queue and schedule-based such as FCFS, EDF, EASY Backfilling, EDF-Backfilling and others) and additional classes for extended functionality [1].

The Alea 2 has some improved modules from the underlying GridSim – a grid job allocation policy, a loader of failures and a loader for machines and jobs. The interaction with the user is also improved. This functionality is obtained by implementing child classes over the parent classes of GridSim. This also allows for easy extensions of the simulator.

3.3. Improved algorithm for scheduling

A new algorithm is proposed to schedule the incoming jobs. Users send jobs dynamically (the scheduler system does not know in advance the submit time of the job, nor its characteristics). When the job is received, a new schedule is computed. If the new schedule has a higher quality (established by comparing with the old schedule based upon an acceptance criterion), it is used.

The new job can be inserted in an empty slot in the old schedule (a backfilling approach), providing that such a slot – with enough free processing units and correct characteristics – exists, or can be appended in the execution queue of the first available resource (if a slot is not available). Alternatively, the entire schedule can be scratched and rebuilt as to ensure a higher quality. This is usually a computationally expensive and time consuming operation, so it is preferred to be done scarcely.

The proposed algorithm tries to schedule efficiently the incoming jobs, while keeping the QoS constraints and provides a better (uniform) job distribution among the available resources. This ensures that the system remains responsively for longer and its usability for users increases.

The algorithm works as follows:

- a new job arrives;
- the available resources are pooled together in a list;
- the list is sorted accordingly their number of free processing units
- the algorithm search for the elements with a maximum amount of free units and compatible with the job's requests;
- if a resource is found, must search for a available slot in the said resource;
- if the slot is found, the new job is placed (backfilling);
- the new schedule is compared with the old one;
- if it is better, the schedule is updated with the new reservation;
- if it is worse, the new schedule is scratched and the iteration of resources continues;
- if no free slot is found, the job is appended to a resource, in its queue (as last job).

4. Experimental data

The algorithm was implemented in the Alea 2 grid simulator for testing. The settings assume that a job can span a resource, but cannot span multiple resources. A resource can run concurrently multiple jobs if there is enough processing elements to support the jobs' requirements.

The tested set contains 12000 jobs taken from a BlueGene cluster (1152 processors) and scheduled for execution, processed by a FCFS algorithm and by the algorithm proposed.

It can be observed that in terms of waiting jobs versus running jobs, the earlier algorithm have more queued jobs – meaning that more jobs can be delayed from execution and consequently the QoS level is lower (the quality of service restrictions are not meet). Moreover, because of better scheduling capabilities, the requests for processing units by the incoming jobs are much better balanced and the jobs are better placed to be executed and terminated on time. The FCFS algorithm scored (from 10731 accepted jobs) 2464 not-delayed jobs when the proposed algorithm scored 7268 not-delayed jobs. In the second case, the quota of on-time jobs is much higher, so a better QoS is achieved (from 22.96% not-delayed jobs on the queue-based scheduler, to 67.72% not-delayed jobs on the schedule-based scheduler). Although the computation time for the schedule is greater than that for the queue distribution, the performance jump justifies the adoption of a schedule-based algorithm.
Fig. 3. Results for FCFS algorithm.

Fig. 4. Results for the proposed algorithm.
5. Conclusions

It can be observed, from the experimental results, that the planning algorithms based on schedules are superior to those based on queues in terms of QoS assurance. However, there is usually a penalty to be paid — the queue-based algorithms are simpler and faster. The FCFS algorithm had a total scheduling time (for the entire job input) of 18783 ms, when the proposed algorithm (schedule-based) had 671621 ms. In practice however, jobs can come inconsistently, so an schedule-based algorithm could function without much delays.

If we consider the ubiquity of the grid and distributed computing systems, there is a need to discover new and improved algorithms for planning the jobs, so that the users can have easier and faster access to the needed computation resources. Optimization of resource usage is also a priority since the grid systems are huge and, consequently, hard to manage efficiently.

Acknowledgement

The testing was done on the Alea 2 grid simulator, provided by [2],[11].

References