Future Generation Computer Systems 29 (2013) 1140–1151

Contents lists available at SciVerse ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Scheduling linear chain streaming applications on heterogeneous systems with failures

Anne Benoit [a], Alexandru Dobrila [b], Jean-Marc Nicod [b], Laurent Philippe [b,*]

[a] ENS Lyon, Université de Lyon, LIP laboratory (ENS, CNRS, INRIA, UCBL), France

[b] FEMTO-ST Institute, UMR CNRS/UFC/ENSMM/UTBM, Besançon, France

ARTICLE INFO

Article history:
Received 1 September 2012
Received in revised form
13 December 2012
Accepted 21 December 2012
Available online 16 January 2013

Keywords:
Heterogeneous computing
Scheduling
Throughput maximization
Failure
Streaming applications
Complexity results
Linear programming

ABSTRACT

In this paper, we study the problem of optimizing the throughput of streaming applications for heterogeneous platforms subject to failures. Applications are linear graphs of tasks (pipelines), with a type associated to each task. The challenge is to map each task onto one machine of a target platform, each machine having to be specialized to process only one task type, given that every machine is able to process all the types before being specialized in order to avoid costly setups. The objective is to maximize the throughput, i.e., the rate at which jobs can be processed when accounting for failures. Each instance can thus be performed by any machine specialized in its type and the workload of the system can be shared among a set of specialized machines.

For identical machines, we prove that an optimal solution can be computed in polynomial time. However the problem becomes NP-hard when two machines may compute the same task type at different speeds. Several polynomial time heuristics are designed for the most realistic specialized settings. Simulation results assess their efficiency, showing that the best heuristics obtain a good throughput, much better than the throughput obtained with a random mapping. Moreover, the throughput is close to the optimal solution in the particular cases where the optimal throughput can be computed.

1. Introduction

In this paper, we address the issue of mapping a linear chain of tasks that processes a flow of jobs on heterogeneous resources subject to failures. Note that the work can be extended to the more general case when the throughput is not preserved all along the chain, i.e., the output throughput of the tasks of the pipeline may be smaller than their input throughput. This may arise for instance in streaming applications, either because the task operates some kind of selection on the input data, or when the task is not able to compute the output, for instance because of failures. So we rather consider the issue of task failures than machine failures.

A streaming application is composed of a flow of elementary jobs (job instances of the same size). Each of these elementary jobs is in turn composed of tasks, linked by precedence constraints. Thus, the platform must continuously execute instances of elementary jobs. The objective is to map the tasks onto a computational platform, consisting of several resources, in order to optimize the job flow through the platform. The goal is therefore to maximize the number of job output per time unit (the throughput), or equivalently, to minimize the time between two output jobs (the period). The problem is rather simple when the resources are homogeneous, but becomes more complex when considering heterogeneous platforms. The originality of our work is that we assume that the flow reduction may be linked to the tasks and/or to the processing resources.

The paper is organized as follows. We first define more precisely the context and we give an overview of related work in Section 2. Then we present the framework, define the failure model and formalize the optimization problems in Section 3. An exhaustive study on the complexity of these problems is provided in Section 4: we exhibit some particular polynomial problem instances, we prove that the remaining problem instances are NP-hard, and we propose some linear programming formulations to solve sub-problems. In Section 5, we design a set of polynomial-time heuristics to solve the most general problem instance. In Section 6, we conduct extensive simulations to assess the relative and absolute performance of the heuristics. Finally, we conclude in Section 7.

2. Context and related work

In the past years, much attention has been paid to workflow applications on the grid and several workflow management systems facilitate their execution on the computing resources [1,2].

* Corresponding author. Tel.: +33 3 81 66 66 54; fax: +33 3 81 66 64 50.
E-mail addresses: Anne.Benoit@ens-lyon.fr (A. Benoit), alex.dobrilla@gmail.com (A. Dobrila), Jean-Marc.Nicod@femto-st.fr (J.-M. Nicod), Laurent.philippe@femto-st.fr (L. Philippe).

0167-739X/$ – see front matter © 2013 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2012.12.015

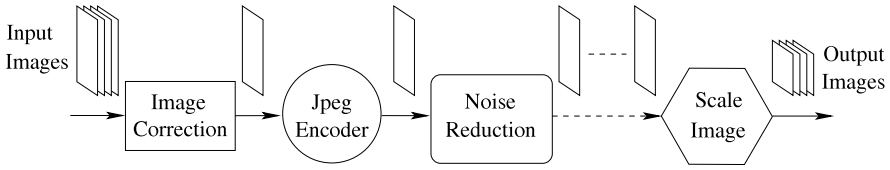

Fig. 1. Application example.

We focus in this paper on linear chain streaming applications, executed on computing grids. In this case, the resources are distributed, heterogeneous, and may not be reliable enough to assume that the failure rate can be ignored in the mapping strategy.

From the application point of view, we deal with *coarse-grain* streaming applications. Applications are a linear graph of tasks (pipeline) with a type associated to each task. In these streaming applications, a series of data enters the pipeline and progresses from task to task until the final result is computed. Examples of such applications are stream-processing applications composed of processing elements as in [3], or pipelined query operators with precedence constraints as in [4], or application based on stream programming as in [5]. An illustrating example of a streaming application is an image processing application as presented in [6]. A stream of intra-vascular ultrasound images are captured by a transducer at a specific rate and sent to the resources to be processed before being displayed (see Fig. 1). A similar application is the Synthetic Aperture Radar (SAR) [7], which creates 2D or 3D images from radar signals gathered by a moving sensor. These images are then used to make decisions. It is important to note that for such streaming applications, it is more relevant to optimize the throughput rather than the total finish time.

The considered resources are typically dedicated execution resources grouped in a distributed platform, a grid, on which we process a batch of input data. Each resource of the platform provides functions or services that are able to handle a task type. As each task is typed, it can only be processed on a resource that implements its task type. In the case of computing resources this model can be illustrated by Software as a Service (SaaS) based platforms [8]. The considered platforms are heterogeneous as the resources are usually not uniform and thus the tasks are processed with different speeds.

Our aim is to efficiently map and schedule the applications onto the resources. We target coarse-grain applications and platforms such that the cost of communications is negligible in comparison to the cost of computations. In the illustrating example, processing an image is indeed much more costly than transferring an image. This is a complex problem (known in the literature as *multi-processor tasks* [9,10]) as the considered resources are heterogeneous. The mapping defines which resource performs which task. So processing a streaming application on the platform amounts to enter jobs on the platform and to progress from resource to resource, following the task chain, until the final result is computed. After an initialization delay, a new job is completed every period and it exits the pipeline. The period is therefore defined as the longest cycle-time of a resource, and it is the inverse of the throughput that can be achieved. The goal is to minimize the period, which is equivalent to maximizing the throughput, i.e., the number of final results that exit the system per time unit. This approach is different from [4,11] where pipelined query operators with precedence constraints are ordered to optimize a bottleneck metric, the slowest stage in the pipeline and the selectivity of the operators. In our case, the operation order is fixed and we target the operator mapping on the resources.

Note that optimizing the schedule of a set of tasks on a heterogeneous platform is complex (NP-hard most of the time), as we will show later in the paper. Given that the optimization target in the throughput optimization and that the tasks of a streaming application remain identical all along the execution, it is worth to take the time to compute a static assignment before execution since the execution parameters, and in particular the execution time of the tasks, do not change during the streaming execution.

Considering platforms such as grids, or clouds, implies to take failure possibilities into account. Failures cannot be ignored when applications last for a long time. The failure rate is too high to assume that no fault will impact the execution. In the grid context, failures may occur because of the nodes, but they also may be related to the complexity of the service [12]. So we consider in our problem that the type of a task affects its computation requirements and its failure rate. This failure rate may also depend on the resource itself, platform heterogeneity also assumes reliability heterogeneity.

Replication is often used to deal with failures in distributed systems [13,14]. To ensure that a result is output, the same execution is replicated among several processors. However, a common property of our target platforms is that we cannot use replication to overcome the faults. For streaming applications, it is too costly to replicate each task (maybe several times if we want a high warranty) and replication shrinks the throughput. Fortunately, losing a few jobs may not be a big deal; for instance, the loss of some images in the illustrating example will not alter the result, as far as the throughput is maintained. This failure model is based on the Window-Constrained [15] model, often used in real-time environment. In this model, only a fraction of the messages will reach their destination: for y messages, only x ($x \leq y$) of them will reach their destination. The y value is called the Window. The losses are not considered as a failure but as a guarantee: for a given network, a Window-Constrained scheduling [16,17] can guarantee that no more than x messages will be lost for every y sent messages.

Other research work already focuses on streaming applications. It operates on data sets but most of the time on homogeneous resources [18]. Other studies as [19,20] target workflow application scheduling on grids but more from a practical point of view. A comprehensive survey of pipelined workflow scheduling is given in [21] but it does not tackle the fault tolerance issue. In [22] a computation is considered to be faulty in case where data is lost. Replication is used to improve the reliability of the system and to optimize two objective functions, the latency and the reliability. This is different from the case tackled in this paper as no throughput change along the pipeline is considered.

In this paper, we therefore solely concentrate on the problem of period minimization (i.e., throughput maximization), where extra jobs are processed to account for failures. For instance, if there is a single task, mapped on a single machine, with a failure rate of $1/2$, a throughput of x jobs per unit time will be achieved if the task processes $2 \times x$ jobs per time unit.

3. Framework and optimization problems

In this section, we define the problems that we tackle. First we present the application, platform and failure models. Then we discuss the objective function and the rules of the game before formally introducing the optimization problems.