Two Stage Optimization of Job Scheduling and Assignment in Heterogeneous Compute Farms.

Lev Markov,
Sun Microsystems, Inc.
lev.markov@sun.com

Abstract

Distributed networked computing in a compute farm environment has attracted great attention in recent years. Specialized management system for a compute farm enables heterogeneous distributed resources to be shared in a seamless way between various competing jobs. A key functionality of such system is a scheduler that controls the assignment of jobs to resources. This paper outlines a range of scheduling constrains as well as a list of required scheduling features for a state-of-the-art management system in distributed farm computing. It also presents a novel two stage static-dynamic scheduling algorithm to deal with the scheduling complexity.

1. Introduction

Distributed networked computing in a compute farm environment has attracted great attention in recent years [1,2,3,4]. A compute farm is an environment in which any number of heterogeneous compute resources, like workstations, servers, storage arrays, are networked together with specialized management system to form a single entity available to all users. Compute farm hardware by itself cannot ensure that enough computing power is available when needed, and that computing resources are not over or underutilized. Specialized management system for a compute farm enables heterogeneous distributed resources to be shared in a seamless way between various competing jobs. It optimizes utilization of software and hardware resources in a networked environment and distributes computational workload across networked servers and workstations in order to simultaneously increase productivity of machines and application licenses while maximizing the number of jobs that can be completed.

A key functionality of a management system is a scheduler that controls the assignment of jobs to resources. It analyzes the pending workload and the available computing resources to create a schedule, a time-ordered assignment of jobs to resources. Both the efficiency and the flexibility of a distributed computing depend critically on the quality and features of the system scheduler. Scheduling for a heterogeneous networked environment is complex since it involves scheduling over two dimensions, time and space, and on two levels, jobs and computing resources [2].

There are many different approaches to job scheduling in networked environment [1,4] but none of them is capable to deal with an entire wide range of constraints and requirements presented by the compute farm users. In many cases system administrators have a great deal of responsibilities in making complex manual decisions related to job/resource management. They are required to set or change job priorities as well as to select the most appropriate resources for each job with a very little help of automation. This task becomes extremely complex in case of data or time dependencies between jobs, various job requirements including deadlines, quite non-uniform nature of the computing resources, etc.

This paper outlines a range of scheduling constraints as well as a list of required scheduling features for a state-of-the-art management system in distributed farm computing. It also presents a novel two stage static-dynamic scheduling algorithms to deal with the scheduling complexity. The scheduler continually re-evaluates (with a specified frequency) all of the submitted jobs and selects them for an assignment based on globally calculated static job priorities. For each job the best suitable resource is calculated dynamically taking into account a current status of already scheduled jobs.

2. Problem specification

In general terms a job scheduling and assignment problem can be stated as a problem of selecting a resource for every job from the pool of submitted jobs (a job assignment task) and finding for every resource the best order of the jobs assigned to it (a job scheduling task). Obviously, any solution has to satisfy all resource and job constraints and requirements.

Proceedings of the 10th IEEE International Workshop on Future Trends of Distributed Computing Systems (FTDCS'04)
0-7695-2118-5/04 $20.00 © 2004 IEEE

Constraints and requirements for the resource management system are separated into two main groups. The first group represents constraints imposed by the natural job-to-job, resource-to-resource and job-to-resource relations. These constraints are dealing with 3 different type of properties: job properties, resource properties and job-to-resource relation properties.

Examples of these groups of properties are:

Properties of jobs.

Initial priority
Dependence on other jobs (time or data)
Processor type required
Licenses required
Partitionable into parallel sub-jobs
Preemptability (can a job be stopped)
Restartability (can a job be restarted)
Completion deadline

Properties of resources.

Processor type
Licenses available
Memory size
Number of processing slots
Link bandwidth with other resources

Properties of job-to-resource relations

Processing speed
Memory required
Number of processing slots required

The second group of constraints represents constraints dealing with desirable scheduling features and capabilities imposed by the users. Examples of required scheduling features are:

Job advance reservation
Job back filling
Job preemption
Parallel job partitioning

It has to be noted that constraints from the first group determine the mathematical model to be used for the scheduling problem, while constraints from the second group determine algorithms to be selected. Both the mathematical model and the algorithms have to be flexible enough to be able to incorporate additional constraints and requirements.

In choosing among alternative job assignments and the corresponding feasible schedules, the scheduler minimizes a cost function. The cost function has to be flexible enough to be changed and/or adjusted depending on the administrator's requirements. There are many examples of an acceptable cost function, e.g., total processing time, average job waiting time, maximum deadline violation, or a weighted combination of these costs.

3. Scheduling implementation

The mathematical model for the job scheduling and assignment is represented by two partially connected graphs. The first directed graph deals with the jobs (job graph) and the other non-directed graph deals with the resources (resource graph). The nodes in the job graph represent individual jobs while the directed links between nodes represent data/time dependencies. Each link has an associated weight representing a time delay or amount of data to be sent from the source job to the sink job. The job graph is a partially connected graph with links only between inter-dependent jobs. The nodes in the resource graph represent individual network resources, while the links between them represent available data channels. Each link has an associated weight representing different quality of the channels. Each node in the resource graph has attached available capacities of the corresponding resource, i.e. memory size, number of processing slots, license availability, etc. Each node in the job graph has attached processing requirements and constraints of the corresponding job, i.e. permission to be preempted, way of being re-started, permission of parallel assignment into multiple queues, application licenses needed, etc. Each node on the job graph has an association with part of the resource graph. The part of the resource graph corresponding to a node in a job graph represents resources suitable for a given job. It is called a suitable resource sub-graph. Any two suitable resource sub-graphs corresponding to different nodes of a job graph can overlap.

The goal of the job scheduling and assignment task is to find for each node in the job graph an appropriate node/nodes from the corresponding suitable resource sub-graph in such a way that all constraints attached to the nodes on the job graph and resource graph are satisfied and the cost function is minimized.

A state-of-the-art resource management system has to be able to deal with several important scheduling features. Some of them have a very significant effect on the complexity of the algorithms as well as on the quality of the results. The most effective scheduling features are automatic partitioning for large parallel jobs, preemption capability of low priority jobs by jobs with higher priorities and advance reservation for high priority jobs with back filling by low priority jobs. Dealing with advance reservations will also resolve any issue related to the job starvation problem.

Partitioning of a large parallel job creates several corresponding sub-jobs that are assigned to several resources so that they are processed in parallel. These sub-jobs must be run concurrently because they may need to communicate with each other. Obviously, if a parallel

sub-job is preempted, the scheduling algorithm has to preempt processing of the other sub-jobs as well.

A job preemption capability allows a high priority job to preempt processing of a low priority job. Not every job can be preempted. Ability to preempt any given job is controlled by the corresponding parameter attached to the job node in the job graph. Also, even if it is allowed to preempt a given job, the preemption itself can take place only in very specific situations. There are several global parameters specified by the system administrator that control preemption situations. Two parameters controlling preemption strategy are the minimum ratio between priorities of a preempting job and a job to be preempted and the minimum ratio between execution time remaining and execution time already invested. Preemption of a job will not be allowed if its processing is "almost" done. A third parameter controls assignment of a job that may be preempted in the future. This parameter is the minimum ratio between execution time spent before a job is preempted and the total required execution time. If the ratio is too small for a job to be started in the particular open time spot, it is not started.

Advance reservation for high priority jobs allows an assignment of resources that don't have enough processing capacity (memory and/or processing slots) at a given open time slot, but will be capable of processing in the future. This situation can arise if a high priority job also requires a large amount of memory and/or processing slots and some of the resource capacities are consumed by already pre-assigned jobs. In such a case, a high priority job will be assigned to the resource but it will be scheduled to start after the required resource capacities are freed. A back filling capability allows the idle time created by an advance reservation on a resource to be filled with a number of low priority jobs. The back filling has to be done in such a way that the filling jobs do not prevent the job with an advance reservation from being started on time. This can be achieved if the filling jobs will be completed in time or if they can be preempted.

Advance reservation is the key strategy to avoid a problem called "job starvation" experienced by many management systems. A term of "job starvation" addresses the situation in which low priority job assignments repeatedly delay scheduling a high-priority job with high processing requirements. If there is not enough resources to schedule a high-priority job at the time when it is ready, then it will be scheduled at the earliest feasible "advance" time and the gabs will be filled with low-priority jobs.

In order to deal with size and complexity issues of the job scheduling and assignment task, we implemented it as two stage optimization process. This constructive heuristic approach uses on both stages priority functions calculated with a global view on the entire problem. In the first stage of the process, every node of the job graph gets a static priority calculated based on the weighted trade-off costs associated with the corresponding job. In the second stage of the process, job nodes are taken according to their static global priorities and assigned to the best suitable resource selected based on the dynamic priorities. For every job selected for an assignment all of the suitable resources are ordered based on the global dynamic priorities recalculated for every new job assignment. A resource with the highest dynamic priority is used for the selected job. This process is repeated for every non-assigned job.

The trade-off costs used to calculate static job priorities deal with job required memory, number of required processing slots, available deadline slack, job waiting time and initial job priority set by the system administrator. Importance weights between different trade-off costs are flexible and easily modifiable. Dynamic resource priorities for every selected job are calculated based on the current resource load, total demand on a given resource and the earliest time slot available for the job assignment.

The entire scheduling and job assignment procedure is repeated with a frequency specified by the system administrator, e.g., every 40-50 seconds. Any job partially executed during the previous scheduling cycle and not finished before the beginning of the current scheduling cycle will be marked as a "pre-assigned" job and will continue execution without being disturbed (although it may be preempted if so allowed). On every scheduling cycle estimations of run times for every presented job are used. These estimations could be adjusted from cycle to cycle if it is required. Because the scheduler is being called frequently to build a schedule based on changing workload and resource situations, the speed of the algorithm is important.

4. Scheduling example

The algorithm described above has been implemented within Sun Grid Engine environment. The following example can give a good illustration of the power and flexibility of the algorithm.

Job description:

18 sequential jobs (with job ID numbers from 1 till 28) are presented to the scheduler. The first 17 jobs require 1 processing slot and 50 MB of memory. A job #18 requires 3 processing slots and 100 MB of memory. The first 5 jobs (numbered from 1 till 5) were scheduled during the previous scheduling cycle and are already running on all 5 machines of the compute farm.

Also, the scheduler is presented with a large parallel job #19 which requires 5 slots and
600MB of memory.

Any job preemption is not allowed. There are no data or time dependencies between jobs.

Compute farm:

Compute farm consist of 5 machines. The first 4 machines have 1 slot and 128 MB of memory each. The last machine has 3 slots and 512 MB of memory.

Static job priorities:

Static global job priorities were calculated by the algorithm. Based on the calculated static priorities the jobs were ordered in the following way (starting with the highest priority job and finishing with the lowest priotity job):
#1, #10, #2, #3, #4, #5, #18, #6, #19, #7, #11, #12, #13, #8, #14, #15, #16, #17, #9

Scheduling process:

To ilustrate the scheduling process a few key steps of the algorithm are presented:

1. Already running jobs are assigned back to the corresponding queues (see Table 1).
1. The next job in the priority list (job #10) is selected to be scheduled. Queue #5 is found to be the best queue for this job. The queue has enough resources to start job #10 at time 0. Finish time is 10800.
2. The next job in the priority list is a job #18. This job needs a lot of resources. These resources are available only on queue #5. But this queue is busy until time 10800. So, an advance reservation is made for the job #18. Start time is 10800 and finish time is 14400.
3. Job #6 is assigned on queue #4 starting at time 9600. Finish time is 14400.
4. The next job in the priority list (job #19) is selected to be scheduled. This is a large parallel job. It is automatically partitioned into 5 smaller jobs. These jobs are scheduled on all 5 machines starting 14400 (this is the time when the job #18 is estimated to be finished on the queue #5). Table 2 presents a partial schedule after the job #19 is scheduled.
6. The rest of the job are assigned and scheduled on different queues to backfill available time slots created as a result of advance reservation made for the job #18.
   The well balanced final schedule is presented in the Table 3.

5. Conclusions

A successful management system in heterogeneous compute farm demands a highly scalable advance job scheduler. The scheduler has to be flexible enough to be able to incorporate a wide range of constraints and requirements presented by complex compute environments. This paper presents a novel two stage static-dynamic scheduling algorithm to deal with scheduling complexity of heterogeneous compute farms.

A step-by-step scheduling example is presented to illustrate the key steps of the algorithm. The algorithm allows to created optimized, well balanced schedules for distributed networked computing.

The algorithm is implemented for the new generation of Grid Engine software within Sun Grid Engine[TM] environment.

References

1. http://gridengine.sunsource.net/

2. D.G Feitelson, L.Rudolph, U.Schwiegelshohn, K.C.Sevick, P.Wong, “Theory and practice in Parallel Job Scheduling”. In Job Scheduling Strategies for Parallel Processing,pp.1-34, 1997.

3. C. Ernemann, V.Hamscher, U.Schwiegelshohn, R.Yahyapour, “On Advantages of Grid Computing for Parallel Job Scheduling”. In 2nd International Symposium on Cluster Computing and Grid, pp.39-46, 2002.

4. J. Krallmann, U.Schwiegelshohn, R.Yahyapour, “On the Design and Evaluation of Job Scheduling Algorithms”. In 5th Workshop on Job Scheduling Strategies for Parallel Processing, pp.17-42, 1999.

Appendix

Table 1: Partial schedule after step 1

Queue	*Assigned Jobs*	*Job Start Time*	*Job Finish Time*
1	3	0	10800
2	5	0	9600
3	1	0	8400
4	4	0	7200

Queue	Assigned Jobs	Job Start Time	Job Finish Time
5	2	0	6000

Table 2: Partial schedule after step 5

Queue	Assigned Jobs	Job Start Time	Job Finish Time
1	3	0	10800
	19-1	14400	18720
2	5	0	9600
	19-2	14400	18720
3	1	0	8400
	19-3	14400	18720
4	4	0	7200
	6	9600	14400
	19-4	14400	18720
5	2	0	6000
	10	0	10800
	18	10800	14400
	19-5	14400	18720

Table 3: Final schedule after step 6

Queue	Assigned Jobs	Job Start Time	Job Finish Time
1	3	0	10800
	19-1	14400	18720
	17	18720	21120
2	5	0	9600
	19-2	14400	18720
	16	18720	21120
3	1	0	8400
	19-3	14400	18720
	15	18720	21120
	13	21120	21720
4	4	0	7200
	6	9600	14400
	19-4	14400	18720
	9	18720	21120

Queue	*Assigned Jobs*	*Job Start Time*	*Job Finish Time*
	2	0	6000
	10	0	10800
	18	10800	14400
	19-5	14400	18720
	8	16320	18720
	14	16320	18720
	7	20520	21120
	11	20520	21120
5	12	20520	21120