Optimal Energy Consumption and Throughput for Workflow Applications on Distributed Architectures

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

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

[Abdallah.BenOthman, Jean-Marc.Nicod, Laurent.Philippe, Veronika.Sonigo]@femto-st.fr

March 10, 2014

Abstract

In this article we study both the throughput and the energy optimization problem 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.

Keywords

Scheduling; workflow applications; energy minimization; fault tolerance; throughput maximization; polynomial complexity

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 adresses 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 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 payed to energy consumption for financial and environmental reasons. This tendency has also reached the distributed computing domain [7, 8, 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 on 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 and $D \subset T \times T$ the dependencies between the tasks (see Figure 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.

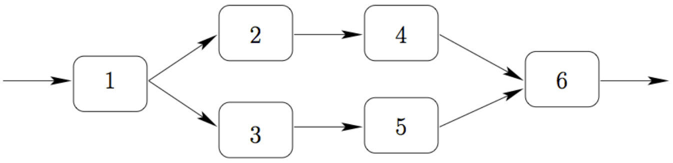

Figure 1: Illustrating task graph

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, $x_i^{l_u} = \frac{x_j^{l_u}}{1 - f_i^{l_u}}$ (see Figure 2 for an example). If the task T_i has several outgoing edges (T_i, T_j) within the DAG, $x_i^{l_u} = \sum_{(T_i, T_j) \in D} \frac{x_j^{l_u}}{1 - f_i^{l_u}}$. 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 Figure 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 rate of task T_1 accordingly depends on l_1 and we have $f_1^{l_1} = 1/6$. The same holds for the other tasks.

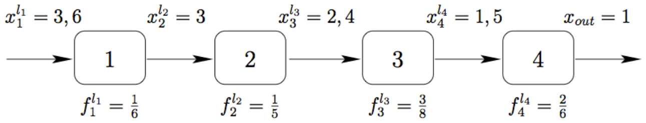
$x_1^{l_1} = 3,6$
$x_2^{l_2} = 3$
$x_3^{l_3} = 2,4$
$x_4^{l_4} = 1,5$
$x_{out} = 1$
1
2
3
4
$f_1^{l_1} = \frac{1}{6}$
$f_2^{l_2} = \frac{1}{5}$
$f_3^{l_3} = \frac{3}{8}$
$f_4^{l_4} = \frac{2}{6}$

Figure 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.

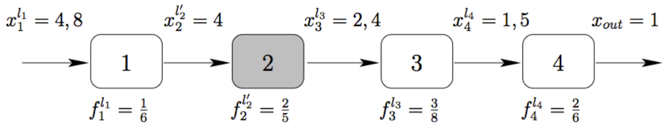
$x_1^{l_1} = 4,8$
$x_2^{l'_2} = 4$
$x_3^{l_3} = 2,4$
$x_4^{l_4} = 1,5$
$x_{out} = 1$
1
2
3
4
$f_1^{l_1} = \frac{1}{6}$
$f_2^{l'_2} = \frac{2}{5}$
$f_3^{l_3} = \frac{3}{8}$
$f_4^{l_4} = \frac{2}{6}$

Figure 3: The acceleration of machine M_2 implies a higher number of input data sets at the application entry.

We suppose to have the failure rates indicated in Figure 2. We can now compute the necessary amount of data sets that each task needs as input to be able to produce at least one result out of the system ($x_{out} = 1$). As indicated earlier, the computation is done backwards and we get $x_4^{l_4} = \frac{1}{1-f_4^{l_4}} x_{out} = 1.5$.

We now suppose that machine M_2 has two possible speed levels l_2 and l'_2, where l'_2 is the accelerated level ($l'_2 < l_2$). The associated failure rate for level l'_2 is $f_2^{l'_2} = 2/5$. If machine M_2 switches to level l'_2, the $x_i^{l_u}$ values have to be recomputed in consequence and you can see the new configuration in Figure 3.

3.4 Throughput/Period Model

We define the platform throughput as the number of data outputs per time unit. We define the period of the platform as the inverse of the throughput: the period defines the maximum duration between the output of two consecutive data outputs. As we already know both the number of tasks that have to be performed to output at least one data set and the mapping of tasks to machines, we can compute the period of each machine of the platform. The task period $p_i^{l_u}$ is the time to perform $x_i^{l_u}$ instances of each task T_i mapped onto machine M_u: $p_i^{l_u} = x_i^{l_u} \times \frac{w_i \times \alpha_u^{l_u}}{s_u}$. Then the machine period $p_u^{l_u}$ on M_u is:

$$p_u^{l_u} = \sum_{T_i | a(i)=u} p_i^{l_u} = \sum_{T_i | a(i)=u} x_i^{l_u} \times \frac{w_i \times \alpha_u^{l_u}}{s_u}$$

The application period on the platform is the longest period over all machines in configuration L:

$$P(L) = \max_{M_u \in M} (p_u^{l_u})$$

We define the critical machine M_c as the machine with the longest period that determines the application period, i.e., $P(L) = p_c^{l_c}$. We denote $M_c(L)$ one critical machine of the configuration L.

3.5 Energy Model

The energy consumption $E(L)$ of the platform in configuration L is the sum of the energy consumption $E^L(u)$ of each machine M_u that performs at least one task of the graph. The energy $E^L(u) = E^L_{stat}(u) + E^L_{dyn}(u)$ is the sum of $E^L_{stat}(u)$, the static part of energy consumed when machine M_u is in service, and $E^L_{dyn}(u)$, the dynamic part of energy consumed when the machine performs its tasks [9].

$E^L_{stat}(u)$ only depends on the duration of the platform usage. The static energy needed to output one data out of the system is thus $E^L_{stat}(u) = \varepsilon_u \times P(L)$ where ε_u is the static energy consumption per time unit, $P(L)$ is the period of the application (or the duration between two consecutive outputs) and L is the configuration of the system.

On the other hand, the dynamic part of the energy depends on the machine speed when it performs tasks. Several models are defined in the literature for the dynamic energy consumption of a machine. It is usually represented by a polynomial of at least second degree as in [19, 20] or a more general strictly convex and increasing function g of the machine speed as in [11] and [15]. As in [20] we assume here that the energy consumed on machine M_u at speed s depends on $C_{en} \times s^{\beta_u}$ where $\beta_u > 1$ and $C_{en} > 0$. As the speed of machine M_u is $\frac{s_u}{\alpha_u^{l_u}}$ the dynamic energy consumed during one period by machine M_u is the sum of the energy consumed to perform all the tasks needed for that period is given by equation 1.

$$E^L_{dyn}(u) = \sum_{T_i|u=a(i)} \left(C_{en} \times \left(\frac{s_u}{\alpha_u^{l_u}} \right)^{\beta_u} \times p_i^{l_u} \right) \tag{1}$$

3.6 Optimization Objectives

In this paper, we are interested in two objectives. First, we aim at minimizing the period and minimizing the energy consumption for the optimal period. Second, we optimize the energy consumption of our platform while minimizing the period for the optimal energy consumption.

4 System Properties

In this section we state some important properties for the changing of the system configuration L and we first exhibit the relation between two system configurations L and L'. The period of a machine M_u in system configuration L' can be expressed through the task periods in system configuration L. Based on the platform model, we know that a machine period is the sum of all its task periods, and we can deduce the following relation:

$$p_u^{l'_u} = \sum_{T_i|a(i)=u} x_i^{l'_u} \times \frac{w_i \alpha_u^{l'_u}}{s_u} = \sum_{T_i|a(i)=u} \frac{x_i^{l'_u} \alpha_u^{l'_u}}{x_i^{l_u} \alpha_u^{l_u}} \times p_i^{l_u} \tag{2}$$

We now consider the influence of slowing down or accelerating machines.

Lemma 1. *When a group of machines is accelerated, the amount of work to output one data set increases.*

Proof. Let L and L' be two system configurations with $l'_u \leqslant l_u$ for each machine $M_u \in M$, i.e., L' has some accelerated machines in comparison to configuration L. We aim at proving that $\forall T_i \in T, x_i^{l_u} \leqslant x_i^{l'_u}$ with $a(i) = u$. That means the amount of input data sets for task T_i is more important in configuration L'.

Let $M_u \in M$ be an accelerated machine whose configuration is set to l'_u. As $l'_u < l_u$, by definition of our model, $f_i^{l_u} < f_i^{l'_u}$ for all tasks $T_i \in T$ with $a(i) = u$ and so $\dfrac{1}{1 - f_i^{l_u}} < \dfrac{1}{1 - f_i^{l'_u}}$. From the definition of the computation of $x_i^{l_u}$, the previous expression implies that the value of $x_i^{l'_u}$ increases on M_u. Moreover since these values are computed backwards (Cf. Figure 2 in Section 3.3), this incrementation recursively modifies the $x_j^{l_v}$ by backwards following the dependency constraints in the group of machines. The global workload of each machine M_v, where $a(j) = v$, thus increases. □

Lemma 2. *When a group of machines is accelerated, it cannot decrease the period of the other machines.*

Proof. Let L and L' be two system configurations with $l'_v \leqslant l_v, \forall M_v \in M$, i.e., L' has some accelerated machines compared to L. Let M_u be a machine with $l_u = l'_u$. We aim at proving that $p_u^{l_u} \leqslant p_u^{l'_u}$.

First, with Lemma 1, we know that for all tasks the amount of work is more important in L' than in L: $x_i^{l_u} \leqslant x_i^{l'_u}, \forall t_i \in T$. Next, we know that if $l_u = l'_u$, the acceleration coefficient α is also the same: $\alpha_u^{l_u} = \alpha_u^{l'_u}$. Hence we have:

$$\forall t_i \in T \;\; s.t. \;\; u = a(i) : \alpha_u^{l_u} x_i^{l_u} \leqslant \alpha_u^{l'_u} x_i^{l'u}$$

and we get:

$$\forall t_i \in T \;\; s.t. \;\; u = a(i) : p_i^{l_u} \leqslant \frac{\alpha_u^{l'_u} x_i^{l_u}}{\alpha_u^{l_u} x_i^{l_u}} \times p_i^{l_u}$$

This holds true for all task periods and we can deduce the machine period by summing up over all task periods of a machine M_u and with Eq. 2, we prove that the machine period of M_u is smaller in configuration L than in L':

$$p_u^{l_u} = \sum_{T_i \in T | a(i) = u} p_i^{l_u} \leqslant \sum_{T_i \in T | a(i) = u} \frac{\alpha_u^{l'_u} x_i^{l'_u}}{\alpha_u^{l_u} x_i^{l_u}} \times p_i^{l_u} = p_u^{l'_u}$$

□

Corollary 1. *The acceleration of another machine than the critical machine cannot decrease the application period.*

Proof. Let L and L' be two system configurations with $l'_v \leqslant l_v, \forall M_v \in M$, i.e., L' has some accelerated machines. Let M_c be the critical machine such that $p_c^{l_c} = P(L)$ and $l_c = l'_c$. With Lemma 2, we know that the machine period of machine M_c is lower (or equal) in L than in L': $p_c^{l_c} \leqslant p_c^{l'_c}$. By definition, we deduce that the period of configuration L is lower than the maximum period in L':

$$P(L) \leqslant p_c^{l'_c} = \max_{M_u \in M} (p_u^{l'_u}) = P(L')$$

□

Lemma 3. *The acceleration of a machine cannot decrease the dynamic energy of any machine.*

Proof. Let L and L' be two system configurations with $l'_u \leqslant l_u, \forall M_u \in M$, i.e., L' has some accelerated machines. We aim at proving that for all machines the dynamic energy is smaller in L than in L': $\forall M_u \in M, E^L_{dyn}(u) \leqslant E^{L'}_{dyn}(u)$.

By definition we have $1 \leqslant \alpha_u^{l'_u} \leqslant \alpha_u^{l_u}$ and $\beta_u > 1$ so $\forall T_i \in T$ we can write:

$$\frac{1}{(\alpha_u^{l_u})^{\beta_u-1}} \leqslant \frac{1}{(\alpha_u^{l'_u})^{\beta_u-1}}$$

By using the three positive constants C_{en}, w_i and s_u and by using Lemma 1, $0 < x_i^{l_u} \leqslant x_i^{l'_u}$, $\forall T_i \in T$ with $a(i) = u$, we obtain:

$$C_{en}\left(\frac{s_u}{\alpha_u^{l_u}}\right)^{\beta_u} x_i^{l_u}\frac{w_i\alpha_u^{l_u}}{s_u} \leqslant C_{en}\left(\frac{s_u}{\alpha_u^{l'_u}}\right)^{\beta_u} x_i^{l'_u}\frac{w_i\alpha_u^{l'_u}}{s_u}$$

$$C_{en}\left(\frac{s_u}{\alpha_u^{l_u}}\right)^{\beta_u} p_i^{l_u} \leqslant C_{en}\left(\frac{s_u}{\alpha_u^{l'_u}}\right)^{\beta_u} p_i^{l'_u}$$

$$\sum_{T_i|a(i)=u} C_{en}\left(\frac{s_u}{\alpha_u^{l_u}}\right)^{\beta_u} p_i^{l_u} \leqslant \sum_{T_i|a(i)=u} C_{en}\left(\frac{s_u}{\alpha_u^{l'_u}}\right)^{\beta_u} p_i^{l'_u}$$

With Equation 1 we have $E^L_{dyn}(u) \leqslant E^{L'}_{dyn}(u)$. □

Lemma 4. *If the application period does not decrease, machine acceleration always increases the energy consumption of the application.*

Proof. Let L and L' be two system configurations with $l'_v \leqslant l_v, \forall M_v \in M$, i.e., L' has some accelerated machines. Let M_u be a machine with $l'_u < l_u$. Thus $P(L) \leqslant P(L')$ and the following inequality on the static energy of the two system configurations holds: $\forall M_u \in M : \varepsilon_u \times P(L) \leqslant \varepsilon_u \times P(L')$. Moreover with Lemma 3 we know that for all machines the sum of the task energy consumption is smaller in L than in L':

$$\forall M_u \in M : E^L_{dyn}(u) \leqslant E^{L'}_{dyn}(u)$$

Thus we deduce that the total energy consumption of a machine M_u behaves the same way as both the static and the dynamic part hold the inequality:

$$\forall M_u \in M : \varepsilon_u \times P(L) + E^L_{dyn}(u) \leqslant \varepsilon_u \times P(L') + E^{L'}_{dyn}(u)$$

$$E^L(u) \leqslant E^{L'}(u)$$

Hence we conclude that $E(L) \leqslant E(L')$. □

5 Algorithms

In this section, we present two algorithms. The first algorithm *OptPer*(L) finds a system configuration with the optimal period and the minimal energy-consumption for this period. The second one, *OptEner*(L), finds a system configuration with the optimal energy-consumption and the minimal period for this consumption.

5.1 Algorithm OptPer

The algorithm $OptPer(L)$ (see Algorithm 1) returns the optimal system configuration resulting from a given system configuration, i.e., the system configuration with the optimal period and the minimal energy consumption for the optimal period. The algorithm starts from an initial configuration L where each machine is set at its maximal slow down level. Then, at each step, the algorithm speeds up the machine $M_c(L)$ by reducing its level l_c by one. The new configuration is noted $\widehat{L}$. If the new system configuration period $P(\widehat{L})$ is better than the current best period, it is stored in L as the new best system configuration. Then a new critical machine $M_c(\widehat{L})$ is identified and the algorithm passes to the next step. Otherwise or if the slow down level of $M_c(\widehat{L})$ is null ($\widehat{l_c} = 0$) the algorithm finishes. The number of steps needed to finish this algorithm takes polynomial time. Indeed, in the worst case, the algorithm iterates $p \times LMAX$ times, with $LMAX = \max_u(\max(l_u))$ a constant which does not depend on the problem size. At each step, the computation of the necessary amount of the data sets for all of the tasks takes $O(n)$ operations and the complexity of algorithm 1 is $O(p \times n)$.

Algorithm 1: $OptPer(L)$

```
$M_c \leftarrow$ critical machine of $L$
$\widehat{L} \leftarrow L$
$\widehat{l_c} \leftarrow \widehat{l_c} - 1$
$\widehat{M_c} \leftarrow$ critical machine of $\widehat{L}$
while $(\widehat{l_c} > 0)$ do
    if $\big(P(\widehat{L}) < P(L)\big)$ then
        $L \leftarrow \widehat{L}$
        $M_c \leftarrow \widehat{M_c}$
    $\widehat{l_c} \leftarrow \widehat{l_c} - 1$
    $\widehat{M_c} \leftarrow$ critical machine of $\widehat{L}$
return L
```

To prove the optimality of the period returned by this algorithm we first set Lemma 5 and its corollary 2. For that we first define $A(L)$ as the set of system configurations resulting from all possible machine accelerations from the system configuration L. Let L and L' be two system configurations.

$$\forall L, L' : L' \in A(L) \Leftrightarrow \forall u : l'_u \leqslant l_u \tag{3}$$

For example, if $L = (2, 1)$:

$$A(L) = \{(2,1), (1,1), (0,1), (2,0), (1,0), (0,0)\}$$

We recall that $M_c(L)$ is one of the critical machines of the configuration L, i.e., one of the slowest machines of the configuration L.

Lemma 5. *Let us consider a configuration L and a subset of system configurations L' in $A(L)$ where $l'_c = l_c$ with M_c a critical machine of the configuration L. Then the period $P(L)$ is smaller than any period $P(L')$:*

$$\forall L' \in A(L) \mid l_c = l'_c \Rightarrow P(L) \leqslant P(L') \tag{4}$$

Proof. From the definition of function $A(L)$ in equation 3 we know that:

$$\forall L' \in A(L) \Rightarrow l'_u \leqslant l_u$$

And from Corollary 1 we know that if the critical machine is not accelerated, the period cannot decrease. By association we get that for all system configurations L' in $A(L)$ with $l'_c = l_c$ the period of configuration L' is higher or equal than the period of the configuration L. □

As a consequence of Lemma 5, only the configurations in $A(L)$ that increase the speed level of a critical machine $M_c(\widehat{L})$ can provide a better period for the system. We formalize this property in the following corollary:

Corollary 2. *The only configuration that is able to decrease the period of the system from a configuration L is to accelerate a critical machine.*

Note that speeding up the critical machine by one level does not always lead to a better period for two reasons. First because speeding up a critical machine does not always lead to improve its own period. This acceleration is however an imposed condition to improve the application period in some cases. Second because there may be several critical machines at the same time, i.e., machines that have the same period, and we must speed up all of them before improving the application period.

Theorem 1. *$OptPer(L)$ finds the optimal system configuration L^* with the optimal period in $A(L)$, i.e.:*

$$L^* = OptPer(L),\ \forall L' \in A(L) \Rightarrow P(L^*) \leqslant P(L').$$

Proof. First we note that, based on the definition of the system period $P(L)$ given in Section 3, the order in which the machines are accelerated to reach a configuration L' from a configuration L does not impact $P(L')$. The period $P(L')$ only depends on the configuration L', so on the $(l'_1, l'_2, \dots, l'_p)$ values.

Now we consider a sequence of configurations $S = \langle L_1, L_2, \dots, L_k \rangle$ with $k = |A(L)|$ such that $L_1 = L$ is the initial configuration of the algorithm where all the machines are set to their lowest speed level and $L_2, \dots, L_k \in A(L_1)$. The optimal configuration L^* can be defined as:

$$L^* = \underset{L_f \in S}{\operatorname{argmin}}(P(L_f))$$

Let L_a and L_b two system configurations such that the configuration L_a is obtained by speeding up one machine from the configuration L_b. From Lemma 5 we know that having $P(L_a) < P(L_b)$ implies that the critical machine $M_c(L_b)$ has necessarily been speeded up to obtain L_a.

As the order in which each machine is accelerated to reach a given configuration L_f from the initial configuration L does not impact the period value $P(L_f)$, we can reorder the sequence S in a new sequence S'. S' is reorganized such that $L_1 = L$ is the first configuration of the sequence and then each configuration L_x is obtained from configuration L_{x-1} by accelerating one of its critical machines $M_c(L_{x-1})$ is placed just after L_x. All other configurations are placed after. This reordering of the sequence does not change the optimal value L^*.

Let L_a be the last configuration of the sequence S' that is obtained by accelerating a critical machine. Then:

$$\forall L_b \in S' \text{ s.t. } b > a \Rightarrow P(L_a) \leqslant P(L_b)$$

and

$$L_a = \underset{L_f \in \{L_a, \ldots, L_k\}}{\text{argmin}} (P(L_f)) \tag{5}$$

Indeed only non critical machines are accelerated after step a and from Lemma 2 we know that this will not decrease the system period.

Here $\langle L_1, \ldots, L_a \rangle$ is the sequence obtained by $OptPer(L)$ step by step. Thanks to the condition on line 6 in Algorithm 1, $OptPer(L)$ takes the best configuration from this sequence. As a consequence:

$$OptPer(L) = \underset{L_f \in \{L_1, \ldots, L_a\}}{\text{argmin}} (P(L_f)) \tag{6}$$

Then, from Equations 5 and 6, we deduce:

$$OptPer(L) = \underset{L_f \in \{L_1, \ldots, L_k\}}{\text{argmin}} (P(L_f)) = L^*$$

□

Additionally to the optimal period algorithm $OptPer(L)$ also finds the configuration that is the less energy consuming.

Theorem 2. *$OptPer(L)$ finds the system configuration with the minimal energy-consumption E^* for the optimal period.*

Proof. First we can remark that the static energy consumption of the machines only depends on the system configuration period $P(L)$ and thus is the same as long as the system keeps the same period. This is in particular true for the optimal period L^* so that the proof can be limited to the study of the optimality of dynamic energy consumption.

Then, as for the period computation, we can note that the energy consumption definition does not depend on the order in which the configurations are used to reach a target configuration. We can arrange the configuration sequence in any order.

We consider again the sequence S' of configurations where the configurations are ordered in such a way that we accelerate critical machines first until L^* and then we put the other configurations after. As defined within the proof of the previous theorem, let L_a be the last configuration of the sequence S' that is obtained by accelerating a critical machine. We have $S' = \langle L_1, \ldots, L_a, \ldots, L_k \rangle$ with $k = |A(L)|$ and $L = L_1$ the initial configuration where each machine configuration is set at its maximal slowdown level. Now each configuration in the sub-sequence $\langle L_{a+1}, \ldots, L_k \rangle$ is a sub-optimal configuration so it is not considered in the following as we are only concerned by configurations which are potentially optimal. In $\langle L_1, \ldots, L^* \rangle$ we also have sub-optimal configurations that are not considered either. As a result we just have to look at configurations L'' in $\langle L^*, \ldots, L_a \rangle$ whose period is minimal $(P(L^*) = P(L''))$.

By definition L^* is the first configuration that reaches an optimal period in S' so that an other configuration L'' in S' with an optimal period is such that

$\forall u : l''_u \leqslant l^*_u$. By using Lemma 3 we deduce that the energy consumed by each configuration L'' with $P(L^*) = P(L'')$ is at least as high as the consumption $E^* = E(L^*) \leqslant E(L'')$ and then that $E(OptPer(L)) = E(L^*)$ is optimal. □

5.2 Algorithm OptEner

From the previous algorithm, it is possible to define another greedy algorithm, $OptEner(L)$ (see Algorithm 2), based on the same approach that finds a configuration L^* with an optimal (lowest) energy consumption and with a minimal period for this energy consumption. Note that the energy consumption, as defined in the framework model, is composed of a static part and a dynamic part. In some cases where the period is too large, the speed of the machines is thus so low that the static part of the energy consumption becomes predominant. It is then possible to increase the speed of the machine while decreasing the energy consumption. On the other hand, if we increase too much the speed of the machines, above the optimal value, the energy starts increasing. Then the $OptEner(L)$ algorithm finds a system configuration whose energy consumption is optimal. As the speed of the machines is increased accordingly, $OptEner(L)$ is also a configuration with the minimal associated period. Note that this algorithm also works to compute the minimal energy consumption for a given period. We assume that the complexity of Algorithm 2 is $O(p \times n)$ considering the same arguments used to compute the complexity of Algorithm 1.

Algorithm 2: $OptEner(L)$

```
$M_c \leftarrow$ critical machine
$\widehat{L} \leftarrow L$
$\widehat{l}_c \leftarrow \widehat{l}_c - 1$
$\widehat{M_c} \leftarrow$ critical machine
while ($\widehat{l}_c > 0$) do
    if $\Big(\big(E(\widehat{L}) < E(L)\big) \vee$
        $\big((E(\widehat{L}) = E(L)) \wedge (P(\widehat{L}) \leqslant P(L))\big)\Big)$ then
        $L \leftarrow \widehat{L}$
        $M_c \leftarrow \widehat{M_c}$
    $\widehat{l}_c \leftarrow \widehat{l}_c - 1$
    $\widehat{M_c} \leftarrow$ critical machine of $\widehat{L}$
return $L$
```

The algorithm starts from L, the initial configuration where each machine is set at its lowest speed level. Step by step, the algorithm looks for configurations where the energy is decreased compared to the current configuration or if the energy is not decreased at least the period is. The algorithm iterates until the critical machine cannot be accelerated anymore, i.e., when the critical machine has reached its highest speed level.

To prove the optimality of the algorithm we prove first that $OptEner(L)$ finds the system configuration with the optimal energy consumption and then we prove that $OptEner(L)$ finds the minimal period. Before these proofs, we set Lemma 6:

Lemma 6. *Let L' be a configuration in $A(L)$ and M_c be a critical machine of configuration L such that the slowdown level for M_c is the same in L as in L', then the energy consumption of L is lower than the energy consumption of L':*

$$\forall L' \in A(L) : l'_c = l_c \Rightarrow E(L) \leqslant E(L')$$

Proof. We know, from the definition of function $A(L)$, that each machine slowdown level l_u of a machine M_u in L is higher or equal than in L' with $L' \in A(L)$. And, considering Corollary 1, we know that the application period with the system configuration L is lower or equal than any other system configuration L' in $A(L)$ where $l'_c = l_c$ ($M_c = M_c(L)$). Thus, by association, we show:

$$\forall L' \in A(L) : M_c = M_c(L) \text{ and } l'_c = l_c \Rightarrow P(L) \leqslant P(L')$$

Moreover, from Lemma 4, if the slowest machine is not accelerated then the energy consumption of the system cannot decrease:

$$\begin{aligned} \forall L, L' :\ & (\forall u \text{ s.t. } 1 \leqslant u \leqslant p,\ l'_u \leqslant l_u \wedge P(L) \leqslant P(L')) \\ & \Rightarrow E(L) \leqslant E(L') \end{aligned}$$

Thus we find that the energy consumption in L is lower than in every system configuration L' in $A(L)$ when $l'_c = l_c$. □

Theorem 3. *$OptEner(L)$ finds the system configuration L^* with the optimal energy consumption $E^* = E(L^*)$ in $A(L)$:*

$$L^* = OptEner(L),\ \forall L' \in A(L) \Rightarrow E(L^*) \leqslant E(L')$$

Proof. Let L be a system configuration and let M_c be a critical machine of configuration L. By using Lemma 6 we assess that the only way to lower the energy consumption is to accelerate the critical machine.

The remainder of the proof is very similar to the proof of the $OptPer(L)$ optimality. First we state that the order in which the machines are accelerated to reach a configuration L' from a configuration L does not impact the energy value. Indeed the energy consumed by a machine M_u only depends on energy constants (C_{en} and β_u), on its speed level ($\alpha_u^{l_u}$) and its period ($p_i^{l_u}$).

Then we consider the sequence of configurations $S = \langle L_1, L_2, \ldots, L_k \rangle$ such that $k = |A(L)|$ and $L = L_1$ is the initial configuration where each machine configuration is set at its maximal slowdown level. We can thus assume that $L_2, \ldots, L_k \in A(L_1)$. We sort these configurations in a new sequence S' where critical machines are accelerated first. S' is reorganized as mentioned before, i.e., $L_1 = L$ is the first configuration of the sequence and then each configuration L_x is obtained from configuration L_{x-1} by accelerating one of its critical machines. All other configurations are placed after. $\langle L_1, \ldots, L_a \rangle$ is the sequence obtained by $OptEner(L)$ step by step. As L_a is the last configuration where a critical machine can be accelerated, we have $l_c = 0$ in L_a. Obviously all configurations that are ordered after L_a are configurations where the critical machines have not been accelerated and these machines cannot have a better period. From Lemma 4 we deduce that these configurations are more energy consuming than configuration L_a:

$$\forall f,\ a < f \leqslant k : E(L_a) \leqslant E(L_f)$$

We recall that the configuration sequence $\langle L_1, \ldots, L_a \rangle$ is explored by the *OptEner* algorithm. Thanks to the condition on line 6 the algorithm only records the best of the covered configurations and thus finds the configuration with the lowest optimal energy consumption. □

Theorem 4. *OptEner finds the system configuration with the minimal period for the optimal energy consumption.*

Proof. We have already shown, in the proof of Theorem 1, that the order in which we consider the machine accelerations does not impact the period value of a given configuration. We also know, from Lemma 3, that a machine acceleration can only decrease the energy consumption of the application if its period is lowered. Let E^* be the optimal energy consumption.

As for Theorem 2, we consider the sequence S' where the configurations that accelerate critical machines come first in the same order as explored by the algorithm (until $l_c = 0$). The other configurations are placed at the end of the sequence. The shape of the sequence is:

$$S' = \langle L_1, \ldots, L^*, \ldots, L_a, \ldots, L_k \rangle$$

From the previous proof (Theorem 3) we know that the optimal energy consumption is reached in this sequence at first for L^* and that configurations after L_a do not have an optimal energy consumption. Then we just have to consider the cases between L^* and L_a. These cases are examined by the algorithm *OptEner* step by step. The algorithm does not finish when it finds the optimal energy consumption with L^* but continues until it reaches the maximum speed level for a critical machine for the first time. This case occurs with L_a. Thanks to the condition on line 6 of Algorithm 2 we guarantee that, in this interval, if a period improvement is possible then it returns this best configuration. □

6 Sub-optimal cases tied by constraints

In some cases the period or the energy may be constrained by external factors, as a limited output throughput or as limited available energy in embedded systems for example. In these cases, the optimal values computed by our algorithms are not reachable and a multi-criteria computation must be done to find either the best corresponding energy or period values. In this case the proposed algorithms can be easily adapted by introducing a new condition in the "*if condition*" of the algorithms. This condition allows to change the L configuration only if the new configuration $\widehat{L}$ does not violate the constraint. It is then interesting to explore different constraints to get the pareto front that gives the best solutions for each configuration between the two optimal solutions.

For illustration we have computed the pareto front for a specific configuration given in Table 1. The platform configuration (see Table 1a) is composed of five machines M_u ($1 \leqslant u \leqslant 5$) with the same initial speed s_u (in computing units per second) and 10 speed levels l_u so that the slowdown factor $\alpha_u^{l_u}$ ranges between 1 and 2. The value of β_u is either set to 2 or 3. The considered application has five tasks. The associated weight w_i (in computing units) and failure rate $f_i^{l_u}$ are given in Table 1b. The failure rate ranges between 0.01% and 10% and depends on l_u.

Machine M_u	s_u	l_u	$\alpha_u^{l_u}$	β_u
0	3	[1,9]	[1,2]	3
1	3	[1,9]	[1,2]	2
2	3	[1,9]	[1,2]	3
3	3	[1,9]	[1,2]	3
4	3	[1,9]	[1,2]	2

(a) Platform configuration

Task T_i	w_i	f_i
0	69	[0.13%,9.24%]
1	31	[0.4%,9.26%]
2	19	[0.51%,7.23%]
3	80	[0.29%,8.91%]
4	54	[0.54%,9.85%]

(b) Application

Table 1: Illustrated configuration parameters

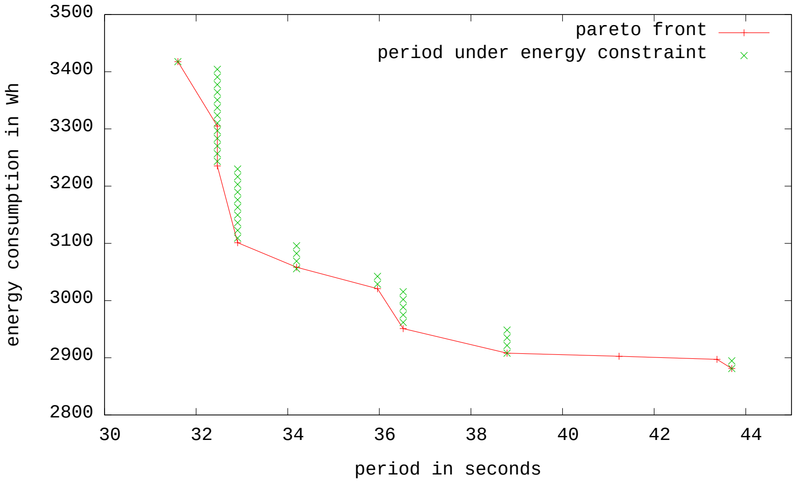

Figure 4: Example of a Pareto front for sub-optimal platform configurations

In Figure 4 we plot the period obtained under energy constraints. About forty values of energy are taken between the optimal energy value (2881.26 Wh) and the energy corresponding to the lowest – optimal – period (3417.62 Wh). Note that the computed results are so close that they shape thick lines. Each line corresponds to the optimal configuration for the critical machine considering a range of energy values. The solid line gives the Pareto front obtained for this platform and this application. The Pareto front contains only 11 solutions. Each of them corresponds to a configuration given by our algorithm to guarantee the lowest period for a given energy level that has not to be exceeded to perform one output out of the system (see Table 2). Note that two Pareto solutions – (32.4637, 3304.99) and (32.4642, 3235.26) – are very close but the corresponding lines do not overlap as Figure 4 gives the impression. Figure 5 gives the details of this area.

7 Conclusion

In this paper we tackle the problem of energy saving in the case of DAG shaped workflow applications executed on distributed unreliable platforms. We assume that the energy consumption and the fault rate decreases when the machines are slowed down. Given an initial mapping of the tasks onto the machines

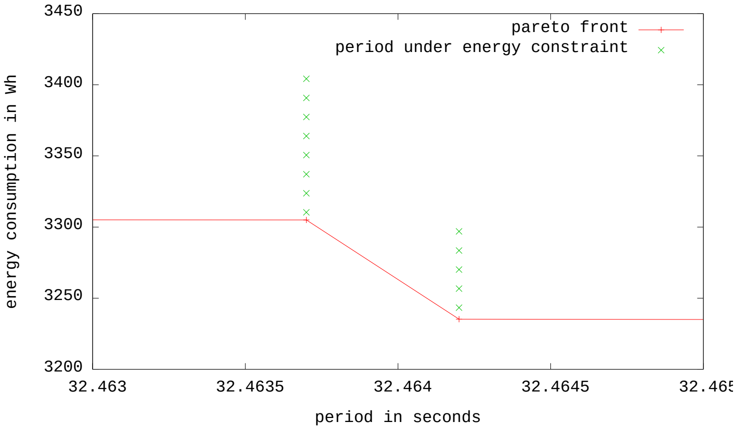

Figure 5: Zoom in the Pareto front for periods in the interval of 32.463 s and 32.465 s

period (s)	31.6122	32.4637	32.4642	32.9058	34.1930	35.9612	36.5210	38.7897	41.2347	43.3722	43.6949
energy (Wh)	3417.62	3304.99	3235.26	3101.17	3058.26	3020.72	2950.99	2908.08	2902.72	2897.35	2881.26

Table 2: List of the 11 points belonging to the Pareto front presented in Figure 4

we propose two algorithms and prove their optimality. The first algorithm minimizes the application period and finds the lowest global energy consumption for that period. The second algorithm optimizes the energy consumption of the system and finds the lowest period for that energy consumption.

In practical systems the energy consumption and the execution period are however often opposed criteria and reducing one of them usually leads to increase the other one. For this reason there is a need to find configurations that balance these criteria. For that we have modified the algorithms and introduced constraints values for both the energy and the period.

In the current work the failure rate of the tasks is a consequence of the speed of the machines. As lowering the task period leads to more faults there is a need to improve the reliability of the system. Improving the system quality implies to find a trade-off between energy, speed and faults. As future work we will consider the problem of optimizing the three criteria of energy, failures and throughput to identify the best configurations of the system given different objectives of improving the reliability, maximizing the speed and lowering the energy consumption. We plan to use multi-criteria techniques and explore different fault models to find good configurations. Due to the complexity of the tri-criteria problem we plan to develop heuristics that give efficient results for each of the three objectives in practical cases.

References

[1] J. Subhlok, G. Vondran, Optimal mapping of sequences of data parallel tasks, SIGPLAN Not. 30 (1995) 134–143.

[2] M. Tanaka, Development of desktop machining microfactory, Journal RIKEN Rev 34 (2001) 46–49.

[3] E. Descourvières, S. Debricon, D. Gendreau, P. Lutz, L. Philippe, F. Bouquet, Towards automatic control for microfactories, in: Proceedings of IAIA'2007, 2007.

[4] V. Rehn-Sonigo, Multi-criteria Mapping and Scheduling of Workflow Applications onto Heterogeneous Platforms, Phd thesis, ENS LYON (Jul. 2009).

[5] S. Diakité, J.-M. Nicod, L. Philippe, L. Toch, Assessing new approaches to schedule a batch of identical intree-shaped workflows on a heterogeneous platform, IJPEDS 27 (1) (2012) 79–107.

[6] A. Benoit, A. Dobrila, J.-M. Nicod, L. Philippe, Mapping workflow applications with types on heterogeneous specialized platforms, Parallel Computing, Special Issue ISPDC'09 37 (8) (2011) 410–427.

[7] G. Da Costa, J.-P. Gelas, Y. Georgiou, L. Lefevre, A.-C. Orgerie, J.-M. Pierson, O. Richard, K. Sharma, The green-net framework: Energy efficiency in large scale distributed systems, in: IPDPS'09, IEEE, 2009, pp. 1–8.

[8] T. Niemi, J. Kommeri, K. Happonen, J. Klem, A.-P. Hameri, Improving energy-efficiency of grid computing clusters, in: GPC'09, Springer-Verlag, 2009, pp. 110–118.

[9] A. Benoit, P. Renaud-Goud, Y. Robert, Performance and energy optimization of concurrent pipelined applications, in: IPDPS'10, IEEE, Atlanta, USA, 2010, pp. 1–12.

[10] A. Benoit, P. Renaud-Goud, Y. Robert, R. Melhem, Energy-aware mappings of series-parallel workflows onto chip multiprocessors, in: ICPP'11, IEEE, 2011, pp. 472–481.

[11] H. Aydin, Q. Yang, Energy-aware partitioning for multiprocessor real-time systems, in: Proceedings of the 17th International Symposium on Parallel and Distributed Processing, IPDPS '03, IEEE Computer Society, Washington, DC, USA, 2003, pp. 113 – 121.

[12] J.-J. Chen, Multiprocessor energy-efficient scheduling for real-time tasks with different power characteristics, in: ICPP'05, IEEE, Oslo, Norway, 2005, pp. 13–20.

[13] A. Benoit, P. Renaud-Goud, Y. Robert, Power-aware replica placement and update strategies in tree networks, in: IPDPS'11, IEEE, Anchorage, USA, 2011, pp. 2–13.

[14] V. Degalahal, L. Li, V. Narayanan, M. Kandemir, M. J. Irwin, Soft errors issues in low-power caches, IEEE Trans. on VLSI 13 (2005) 1157–1166.

[15] Z. Zhang, F. Li, H. Aydin, Optimal speed scaling algorithms under speed change constraints, in: High Performance Computing and Communications (HPCC), 2011 IEEE 13th International Conference on, 2011, pp. 202–210. doi:10.1109/HPCC.2011.35.

[16] B. Schroeder, G. Gibson, A large-scale study of failures in high-performance computing systems, Dependable and Secure Computing, IEEE Transactions on 7 (4) (2010) 337–350. doi:10.1109/TDSC.2009.4.

[17] B. Schroeder, E. Pinheiro, W.-D. Weber, Dram errors in the wild: a large-scale field study, in: Proceedings of the eleventh international joint conference on Measurement and modeling of computer systems, SIGMETRICS '09, ACM, New York, NY, USA, 2009, pp. 193–204. doi:10.1145/1555349.1555372.

[18] E. B. Nightingale, J. R. Douceur, V. Orgovan, Cycles, cells and platters: an empirical analysisof hardware failures on a million consumer pcs, in: Proceedings of the sixth conference on Computer systems, EuroSys '11, ACM, New York, NY, USA, 2011, pp. 343–356. doi:10.1145/1966445.1966477.

[19] T. Ishihara, H. Yasuura, Voltage scheduling problem for dynamically variable voltage processors, in: Symposium on Low Power Electronics and Design, IEEE, 1998, pp. 197–202.

[20] N. Bansal, T. Kimbrel, K. Pruhs, Speed scaling to manage energy and temperature, J. ACM 54 (1) (2007) 3:1–3:39. doi:10.1145/1206035.1206038.