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

***Abstract*—In this paper 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 which 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 article we propose two polynomial algorithms that optimize the two objectives of period (i.e., throughput) and energy minimization and we prove that the proposed algorithms give optimal results. Our optimization approach is hierarchic in the sens that we either minimize the energy consumption for an optimal period or minimize the period for the optimal energy consumption.**

***Keywords*-Scheduling, workflow applications, energy minimization, fault tolerance, throughput, polynomial complexity**

I. 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] or configurable production systems [2], [3]. Illustrations of these contexts are a flow of images generated by cameras 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 whole 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 period minimization, the inverse of the throughput, which amounts to the same but is more widely used in workflow system optimization.

In this article we tackle the problem of using a dedicated system that continuously executes the same DAG of tasks on 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. The paper is organized as follows: Section II discusses some related work. In Section III we give a formal definition of the problem. In section IV we present and prove several lemmas that are used in section V to define the proposed algorithms. We conclude the article in section VI.

II. Related Work

Nowadays more and more attention is being payed 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 throughput reachable 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 intense applications as flow applications. The failures may append for numerous reasons 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 that leads to more errors when the frequency is scaled down [14]. Defining a global error model for all distributed systems is however not conceivable as they are composed of so much elements each with their own failure model. Indeed, increasing the speed of computation or processing can also lead to more contention in buses or networks and less reliable tasks. It can thus affect the reliability of the system. In this paper, we assume that optimizing the energy consumption of the system by decreasing its speed leads to decrease the fault rate in addition to period

minimization. We propose two algorithms that minimize either the energy consumption for an optimal period or find the lowest period for a minimal energy consumption.

III. Framework

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

A. 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). So a data set enters the graph at the source task and traverses 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.

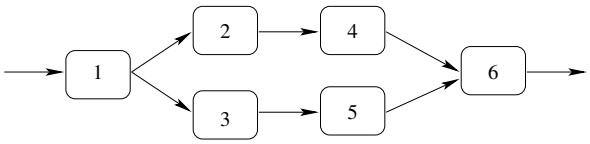

Figure 1. Illustrating task graph

B. 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, thank to the data buffering, the former are covered by computations and thus can be neglected.

The tasks are statically allocated to the 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 on a given mapping.

A machine M_u can run 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: $f_u^{l_u} < f_u^{l'_u}$ if $l_u > l'_u$. 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 the failures. Considering L, the configuration of the platform, it is possible to compute

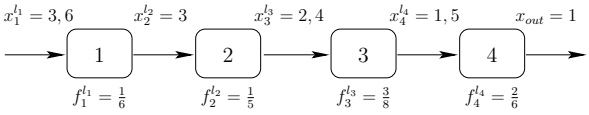

Figure 2. Example for the backward computation of the necessary amount of data sets for each task in a linear application, taking into account the failure rates.

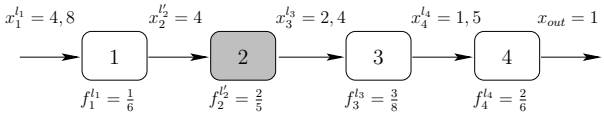

Figure 3. The acceleration of machine M_2 implies a higher number of input data sets at the application entry.

$x_i^{l_u}$ 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.

C. Example of the Platform and Execution Model

To clarify the above stated platform and execution model, we consider the application in Figure 2. To keep the example simple, we suppose that task T_i is mapped onto machine M_i and 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 $f_1^{l_1} = 1/6$. For the other tasks holds the same.

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 ($x_{out} = 1$). 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.

D. Throughput/Period Model

We define the platform throughput as the 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.

E. 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. So the static energy needed to output one data out of the system is $E^L_{stat}(u) = \varepsilon_u \times P(L)$ 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 a task. Considering one task T_i, the dynamic part of the energy consumption is about s^{β_i} where $\beta_i > 1$ an arbitrary rational number [15]. We introduce the following positive constants λ_i and C_i to guarantee the dimensional homogeneity of the equation 1. The dynamic energy consumed during one period by the machine M_u is the sum the energy consumed to perform all the tasks needed for that period:

$$E^L_{dyn}(u) = \sum_{T_i|u=a(i)} \left(\lambda_i \left(\frac{C_i \times s_u}{\alpha_u^{l_u}} \right)^{\beta_i} \times p_i^{l_u} \right) \quad (1)$$

F. 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.

IV. 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:

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

We now consider the influence of slowing down or accelerating machines or groups of 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 $l'_u \leq l_u$ for each machine $M_u \in M$, i.e., L' has some accelerated machines in comparison to configuration L. We aim at proving that $\forall T_i \in T, x_i^{l_u} \leq x_i^{l'_u}$ 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 $\frac{1}{1 - f_i^{l_u}} < \frac{1}{1 - f_i^{l'_u}}$. 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 III-C), this incrementation recursively modifies the $x_j^{l_v}$ by following backwards the dependency constraints in the group of machines. So the global workload of each machine M_v where $a(j) = v$ 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 $l'_v \leq l_v, \forall M_v \in M$, 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 $p_u^{l_u} \leq p_u^{l'_u}$.

First, with Lemma 1, we know that for all tasks the amount of work is more important in L' than in L: $x_i^{l_u} \leq x_i^{l'_u}, \forall t_i \in T$. 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: $\forall t_i \in T \;\; s.t. \;\; u = a(i) : \alpha_u^{l_u} x_i^{l_u} \leq \alpha_u^{l'_u} x_i^{l'u}$ and we get: $\forall t_i \in T \;\; s.t. \;\; u = a(i) : p_i^{l_u} \leq \frac{\alpha_u^{l'_u} x_i^{l_u}}{\alpha_u^{l_u} x_i^{l_u}} \times p_i^{l_u}$.

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':

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

■

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 $l'_v \leq l_v, \forall M_v \in M$, 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': $p_c^{l_c} \leq p_c^{l'_c}$. By definition, we deduce that the period of configuration L is lower than the maximum period in L': $P(L) \leq p_c^{l'_c} = \max_{M_u \in M}(p_u^{l'_u}) = P(L')$. ■

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

The principle of the proof of this lemma is to compare two configurations L and L', L' with more accelerated

machines. The detailed proof is available in the companion research report [16].

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

The principle of the proof is the same as Lemma 3. The detailed proof is available in the companion research report [16].

V. 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.

A. 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 a polynomial time. Indeed, in the worth 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. So the complexity of the algorithm 1 is $O(p \times n)$.

Algorithm 1: $OptPer(L)$

```
M_c ← critical machine of L
L̂ ← L
l̂_c = l̂_c − 1
M̂_c ← critical machine of L̂
while (l̂_c ≥ 0) do
    if (P(L̂) < P(L)) then
        L ← L̂
        M_c ← M̂_c
    l̂_c = l̂_c − 1
    M̂_c ← critical machine of L̂
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.

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

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 $P(L')$:

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

Proof: From the definition of function $A(L)$ in equation 3 we know that: $\forall L' \in A(L) \Rightarrow l'_u \leq l_u$.

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 than the period of the configuration L. ■

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

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

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

Proof: First we note that, based on the definition of the system period $P(L)$ given in Section III, 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 $(l'_1, l'_2, \ldots, l'_p)$ values.

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

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 necessary 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:

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

and

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

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:

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

Then, from Equations 5 and 6, we deduce:

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

■

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 the 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. So $S' = \langle L_1, \ldots, L_a, \ldots, L_k \rangle$ 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 subsequence $\langle L_{a+1}, \ldots, L_k \rangle$ is a sub-optimal configuration so it is not considered in the following as we are just 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. So we just have to look at configurations L'' in $\langle L^*, \ldots, L_a \rangle$ whose period is optimal ($P(L^*) = P(L'')$).

By definition L^* is the first configuration that reaches an optimal period in S'. So an other configuration L'' in S' with an optimal period is such that $\forall u : l''_u \leq l^*_u$. So 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 $E^* = E(L^*) \leq E(L'')$ and then that $E(OptPer(L)) = E(L^*)$ is optimal. ■

B. Algorithm OptEner

Algorithm 2: $OptEner(L)$

```
1  M_c ← critical machine
2  L̂ ← L
3  l̂_c ← l̂_c − 1
4  M̂_c ← critical machine
5  while (l̂_c ≥ 0) do
6      if ((E(L̂) < E(L)) ∨ ((E(L̂) = E(L)) ∧ (P(L̂) ≤ P(L)))) then
7          L ← L̂
8          M_c ← M̂_c
9      l̂_c ← l̂_c − 1
10     M̂_c ← critical machine of L̂
11 return L
```

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. So 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 the algorithm 2 is $O(p \times n)$ considering the same arguments used to compute the complexity of the algorithm 1.

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 same in L as in L', then the energy consumption of L is lower than the energy consumption of L':

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

The proof of this lemma is based upon Corollary 1 and Lemma 4. The detailed proof is available in the companion research report [16].

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

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

The proof of this theorem is based on Lemma 6 and obeys to the same principle as Theorem 1. The detailed proof is available in the companion research report [16].

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

The proof involves the construction of a sequence of configurations as in the proofs of previous theorems. The detailed proof is available in the companion research report [16].

VI. 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 on the machines 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. As these algorithms give optimal results there is no need to simulate them and assess their performance.

In practical systems these two criteria are however often opposed and reducing one of them usually leads to increase the other one. So that there is a need to find configurations that balance these criteria and we will consider as future work the tri-criteria problem with energy, failures and throughput. We plan to use multicriteria techniques to find good configurations and to develop heuristics that give efficient results in practical cases.

References

[1] J. Subhlok and G. Vondran, "Optimal mapping of sequences of data parallel tasks," *SIGPLAN Not.*, vol. 30, pp. 134–143, August 1995.

[2] M. Tanaka, "Development of desktop machining microfactory," *Journal RIKEN Rev*, vol. 34, pp. 46–49, April 2001.

[3] E. Descourvières, S. Debricon, D. Gendreau, P. Lutz, L. Philippe, and 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, and L. Toch, "Assessing new approaches to schedule a batch of identical intree-shaped workflows on a heterogeneous platform," *IJPEDS*, vol. 27, no. 1, pp. 79–107, 2012.

[6] A. Benoit, A. Dobrila, J.-M. Nicod, and L. Philippe, "Mapping workflow applications with types on heterogeneous specialized platforms," *Parallel Computing, Special Issue ISPDC'09*, vol. 37, no. 8, pp. 410–427, 2011.

[7] G. Da Costa, J.-P. Gelas, Y. Georgiou, L. Lefevre, A.-C. Orgerie, J.-M. Pierson, O. Richard, and 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, and 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, and Y. Robert, "Performance and energy optimization of concurrent pipelined applications," in *IPDPS'10*. Atlanta, USA: IEEE, 2010, pp. 1–12.

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

[11] H. Aydin and Q. Yang, "Energy-aware partitioning for multiprocessor real-time systems," in *IPDPS'03*. IEEE, 2003.

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

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

[14] V. Degalahal, L. Li, V. Narayanan, M. Kandemir, and M. J. Irwin, "Soft errors issues in low-power caches," *IEEE Trans. on VLSI*, vol. 13, pp. 1157–1166, oct 2005.

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

[16] A. Ben Othman, J.-M. Nicod, L. Philippe, and V. Rehn-Sonigo, "Optimal Energy Consumption and Throughput for Workflow Applications on Distributed Architectures," FEMTO-ST, Research Report RR-2012-01, Apr. 2012, http://hal.archives-ouvertes.fr/hal-00691119.