Sustainable Computing: Informatics and Systems 4 (2014) 44–51

Contents lists available at ScienceDirect

Sustainable Computing: Informatics and Systems

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

Optimal energy consumption and throughput for workflow applications on distributed architectures

Abdallah Ben Othman, Jean-Marc Nicod*, Laurent Philippe, Veronika Rehn-Sonigo

FEMTO-ST Institute, CNRS/UFC/ENSMM/UTBM, Besançon, France

ARTICLE INFO

Article history:
Received 31 August 2012
Received in revised form
20 December 2013
Accepted 14 January 2014

Keywords:
Scheduling
Workflow applications
Energy minimization
Fault tolerance
Throughput maximization
Polynomial complexity

ABSTRACT

In this article we study both the throughput and the energy optimization problems for a distributed system subject to failures that executes a workflow at different speed levels. The application is modeled as a directed acyclic graph composed of typed tasks linked by dependency constraints. A continuous flow, or a great number of application instances, has to be processed. Optimizing the collaborative system performance implies to increase the throughput – the number of application instances processed by time unit – or to decrease the period – the time needed to output one instance of the system. The system is designed as a collaborative platform of distributed machines. Each machine collaborates with others by performing all the instances of at least one task of the DAG. The problem we tackle is to optimize the configuration of the platform. In this paper we propose two polynomial algorithms that optimize the two objectives of period (i.e., throughput) minimization and energy minimization. We prove that the proposed algorithms give optimal results. Our optimization approach is hierarchic in that we either minimize the energy consumption for the optimal period or minimize the period for the optimal energy consumption. Moreover a minor modification of our algorithms allows to compute the Pareto front between the two optimal solutions.

1. Introduction

In this paper we focus on workflow applications described as directed acyclic graphs (DAGs). An application is mapped on a set of distributed machines and a flow of instances has to be processed. This is the case of systems that continuously input raw data to which several processing stages or tasks must be applied to obtain a final result [1]. Another example are configurable production systems [2,3]. The system is designed as a collaborative platform of distributed machines. Each machine collaborates with others by performing all the instances of at least one task of the DAG. Illustrations of these contexts are a flow of images generated by a camera that must be processed in several stages or a production flow with several succeeding tasks. The considered tasks are of different types that represent the different processing procedures (e.g., filters, analysis, assembly and so on). When the data processing in the application is substantial, several computers or production cells must be used to be able to process the entire input flow and the problem of scheduling the tasks on the resources becomes complex due to the heterogeneity of the processing times on the resources [4]. The complexity of the problem may be lowered by considering that each machine only executes one task type thus avoiding costly context changes and cases where a machine executes parts of several tasks [5]. Then the initial problem becomes a mapping problem where task types must be mapped onto machines and the objective function is to find the best possible throughput, i.e., to maximize the number of instances processed per time unit [6]. Note that the objective function used in this paper is the period minimization – the time between two consecutive outputs. The period is the inverse of the throughput, which amounts to the same but is more widely used in workflow system optimization.

This paper addresses the problem of using a dedicated system that continuously executes the same DAG of tasks onto different instances with transient failures that sometimes destroy one instance. In this context the objective is to provide the lowest period for the system output. Application flow execution as image processing may be very energy consuming for large images. In the workflow all execution steps however do not have the same execution cost. Lowering the execution speed of the less loaded resources can substantially reduce the global energy consumption. We thus propose two polynomial algorithms based on a greedy approach that either reach the objective of period minimization for a fixed energy consumption or reach the objective of energy minimization for a fixed period. Furthermore we prove the optimality of both the two approaches. For cases where one of the constraints is not compliant with the optimal solution, we propose an adaptation of our algorithms that provides the Pareto front which links the two

* Corresponding author. Tel.: +33 381 40 28 11.
E-mail address: Jean-Marc.Nicod@femto-st.fr (J.-M. Nicod).

2210-5379/$ – see front matter © 2014 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.suscom.2014.01.001

optimal solutions – the lowest energy consumption with an optimal period and the lowest period with the optimal energy consumption.

The paper is organized as follows: Section 2 discusses related work. In Section 3 we give a formal definition of the problem. In Section 4 we present and prove several lemmas that are used in Section 5 to define the proposed algorithms. Then, in Section 6, we present the result of the algorithm implementation, in the shape of a Pareto front, for cases where the constraints do not allow to reach the optimal solutions. We conclude the article in Section 7.

2. Related work

Nowadays more and more attention is paid to energy consumption for financial and environmental reasons. This tendency has also reached the distributed computing domain [7–9]. In the case of flow applications where the global throughput is directed by the lower throughput of the graph, it is not always necessary that all machines run at maximum speed [10]. Several papers define an energy model based on power consumption modes where the processing capabilities depend on the supplied voltage [11,12]. Then voltage scaling is used to slow down some of the machines – and as a consequence energy spared – without affecting the global throughput [13]. It is thus worth to find the lowest possible speed for each machine for a given throughput or, on the opposite, the best reachable throughput for a given energy consumption.

On the other hand in distributed environments such as GRIDs or micro-factories, the risk of task failures cannot be ignored, in particular for long running and communication intensive applications like flow applications. The failures may append for numerous reasons such as network or computing errors, network contention, task complexity and so on. Numerous works on reliability and energy focus on the problem of Dynamic Voltage and Frequency Scaling (DVFS) which leads to more errors when the frequency is scaled down [14]. This assumption however only lays on the assumption that with low voltage the processor becomes more likely to be pone to errors. Defining a global error model for an entire distributed system however is not so simple as these systems are composed of so many elements, each with their own failure model. For instance, [15] uses a model where the reliability of the processor is directly related to the number and span of speed changes. Increasing the speed of computations or/and the processing load can also lead to less reliable systems as it is shown in real HPC in-production systems [16], or to DRAM errors [17]. [18], analyzing a large database of failure in PC systems, suggests that *“one can minimize the likelihood of failure over a given time period by operating at the slowest CPU speed sufficient to achieve desired performance”*. Based on this observation, we assume in this paper that optimizing the energy consumption of the system by decreasing its speed leads to decrease the fault rate in addition to period minimization. In this context we propose two algorithms that minimize either the energy consumption for an optimal period or find the lowest period for a minimal energy consumption. Note that the antagonism between reliability and machine speed induces the complexity of the problem and, in particular, the properties set in Section 4.

3. Framework

In this section we formally define the application, platform and energy models and our optimization objective.

3.1. Application model

We consider a workflow application that is running during infinite or long time. The application is modeled as a directed acyclic graph (DAG) $G(T, D)$, with $T = \{T_1, \ldots, T_n\}$ the tasks of the application

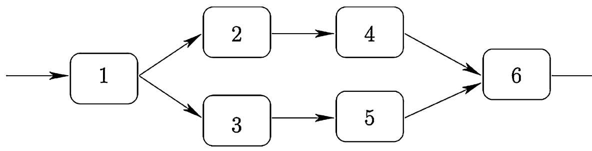


Fig. 1. Illustrating task graph.

and $D \subset T \times T$ the dependencies between the tasks (see Fig. 1). Data sets enter the graph at the source task and traverse the graph from one task to another before producing a final result at the sink task. A weight w_i is associated to each task T_i that corresponds to the amount of work to be done to perform the task.

3.2. Platform and execution model

The platform is modeled as a set $M = \{M_1, \ldots, M_p\}$ of p machines fully interconnected. Each machine has input and output communication buffers to store temporary data. We assume that the communication times are shorter than the computation times so that, thanks to the data buffering, the former are covered by computations and thus can be neglected.

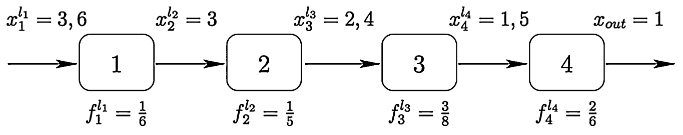
The tasks are statically allocated to machines according to an allocation function a such that $a(i) = u$, i.e., all data instances that enter task T_i are performed by machine M_u. Note that in this work we assume that the mapping is already defined thanks to mapping algorithms as defined in [6] and we concentrate on period and energy optimization of a given mapping.

A machine M_u runs at different speed levels l_u ($l_u \in \{0, 1, 2, \ldots, \max(l_u)\}$) with an associated slow down factor $\alpha_u^{l_u} \in [1, +\infty)$. Note that M_u runs at its highest speed, noted s_u, for level $l_u = 0$. The system configuration L is given by vector $L = (l_1, l_2, \ldots, l_u, \ldots, l_p)$ that describes the speed level of each machine.

Tasks are subject to transient failures. In case of failure, the current data is lost and the task starts to process the next data. The failure rate is defined for each task as the percentage of failures. For a task T_i, allocated to machine M_u, we assume that the failure rate $f_i^{l_u}$ depends on the task and on the machine speed level l_u. We also assume that the failure rate increases with the machine speed: $f_i^{l_u} < f_i^{l'_u}$ with $l_u > l'_u$. It comes that if machine M_u performs $x_i^{l_u}$ input data sets with task T_i, it outputs $(1 - f_i^{l_u})x_i^{l_u}$ data sets due to failures. Considering L, the configuration of the platform, it is possible to compute $x_i^{l_u}$ backwards for each data output of the application. If task T_i has only one outgoing edge (T_i, T_j) within the DAG, $x_i^{l_u} = x_j^{l_u}/(1 - f_i^{l_u})$ (see Fig. 2 for an example). If the task T_i has several outgoing edges (T_i, T_j) within the DAG, $x_i^{l_u} = \sum_{(T_i,T_j) \in D} x_j^{l_u}/(1 - f_i^{l_u})$. Thus $x_i^{l_u}$ is the average number of data sets that machine M_u has to perform with task T_i so as to output at least one result data set out of the system.

3.3. Example of the platform and execution model

To clarify the above stated platform and execution model, we consider the application given in Fig. 2. To keep the example simple, we suppose that task T_i is mapped onto machine M_u with $i = u$ and that each machine runs at is lowest speed level. Hence task T_1 is mapped onto machine M_1 which is running at a level l_1. The failure



$x_1^{l_1} = 3,6$
$x_2^{l_2} = 3$
$x_3^{l_3} = 2,4$
$x_4^{l_4} = 1,5$
$x_{out} = 1$
$f_1^{l_1} = \frac{1}{6}$
$f_2^{l_2} = \frac{1}{5}$
$f_3^{l_3} = \frac{3}{8}$
$f_4^{l_4} = \frac{2}{6}$

Fig. 2. Example for the backward computation of the necessary amount of data sets for each task in a linear application, taking into account the failure rates.