A Two-level Job Scheduler for Large Distributed Systems

Bernabé Dorronsoro[1] and Sergio Nesmachnow[2]

[1] SnT, University of Luxembourg
bernabe.dorronsoro@uni.lu
[2] Universidad de la República, Uruguay
sergion@fing.edu.uy

1 Introduction

Datacenters are typically composed by a huge number of computational resources, including high performance clusters, large storage systems, and/or components of large grids or cloud systems [1]. This extended abstract presents two novel two-level algorithms for scheduling a large number of workloads in a heterogeneous distributed system composed of multi-core processors. In the higher-level (i.e., between distributed systems), our schedulers implement either the traditional Round Robin (RR) and Load Balancing (LB) techniques to distribute the workflows. In the lower-level (i.e., within each datacenter), we propose a novel scheduler technique, based on HEFT and using backfilling, to schedule the locally assigned workflow using the available local resources.

We propose in Section 2 a novel formulation of the problem, and we evaluate the two proposed schedulers according to three different metrics (described in Section 3), as the makespan, the energy consumption of the datacentres to execute all tasks, and the penalization due to violated deadlines. The schedulers used are explained in Section 4, and some interesting results we found in our experimentations are outlined in Section 5.

2 Problem Definition

Our problem is to schedule large workloads of n independent heterogeneous jobs $J = \{j_0, j_1, \ldots, j_n\}$ on a set of k heterogeneous computing nodes $CN = \{CN_0, CN_1, \ldots, CN_k\}$. We define:

- Each computing node CN_r is a collection of NP_r multicore processors (NP_r may be different for every CN_r). It is represented as a tuple $(ops_r, c_r, E^r_{IDLE}, E^r_{MAX}, NP_r)$ defining the performance of its composing processors in terms of the floating-point operations per second (FLOPS) they can process, their number of cores and energy consumption at idle and peak usage, as well as the number of processors composing it, respectively.
- Each job j_q is a parallel application that is decomposed into a set of tasks $T_q = \{t_{q0}, t_{q1}, \ldots t_{qm}\}$ with dependencies among them, typically having each task different computing requirements.
- Every job j_q has an associated deadline D_q before it must be accomplished.
- Each task $t_{q\alpha}$ is a duple $t_{q\alpha} = (o_{q\alpha}, nc_{q\alpha})$ containing its length $o_{q\alpha}$ (in terms of its number of operations), and the number of processors required to execute it in parallel, $nc_{q\alpha}$.

We represent every job as a *Directed Acyclic Graph* (DAG). It is a precedence task graph $j_q = (V, E)$, where V is a set of m nodes, each one corresponding to the task $t_{q\alpha}$ ($0 \leq \alpha \leq m$) of the parallel program j_q. E is the set of directed edges between the tasks that maintain a partial order among them. Let $\prec$ be the partial order of the tasks in G, the partial order $t_{q\alpha} \prec t_{q\beta}$ models the precedence constraints. That is, if there is an edge $e_{\alpha\beta} \in E$ then task $t_{q\beta}$ cannot start its execution before task $t_{q\alpha}$ completes. We consider negligible communication costs, as communications only happen between servers within the same CN.

3 Performance Metrics

We consider several performance metrics to evaluate the solution quality of each algorithm:

- Makespan, defined as the maximum completion time of the computing nodes.
- Energy consumption, defined as the energy required to compute all tasks. It is computed following the model by Nesmachnow et al. in [2].
- Cost due to penalizations. Every workflow has a deadline before it should be completed and a penalty function to apply when it is violated. If the deadline is respected, the cost for that application is 0, in other case it can be either $\sqrt{D_q - F_q}$, $D_q - F_q$, or $(D_q - F_q)^2$ (where F_q is the time when task q is finished), and it is specified by the problem instance for every workflow.

4 Schedulers

As mentioned, we propose two different classic methods for the high-level scheduler, and a novel method for the low-level. For the high-level, we use RR and LB techniques. On the one hand, RR iteratively assigns every job to the next computing node. If the job can not be executed in the selected computing node (because some task in it requires more cores than the number of cores of the servers in the computing node), then the heuristic continues the iteration to the next ones until a suitable computing node is found. On the other hand, LB adjusts the number of jobs assigned to every computing node. Jobs are ordered according to the number of cores they require. The number of cores a job requires is defined as the maximum number of cores demanded by any of its tasks. Then, the jobs with higher cores requirements are assigned first. They are allocated to the computing node with the lowest number of jobs assigned, among those than can execute it.

The proposed low-level scheduling heuristic is based on the Heterogeneous Earliest Finish Time (HEFT) strategy [3], but it uses a backfilling technique and adapts the logic to work with multicore computing resources, by taking into account the "holes" that appear when a specific computing resources is not fully used by a single task (i.e., because the task only requires a part of the available cores in the machine). It sorts the tasks according to the upward rank values, then gives priority to assign the tasks to existing holes rather than using empty machines in the CN. When a given task fits on more than one hole, the heuristic selects the hole that "best fit" the task (i.e. the hole that minimizes the difference between the hole duration and the time to compute the task), disregarding the finishing time of the task. When no hole is available to execute the task, the heuristic selects the machine with the minimum finishing time for the task. The machines (and also machine holes) within the CN are processed sequentially and ordered (from machine #0 to machine #M). The rationale behind this strategy is to use available holes and left unoccupied large holes and empty machines for upcoming tasks. Ties between holes as well as between machines are decided lexicographically, as the method searches sequentially (in order) both holes and machines.

5 Experimental Results

We analyzed the behavior of the two algorithms in 25 instances that are composed by $1,000$ workflows of different characteristics, as well as single tasks. The values used to characterize the processors are real ones taken from the current market. Table 1 presents the average performance difference (in %) for makespan (f_M) energy (f_E) and penalizations' cost (f_C) of every algorithm with respect to the best result for every instance. Sched1 is the method using RR in the higher level, while Sched2 uses LB technique. We can see that Sched1 clearly outperforms Sched2 in terms of makespan, energy, and cost, meaning that the RR technique distributes more effectively the workflows among the different clusters. In Fig. 1, we can see that Sched1 provides solutions of around 20% better makespan with 17% lower energy requirements for instance 1, which is a representative case.

Table 1: Average performance difference (in %) for makespan (f_M) energy (f_E) and penalizations' cost (f_C) of every algorithm with respect to the best result for every instance.

Algorithms	f_M	f_E	f_C
Sched1	**0.8**	**0.8**	**0.2**
Sched2	11.9	8.9	13.8

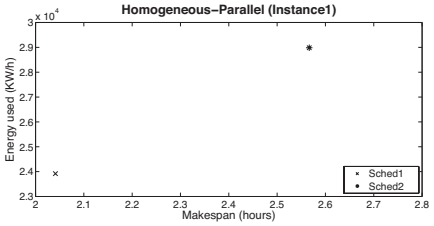

Fig. 1: Energy and makespan values provided by the algorithms on a representative case

References

1. A. Y. Zomaya, Y. C. Lee, Energy Efficient Distributed Computing Systems, Wiley-IEEE Computer Society Press, 2012.
2. S. Nesmachnow, B. Dorronsoro, J. E. Pecero, P. Bouvry, Energy-aware scheduling on multicore heterogeneous grid computing systems, Journal of Grid Computing 11 (4) (2013) 653–680.
3. H. Topcuouglu, S. Hariri, M.-y. Wu, Performance-effective and low-complexity task scheduling for heterogeneous computing, IEEE Transanctions on Parallel and Distributed Systems 13 (3) (2002) 260–274.