Online Scheduling of Splittable Tasks

Leah Epstein[1,⋆] and Rob van Stee[2,⋆⋆]

[1] School of Computer Science, The Interdisciplinary Center, Herzliya, Israel.
lea@idc.ac.il.
[2] Centre for Mathematics and Computer Science (CWI), Amsterdam, The Netherlands.
Rob.van.Stee@cwi.nl.

Abstract. We consider online scheduling of splittable tasks on parallel machines. In our model, each task can be split into a limited number of parts, that can then be scheduled independently. We consider both the case where the machines are identical and the case where some subset of the machines have a (fixed) higher speed than the others. We design a class of algorithms which allows us to give tight bounds for a large class of cases where tasks may be split into relatively many parts. For identical machines we also improve upon the natural greedy algorithm in other classes of cases.

In this paper, we consider the problem of distributing tasks on parallel machines, where tasks can be split in a limited amount of parts. A possible application of the splittable tasks problem exists in peer-to-peer networks [5]. In such networks large files are typically split and the parts are downloaded simultaneously from different locations, which improves the quality of service (QoS).

More generally, computer systems often distribute computation between several processors. This allows the distributed system to speed up the execution of tasks. Naively it should seem that the fastest way to run a process would be to let all processors participate in the execution of a single process. However in practice this is impossible. Set-up costs and communication delays limit the amount of parallelism possible. Moreover, some processes may have limited parallelism by nature. In many cases, the best that can be done is that a process may be decomposed into a limited number of pieces each of which must be run independently on a single machine.

The definition of the model is as follows. In the sequel, we call the tasks "jobs" as is done in the standard terminology. We consider online scheduling of splittable jobs on m parallel machines. A sequence of jobs is to be scheduled on a set of machines. Unlike the basic model which assumes that each job can be executed on one machine (chosen by the algorithm), for splittable jobs, the required processing time p_j of a job j may be split in an arbitrary way into (at most) a given number of parts ℓ. Those parts become independent and may run in parallel or at different times on different processors. After a decision (on the way a job is split) has been made, the scheduler is confronted by the basic scheduling problem, where each piece of job is to be assigned non-preemptively to one machine. In the on-line version, jobs are presented to the algorithm in a list, this

⋆ Research supported by Israel Science Foundation (grant no. 250/01).
⋆⋆ Research supported by the Netherlands Organization for Scientific Research (NWO), project number SION 612-30-002.

Dagstuhl Seminar Proceedings 05031
Algorithms for Optimization with Incomplete Information
http://drops.dagstuhl.de/opus/volltexte/2005/74

means that each job must be assigned before the next job is revealed. Only after the process of job splitting and assignment is completed, the next job is presented to the algorithm. The goal is to minimize the makespan which is the last completion time of any part of job.

We consider two machine models. The first one is the well known model of identical machines, where all machines have the same speed (w.l.o.g. speed 1). The second case relates to systems where several processors are faster (by some multiplicative factor) than the others. In this case let s be the speed of the fast processors. The other processors have speed 1. This also contains the model where one processor is fast and all others are identical [15, 8, 3, 14]. We call the machines of speed s *fast*, and all other machines are *regular* machines. The number of fast machines is denoted by f whereas the number of regular machines is $m - f$. The processing time of job j on a machine of speed s is p_j/s. Each machine can process only one job (or part of job) at a time, and therefore the completion time of the machine is the total processing time of all jobs assigned to it (normalized by the speed), which is also called the *load* of the machine. In the context of downloading files in a peer-to-peer network, the speeds correspond to the bandwidths for the different connections.

We use competitive analysis and given a problem we would like to determine its competitive ratio. The competitive ratio of an algorithm is the worst case ratio between the makespan of the schedule produced by the algorithm, and the makespan of an optimal offline algorithm which receives all input tasks as a set and not one by one. We denote the cost of this optimal offline algorithm by OPT. The competitive ratio of a problem is the best possible competitive ratio that can be achieved by a deterministic on-line algorithm.

Previous work

The basic model (with $\ell = s = 1$) was studied in a sequence of papers, each improving either the upper bound or the lower bound on the competitive ratio [10, 7, 2, 12, 1, 9, 6, 11]. The offline splittable jobs problem was studied by Shachnai and Tamir [19]. They showed that the problem is NP-hard (already for identical machines) and gave a PTAS for uniformly related machines. The problem was also studied by Krysta, Sanders and Vöcking [13] who gave an exact algorithm which has polynomial running time for any constant number of uniformly related machines. A different model that is related to our model is scheduling of parallel jobs. In this case, a job has several identical parts that must run simultaneously on a given number of processors [4, 16].

Our results

We first analyze a simple greedy-type algorithm that splits jobs into at most ℓ parts, while assigning them in a way that the resulting makespan is as small as possible. We then introduce a type of algorithm that always maintains a subset of $k < \ell$ machines with maximal load (while maintaining a given competitive ratio), and show that it is optimal as long as ℓ is sufficiently large in relation to $m + f$. The case $f = m - 1$ is treated separately. For smaller ℓ, we give an algorithm for identical machines that uniformly improves upon our greedy algorithm. Finally, we consider the special case

of four identical machines and $\ell = 2$, which is the smallest case for which we did not find an optimal solution. The algorithms assume that it is always possible to compute the value of OPT for a subsequence of jobs which already arrived. In the full paper, we explain how to compute this value.

References

1. Susanne Albers. Better bounds for online scheduling. *SIAM Journal on Computing*, 29:459–473, 1999.
2. Yair Bartal, Howard Karloff, and Yuval Rabani. A better lower bound for on-line scheduling. *Information Processing Letters*, 50:113–116, 1994.
3. Yookun Cho and Sartaj Sahni. Bounds for List Schedules on Uniform Processors. *SIAM Journal on Computing*, 9(1):91–103, 1980.
4. Anja Feldmann, Jiří Sgall, and Shang-Hua Teng. Dynamic scheduling on parallel machines. *Theoretical Computer Science*, 130:49–72, 1994.
5. Amos Fiat and Jared Saia. Censorship resistant Peer-to-Peer content addressable networks. In *Proceedings of the 13th Annual ACM-SIAM Symposium On Discrete Algorithms (SODA 2002)*, pages 94–103, 2002.
6. Rudolf Fleischer and Michaela Wahl. Online scheduling revisited. *Journal of Scheduling*, 3:343–353, 2000.
7. Gabor Galambos and Gerhard J. Woeginger. An on-line scheduling heuristic with better worst case ratio than Graham's list scheduling. *SIAM Journal on Computing*, 22:349–355, 1993.
8. Teofilo F. Gonzalez, Oscar H. Ibarra, and S. Sahni. Bounds for LPT Schedules on Uniform Processors. *SIAM Journal on Computing*, 6(1):155–166, 1977.
9. Todd Gormley, Nick Reingold, Eric Torng, and Jeffery Westbrook. Generating adversaries for request-answer games. In *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 564–565, 2000.
10. Ronald L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, 45:1563–1581, 1966.
11. J.F. Rudin III. *Improved bounds for the on-line scheduling problem*. PhD thesis, The University of Texas at Dallas, May 2001.
12. David R. Karger, Steven J. Phillips, and Eric Torng. A better algorithm for an ancient scheduling problem. *Journal of Algorithms*, 20:400–430, 1996.
13. Piotr Krysta, Peter Sanders, and Berthold Vöcking. Scheduling and traffic allocation for tasks with bounded splittability. In *Proc. of the 28th International Symposium on Mathematical Foundations of Computer Science (MFCS 2003)*, pages 500–510, 2003.
14. Rongheng Li and Lijie Shi. An on-line algorithm for some uniform processor scheduling. *SIAM Journal on Computing*, 27(2):414–422, 1998.
15. Jane W. S. Liu and C. L. Liu. Bounds on scheduling algorithms for heterogeneous computing systems. In Jack L. Rosenfeld, editor, *Proceedings of IFIP Congress 74*, volume 74 of *Information Processing*, pages 349–353, 1974.
16. Edwin Naroska and Uwe Schwiegelshohn. On an on-line scheduling problem for parallel jobs. *Information Processing Letters*, 81(6):297–304, 2002.
17. Jiří Sgall. A Lower Bound for Randomized On-Line Multiprocessor Scheduling. *Inf. Process. Lett.*, 63(1):51–55, 1997.
18. Jiří Sgall. *On-Line scheduling on parallel machines*. PhD thesis, Carnegie-Mellon University, Pittsburgh, PA, USA, 1994.
19. Hadas Shachnai and Tami Tamir. Multiprocessor scheduling with machine allotment and parallelism constraints. *Algorithmica*, 32(4):651–678, 2002.