A Round-Robin Scheduling for Computer Clusters with Compatibility Constraints

Thomas Bonald*
*Télécom ParisTech
Université Paris-Saclay, France

Céline Comte†*
†Nokia Bell Labs, France

***Abstract*—We consider a computer cluster with some arbitrary bipartite graph of compatibilities between jobs and computers. Assuming jobs can be processed in parallel on several machines, we design a scheduling algorithm achieving balanced fair sharing of the computing resources, similar to round-robin scheduling in the presence of a single computer. We analyze the performance of this algorithm by introducing a new class of queueing system. In this model, the service interruptions and resumptions of the algorithm are reinterpreted in terms of random routing.**

A full version of this paper was submitted to Performance Evaluation Journal. It is available on arXiv [1].

***Index Terms*—Scheduling, parallel processing, order-independent queues, balanced fairness, insensitivity.**

I. Computer cluster

We consider a cluster of computers which can process jobs in parallel. Each computer has a fixed capacity expressed in floating-point operations per second. Each job consists of some random number of floating-point operations called the job size.

Compatibility constraints. When a job enters the cluster, it is assigned to a set of computers. This set may be chosen by the service provider (e.g. to randomly balance the load between machines) or imposed by technical constraints like data availability. We assume that the assignment is independent of the current state of the cluster, and in particular of the number of jobs at each computer.

The set of machines to which a job is assigned defines its class. The possible assignments are then represented by a bipartite *graph of compatibilites* between classes and computers, where there is an edge between a class and a computer if this computer can process the jobs of this class. A toy example is given in Fig. 1.

Parallel processing. Each computer processes its jobs sequentially in first-come first-served (FCFS) order. When a job is in service on several computers, these are pooled so that the service rate of the job is the sum of the capacities of the computers. The validity of this last assumption will be discussed in the next section.

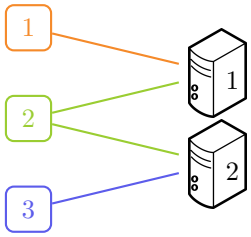

Fig. 1. Bipartite graph of compatibilities. The cluster contains two computers. There are three job classes, one for each possible assignment: class-1 and 3 jobs can only be served by computers 1 and 2, respectively, while class-2 jobs can be served by both computers.

II. Scheduling algorithm

A single computer. Round-robin scheduling consists in allocating equal-size time slices to the jobs in circular order. Equivalently, each job is interrupted once some fixed quantity of floating-point operations was processed.

The system state can be described as an ordered sequence of jobs. The computer processes the job which is at the head. When the service of a job is interrupted, this job is moved to the end of the queue while the computer starts processing the next job. An incoming job is appended to the end of the queue.

Multi-computer extension. Our extension consists in allocating the same quantity of work (expressed in floating-point operations for instance) to each job before service interruption, independently of the number and capacity of the computers that are processing this job.

We propose a distributed implementation where each computer is equipped with an exponentially distributed random timer. The expiration rate of the timer is proportional to the service capacity of the computer. When a job is in service on several machines, its service is interrupted whenever the timer of one of its machine

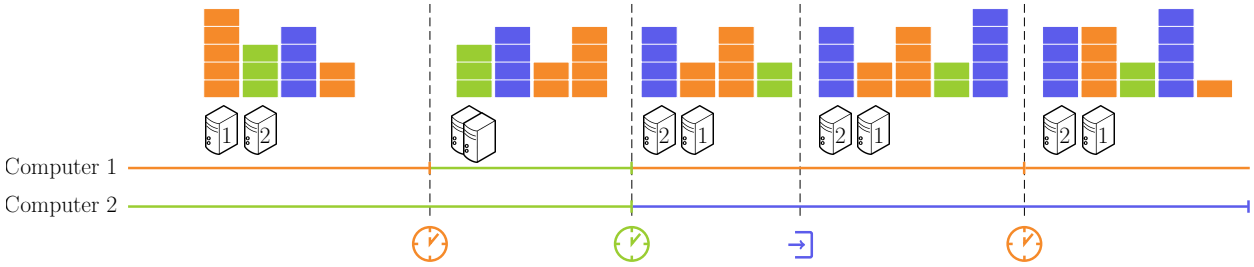

Fig. 2. Example of execution of the scheduling algorithm. The first transition is triggered by the expiration of the timer of computer 1. The first orange job, which was in service on this computer, is moved to the end of the queue. The second transition corresponds to the expiration of one of the two timers, after which the service of the green job is interrupted. Then a blue job arrives and is appended to the end of the queue. Finally, the timer of computer 1 expires and the first orange job is moved to the end of the queue.

expires. The time before interruption is thus exponentially distributed, with a rate which is proportional to the service rate of the job. Hence, the quantity of work given to a job before interruption is on average the same for all jobs. This mean quantity is denoted by θ. Fig. 2 gives an example for the configuration of Fig. 1.

The key parameter is the mean number $m = \frac{\sigma}{\theta}$ of interruptions per job, where σ is the mean job size. From the single-computer case, one can expect that the resource allocation is all the more *insensitive* when m increases, in the sense that the performance only depends on the traffic intensity of each class; however, taking m too large may induce a too high parallelization overhead.

III. Queueing model

Order-independent queue. The system is modeled by an order-independent queue [2] containing as many servers as there are computers in the cluster. Jobs of each class enter as an independent Poisson process. Each server processes its jobs sequentially in FCFS order and each job can be processed in parallel by several servers. Jobs have independent exponentially distributed unit sizes, which corresponds to the exponential timer in our algorithm. Upon service completion, a job may leave the queue or re-enter it with some fixed probability which depends on its class. This is the queueing interpretation of the interruptions and resumptions of the algorithm.

The queue state is naturally described by the ordered sequence of job classes. We show that its stationary distribution does not depend on the detailed traffic characteristics beyond the traffic intensity of each class.

Network of processor-sharing queues. Looking at the detailed queue state is not relevant if we only want to observe the performance of the system in the long run. We consider instead an aggregate state, only retaining the number of jobs of each class in the queue. The stationary distribution of this state proves to be that of a network of processor-sharing queues (with one queue for each class in the cluster), as illustrated in Fig. 3 for the system depicted in Fig. 1. We also show that the average service rates in this aggregate state are allocated according to *balanced fairness* [3], a resource allocation which is known for its insensitivity property. These results are supported by simulations.

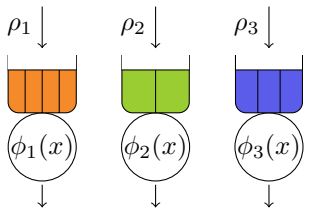

Fig. 3. Equivalent network of processor-sharing queues. Each queue corresponds to one class of jobs. The performance of this system only depends on the traffic intensity ρ_i of class i, for each $i = 1, 2, 3$.

IV. Conclusion

We have introduced a new algorithm, similar to round-robin scheduling, for computer clusters with compatibility constraints. We have assessed its performance with a new queueing model and showed that it achieves a balanced fair sharing of the computing resources. For the future works, we aim at gaining more insight on the impact of the mean number of interruptions per job on the sensitivity of the resource allocation.

References

[1] T. Bonald and C. Comte, "Balanced fair resource sharing in computer clusters," *CoRR*, vol. abs/1604.06763, 2017. [Online]. Available: https://arxiv.org/abs/1604.06763v2

[2] A. E. Krzesinski, "Order independent queues," in *Queueing Networks: A Fundamental Approach*, R. J. Boucherie and N. M. van Dijk, Eds. Boston, MA: Springer US, 2011, pp. 85–120.

[3] T. Bonald and A. Proutière, "Insensitive bandwidth sharing in data networks," *Queueing Syst.*, vol. 44, no. 1, pp. 69–100, 2003.