Resource-aware allocation strategies for divisible loads on large-scale systems

Anne Benoit [2,4,5] Loris Marchal [1,4,5] Jean-François Pineau [6]
Yves Robert [2,4,5] Frédéric Vivien [3,4,5]

[1] CNRS [2] ENS Lyon [3] INRIA [4] Université de Lyon

[5] LIP laboratory, UMR 5668, ENS Lyon–CNRS–INRIA–UCBL, Lyon, France

[6] LIRMM laboratory, UMR 5506, CNRS–Université Montpellier 2, France

{anne.benoit,loris.marchal,jean-francois.pineau,yves.robert,frederic.vivien}@ens-lyon.fr

Abstract

In this paper, we deal with the large-scale divisible load problem studied in [12]. We show how to reduce this problem to a classical preemptive scheduling problem on a single machine, thereby establishing new complexity results, and providing new approximation algorithms and heuristics that subsume those presented in [12]. We also give some hints on how to extend the results to a more realistic framework where communication costs are taken into account.

1 Introduction

In this paper, we deal with the large-scale divisible load problem introduced by Viswanathan, Veeravalli, and Robertazzi [12]. In a nutshell, a grid system is deployed, with several divisible applications on one side and several computing servers on the other side. The problem is to determine when each application should be executed, and on which processors. Divisible load applications are quite interesting in such frameworks because (i) they encompass quite a variety of applications [11], and (ii) they are easily amenable to be distributed on several servers. As larger and larger collections of resources are made available to users, the problem of matching applications and servers becomes more and more difficult. Therefore the problem addressed in [12] is both very natural and of practical importance.

In [12], the problem is formulated in a quite general setting. However, most of the results presented by the authors only apply to a simplified scenario where communication costs are ignored. The latter scenario remains important because, in this context, the heterogeneous load-balancing problem is still difficult. In this paper we revisit this problem without communication costs. We show how it can be reduced to a well-studied scheduling problem on a uni-processor machine, that of scheduling preemptive tasks with release dates and deadlines (see Section 3). The main objective for the uni-processor problem is to maximize the number of tasks processed before their deadlines. Owing to this reduction, we establish new complexity results, and we provide new approximation algorithms and heuristics that subsume those presented in [12].

The general problem exposed in [12] seems much more difficult to solve when communication costs are considered. Nevertheless, we give hints and research directions in Section 4. Beforehand, we start by describing the target problem in Section 2.

2 Problem description

We only give an operational description of the key features of the problem considered in [12] and we refer the reader to the original article for motivation and additional details. For the convenience of the reader, we keep all notations of [12].

There are N sources (data servers) S_i that provide divisible loads to be processed by M sinks (computing processors) K_j. All sources are directly connected to all sinks via a complete bipartite network. The execution proceeds through iterations. Let $L_i^{(q)}$ be the load of source S_i at the beginning of iteration q, and let $L^{(q)} = \sum_{i=1}^{N} L_i^{(q)}$ be the total load in the system. A fraction $Y^{(q)}$ of $L^{(q)}$ will be processed during the iteration.

The main constraint to determine $Y^{(q)}$ is that each sink K_j must accommodate input data in a limited-size buffer. Let $B_j^{(q)}$ be the number of load units that K_j can store during iteration q (these quantities will vary dynamically across iterations). The idea of [12]

978-1-4244-3750-4/09/$25.00 ©2009 IEEE

is to saturate (at least one of) these M sink buffers. If $Y^{(q)}L^{(q)}$ units are to be distributed across all sinks, sink K_j should receive a share $\alpha_j Y^{(q)}L^{(q)}$ that is proportional to its computing power: $\alpha_j = \frac{\frac{1}{w_j}}{\sum_{x=1}^{M}\frac{1}{w_x}}$, where w_j is the inverse of the speed of K_j. Equivalently, K_j needs a time $w_j T_{cp}$ to execute one load unit, where T_{cp} is the computation intensity constant. Therefore $\alpha_j Y^{(q)}L^{(q)} \leq B_j^{(q)}$ for all j, hence the value $Y^{(q)} = \min_j \frac{B_j^{(q)}}{\alpha_j L^{(q)}}$. There remains to decide which source sends inputs to which sink. It is done proportionally again: S_i sends $\alpha_{i,j}^{(q)}$ units to K_j, where $\alpha_{i,j}^{(q)} = \frac{L_i^{(q)}}{L^{(q)}}\left(\alpha_j Y^{(q)}L^{(q)}\right) = \alpha_j Y^{(q)}L_i^{(q)}$.

The authors in [12] state that "*communication is [...] assumed to be negligible within a cluster node*". In fact, they consider all communications to be instantaneous. In particular, sources can send data to any number of sinks in parallel, while sinks can receive from any number of sources. Furthermore, all sinks can start computing at the very beginning of a period, because sinks can start computing as soon as they start receiving their first bit of data (hence a *fluid* model [3]). With the above *fluid* computation model and work distribution, all sinks start and end their computation simultaneously, as all computation times are equal: $\left(\alpha_j Y^{(q)}L^{(q)}\right) w_j T_{cp} = T^{(q)}$ for all j. Therefore buffers are needed only to prevent sinks to be overflowed with incoming data (sinks are computing at a lower rate than they are receiving). Anyway, the time to compute $Y^{(q)}L^{(q)}$ load units is $T^{(q)}$, and the time to compute the entire load is $\widehat{T} = \frac{T^{(q)}}{Y^{(q)}} = \frac{L^{(q)}T_{cp}}{\sum_{i=1}^{M}\frac{1}{w_i}}$, which shows that the platform is utilized at its full potential.

Each source S_i has a deadline T_{d_i} and, in an online setting, new sources can join the system at the beginning of each new period. The authors discuss several heuristic strategies to decide whether to accept new sources or not depending upon whether none, some, or all task deadlines are greater than $\widehat{T}$. As stated above, buffer sizes vary during the execution, which, according to the authors, calls for changing the duration of consecutive periods. In fact, because (i) the computation model is linear and (ii) overhead is paid neither for communication nor for control (e.g., for updating values from one period to the other, for taking scheduling decisions, etc.), periods could be arbitrarily small. From any solution in [12], by dividing each period into two same-size periods where each quantity is halved, one gets a solution requiring exactly half the buffer space. Therefore, buffers are not actually constraining. We can state this differently: we can slow down communications and send data from source S_i to sink K_j exactly at the rate this data is consumed (executed) by K_j, thereby eliminating the need for any temporary storage.

3 Equivalence to uni-processor with preemption

From the above description it should be clear that the problem reduces to scheduling N preemptive tasks $\mathcal{T}_i$ on a uni-processor machine, where each task $\mathcal{T}_i$ has a size L_i, a deadline T_{d_i}, and possibly a release date T_{r_i}, and the machine has speed $\sum_{i=1}^{M}\frac{1}{w_i}$. Indeed, since all sources can simultaneously send data to all sinks, without any type of communication (or buffer) constraints, a system with N sources is equivalent to one with a single source holding all the N loads. Similarly, since all sinks can simultaneously receive from all sources, a system with M sinks is equivalent to one with a single sink of cumulated speed. Thus, in the classical notation of scheduling theory, the studied problem is written $(1|\text{pmtn}; r_j|\sum U_j)$ [10]. Here 1 stands for 1 machine, *pmtn* means that task executions can be interrupted and later resumed, $r_j = T_{r_j}$ denotes the release dates and $\sum U_j$ is the number of late tasks (those having missed their deadlines).

The offline case, where everything is known in advance, can be solved in polynomial time. This was first shown in 1990 by Lawler [9]. Later, Baptiste [1] presented another algorithm, of lower complexity. The latter algorithm can be used as an absolute baseline reference to estimate (afterward) the performance of any online algorithm on any instance. For our problem, the competitive ratio of an online algorithm is the minimum, over all possible instances, of the ratio of the number of tasks that the algorithm succeeded to complete before their deadlines in that instance (the so called *early tasks*) and of the number of early tasks in the optimal scheduling for that instance. Baruah, Haritsa, and Sharma have shown [2] that no online algorithm has a competitive ratio larger than $\frac{1}{n}$, and thus that no online algorithm has constant competitive ratio (here, n is the optimal number of early tasks). Nevertheless, Kalyanasundaram and Pruhs have shown [8] that for any instance either their Lax algorithm or the well-known Shortest Remaining Processing Time (SRPT) is constant competitive on the instance. This leads them to design a constant-competitive randomized algorithm (but with a very very small constant). Furthermore, there are constant-competitive online algorithms for special instances, such as equal-size tasks, non-decreasing deadlines (with release times), or equal relative deadlines (deadline minus release date) [2]. Finally, if there exists a schedule satisfying all deadlines, then Earliest Deadline First (EDF) always finds it, even in an online setting [5].

In addition to establishing the complexity of the problem considered in [12], for both the offline and online settings, this short overview gives hints on how to better assess the quality of the three greedy online heuristics (EDF, Progressive, Non-interleaved) proposed in [12]. Indeed, these heuristics should be compared to their natural competitors: SRPT, Lax, and the randomized algorithm. In addition, a comparison to the optimal offline solution would give an absolute estimation of their performance.

4 General problem

The general problem addressed in [12] is challenging. An important question is to derive a good model, i.e., a model that is both realistic and tractable. The first issue is granularity. Multi-round (or multi-iteration) divisible load scheduling with a linear cost model faces the well-known problem that the optimal solution is to have infinitely many rounds of infinitely small size [6]. There are two ways to solve this problem: (i) introducing start-up costs for communications (and possibly for computations); (ii) introducing fixed-size tasks whose size corresponds to the atomicity of the application. The second issue is related to communication costs. It is not realistic to assume unlimited bandwidth capacity as links and network cards have a maximum capacity. We suggest two candidate models: (i) the *one-port* model [4] where two communications involving the same sender (or receiver) are serialized; (ii) the *bounded multi-port* model [7] where several communications involving the same sender (or receiver) can take place in parallel, provided that no device capacity is exceeded.

When communication costs are taken into account, buffer sizes truly matter when all applications do not have the same communication to computation ratio. Indeed, if a computationally intensive application is released before a communication intensive one, we must send as much data as possible for the first application before the second one is released, that is before the communication medium gets saturated by the second application. Once the second application is released, we saturate both communication and computation resources, thereby achieving an efficient platform utilization.

In [3] we studied the scheduling of several divisible loads arriving over time, in a system with one source, several sinks, and buffer sizes. We aimed at minimizing the maximum slowdown, or stretch. Minimizing the maximum stretch is a special case of scheduling with deadlines. For the offline version of the problem in [12], and with a finite number of changes of buffer sizes, the techniques presented in [3] will always find a schedule satisfying all deadlines when one exists. This is true in the bounded multi-port model with fluid computation and synchronous start, which is the bounded multi-port version of the problem in [12]. These techniques can be straightforwardly extended to platforms with multiple sources, and are expected to perform well in online settings, as was the case in [3]. However, they would need to be extended with some load admission policy for problems where all deadlines cannot be met.

5 Conclusion

The main goal of this paper was to revisit the work of [12] in the absence of communication costs, and to derive new complexity results and heuristics owing to a reduction to a classical scheduling problem.

The general problem with communication costs remains mostly open. We have only provided hints on how to tackle this challenging question. We plan to devote our future work on designing algorithms and heuristics for the general problem.

Acknowledgment

We thank the reviewers for their comments and suggestions. This work was supported in part by the ANR StochaGrid project.

References

[1] P. Baptiste. An $o(n^4)$ algorithm for preemptive scheduling of a single machine to minimize the number of late jobs. *Operations Research Letters*, 24(4):175–180, May 1999.

[2] S. Baruah, J. Haritsa, and N. Sharma. On-line scheduling to maximize task completions. In *Proc. IEEE Real-Time Systems Symposium*, pages 228–236. IEEE Computer Society Press, 1994.

[3] A. Benoit, L. Marchal, J.-F. Pineau, Y. Robert, and F. Vivien. Offline and online scheduling of concurrent bags-of-tasks on heterogeneous platforms. In *APDCM 2008*. IEEE Computer Society Press, 2008.

[4] P. Bhat, C. Raghavendra, and V. Prasanna. Efficient collective communication in distributed heterogeneous systems. *Journal of Parallel and Distributed Computing*, 63:251–263, 2003.

[5] M. L. Dertouzos. Control robotics: the procedural control of physical processes. In *Proceedings of IFIP Congress*, pages 897–813. IFIP, 1974.

[6] M. Gallet, Y. Robert, and F. Vivien. Comments on "design and performance evaluation of load distribution strategies for multiple loads on heterogeneous linear daisy chain networks". *J. Parallel and Distributed Computing*, 68(7):1021–1031, 2008.

[7] B. Hong and V. K. Prasanna. Adaptive allocation of independent tasks to maximize throughput. *IEEE Trans. Parallel Distributed Systems*, 18(10):1420–1435, 2007.

[8] B. Kalyanasundaram and K. R. Pruhs. Maximizing job completions online. *J. Algorithms*, 49(1):63–85, 2003.

[9] E. L. Lawler. A dynamic programming algorithm for preemptive scheduling of a single machine to minimize the number of late jobs. *Ann. Oper. Res.*, 26(1-4):125–133, 1990.

[10] J. Lenstra, R. Graham, E. Lawler, and A. Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, 5:287–326, 1979.

[11] T. Robertazzi. Ten reasons to use divisible load theory. *IEEE Computer*, 36(5):63–68, 2003.

[12] S. Viswanathan, B. Veeravalli, and T. G. Robertazzi. Resource-aware distributed scheduling strategies for large-scale computational cluster/grid systems. *IEEE Trans. Parallel Distributed Systems*, 18(10):1450–1461, 2007.

Biographies

Anne Benoit received the PhD degree from Polytechnical Institute of Grenoble (INPG) in 2003. She currently holds a position of Associate Professor at Ecole Normale Superieure in Lyon, France. She is the author of 10+ papers published in international journals, and 25+ papers published in international conferences. Her research interests include algorithms design and scheduling techniques for parallel and distributed platforms, and also the performance evaluation of parallel systems and applications. She is a member of the IEEE.

Loris Marchal received the PhD degree from École Normale Supérieure de Lyon in 2006. He is currently a CNRS researcher in the Computer Science Laboratory LIP at ENS Lyon. His research interest includes parallel algorithm design and scheduling for heterogeneous platforms.

Jean-François Pineau received the PhD degree from École Normale Supérieure de Lyon in 2008. He is currently a postdoctoral fellow with the Computer Science Laboratory LIRM at Montpellier. He is mainly interested in the design of parallel algorithms for heterogeneous platforms and in scheduling techniques.

Yves Robert received the PhD degree from Institut National Polytechnique de Grenoble in 1986. He is currently a full professor in the Computer Science Laboratory LIP at ENS Lyon. He is the author of five books, 100+ papers published in international journals, and 130+ papers published in international conferences. His main research interests are scheduling techniques and parallel algorithms for clusters and grids. Yves Robert served on many editorial boards, including IEEE TPDS. He was the program chair of HiPC'2006 in Bangalore and of IPDPS'2008 in Miami. He is a Fellow of the IEEE. He has been elected a Senior Member of Institut Universitaire de France in 2007.

Frédéric Vivien received a Ph.D. degree from École normale supérieure de Lyon in 1997. From 1998 to 2002, he was an associate professor at Louis Pasteur University, in Strasbourg, France. He spent the year 2000 working in the Computer Architecture Group of the MIT Laboratory for Computer Science. He is currently a full researcher from INRIA, working at the ENS Lyon. He is the author of one book, 25 papers published in international journals, and 35+ papers published in international conferences. His main research interests are scheduling techniques and parallel algorithms for clusters and grids.