Analyzing Scheduling with Transient Failures

Erik Saule and Denis Trystram

LIG, Grenoble University

1 Introduction

When using thousands of processors simultaneously, the application developer can no longer assume that the computing platform is failure free. The probability that one of the computing processors crashes drastically increases with the number of processors [9]. Safety issues not only occur in large scale computing platforms but also in some real-time embedded systems [2].

There exist a lot of safety models. One of the most popular has been proposed by Shatz [11] where the application is represented as a set of tasks to schedule on processors. Under this model, the faults are supposed to be transient (which means that the processors recover just after a failure), their occurrence are supposed to follow a Poisson's process and to be statistically independent. The main performance index related to safety is the reliability, *i.e.* the probability that the application completes successfully. Improving the reliability can be achieved by a smart allocation of tasks on the processors. However, such an allocation can not improve the reliability by more than one order of magnitude. To reach a better reliability while not worsening the makespan too much, replication should be used.

Unfortunately, the optimization of both the efficiency and the reliability is a difficult problem. Several algorithms have been proposed but most results are heuristics which do not lead to theoretical guarantees [2, 5]. Three prior works give strict guarantees on obtained schedules. First, without allowing replication and under the fail-stop model (processors can crash and will never be operational

again), approximation algorithms where provided in [6, 8] for independent tasks. Second, with time-independent failures, the problem of scheduling a pipeline of tasks has been considered in [3].

Realistic models are likely to be intractable and heuristics should be the only way to tackle them. In this paper, we provide a theoretical analysis of two basic problems which should help to design such heuristics for more general graphs. These problems are the scheduling of a chain of tasks, on heterogeneous processors (in Section 3) and the scheduling of independent tasks on identical processors (in Section 4). Both problems are solved using the same framework (dynamic programming, scaling technique and Pareto set approximation). They represent two interesting sub-cases of the general problem that can be solved almost optimally.

2 Multi-objective approximation

In multi-objective optimization, the solutions are not totally ordered. It is often possible to degrade one objective to improve the other one. However, a solution can be absolutely better than another one which is the relation of Pareto dominance. A solution is said to be **Pareto optimal** if no solution is as good as it is on all objective values and better on at least one. The **Pareto set** (denoted by P^*) of a problem is the set of all Pareto optimal solutions [12].

Pareto sets are usually of exponential cardinality. Papadimitriou and Yannakakis introduced the concept of approximated Pareto set in [10]. P is a (ρ_1, ρ_2)-**approximation of the Pareto set** P^* if each solution $\pi^* \in P^*$ is (ρ_1, ρ_2) approximated by a solution $\pi \in P$. It was shown that most scheduling problems admit such a polynomial Pareto set approximation.

Classically, an approximated Pareto set is obtained using a $\langle \bar{\rho_1}, \rho_2 \rangle$-approximation algorithm. Such an algorithm takes as a parameter a threshold ω on one objective and returns a solution which first objective is bounded by $\rho_1 \omega$ and which is a ρ_2-approximation of the second objective among the solutions where the first

objective is better than ω.

This algorithm is called multiple times with different values ω of the constrained objective. The successive values of ω are taken accordingly to a geometrical sequence of common ratio $(1+\rho)$ from a lower bound ω^{min} to an upper bound ω^{max} that cover all the Pareto set. The set of generated solutions is a $((1+\rho)\rho_1, \rho_2)$-approximation of the Pareto set. The principle is that the solution generated with $\omega = (1+\rho)^i \omega^{min}$ is a $((1+\rho)\rho_1, \rho_2)$-approximation of all solutions whose values of the first objective is in $[(1+\rho)^{i-1}\omega^{min}; (1+\rho)^i \omega^{min}]$. The generated Pareto set approximation is polynomial as long as the size of ω^{min} and ω^{max} are polynomial in the size of the instance. This technique was first used in [1] and applied later on the reliability objective in [6, 8] under the fail-stop model (see [8] for details on the approximation technique).

3 Chain of tasks on heterogeneous processors

3.1 Model

Let $T = \{1, \ldots, n\}$ be a chain of tasks to schedule on the set $M = \{1, \ldots, m\}$ of processors. The result of task $i-1$ must be available before task i starts. The computation time of task i on processor j is denoted by p_{ij}. In this model, the communication times are neglected, that is, the result of task i is available on all processors right after the completion of i. The failure occurrences on processor j follows a Poisson's process of parameter λ_j, the failure rate of j. Thus, the probability of success of the execution of i on processor j is $P(i,j) = e^{-\lambda_j p_{ij}}$.

Reliability is increased by using active replication of tasks. Computing the reliability is difficult in presence of replicas and precedence since two different tasks can be executed at the same time. That is why we use the replication for reliability scheme. The point is that each replica of task i waits for the completion of all the replicas of task $i-1$.

A schedule is defined by a function $\pi : T \to \mathcal{P}(M)$ where $\mathcal{P}(M)$ is the set of all the subsets of M. A schedule is valid as soon as each task is executed on at

least one processor.

In this model, the makespan of a schedule is $C_{max}(\pi) = \sum_{i \in T} \max_{j \in \pi(i)} p_{ij}$. The reliability of the schedule is the probability of all tasks to be operational and a task is operational if at least one of the replica of the task finishes its execution successfully. Thus, reliability (the probability of success) of a schedule is $rel(\pi) = \prod_{i \in T} \left(1 - \prod_{j \in \pi(i)} (1 - P(i,j))\right)$.

3.2 Solving Using Dynamic Programming

Let us present an $\langle \bar{\rho_1}, \rho_2 \rangle$-approximation algorithm by setting a constraint on the makespan. Thus, we are interested in schedules of optimal reliability whose makespan is less than ω.

Since the application graph is a chain and replication is used for reliability, if two replica are scheduled at the same time then they are replica of the same task. The maximum possible reliability of schedules of ω time units can be obtained using the following principle : For each task, we have to choose which processors will execute a replica of this task. Remark that, if task i is scheduled on processor j, then it should also be scheduled on j' such that $p_{ij'} \leq p_{ij}$. This will improve the reliability without worsening the makespan. If p_{ij} time units are spent for executing task i then it remains $\omega - p_{ij}$ time units for the other tasks. This principle is expressed in the recursive expression of the optimal reliability $R(\omega, n)$ which can be reached on the first n tasks in ω time units:

$R(\omega, n) = \max_{j \in M} \left(R(\omega - p_{nj}, n-1) \left(1 - \prod_{j' \in M | p_{nj'} \leq p_{nj}} (1 - P(n, j'))\right)\right)$.
$R(\omega, 0) = 1$ if $\omega \geq 0$ and 0 otherwise. Notice that there is no task 0, it is only used to initialize the recursive expression.

This dynamic programming formulation leads to an algorithm that computes a matrix of n rows and ω columns where the value of the element at the xth column and y rows is $R(x, y)$. The matrix is filled row after row using the recursive equation. The last row is then parsed to find the highest value of $R(x, n)$ that is the optimal reliability in schedules of makespan better than ω.

Remark that the value of $R(y, x)$ is obtained from a value of row $x - 1$ and

from a set of processors $M_j = j' \in M \mid p_{nj'} \leq p_{nj}$. Each element $[x, y]$ of the matrix is associated with a schedule that reaches $R(x, y)$ which is constructed from the schedule associated with the value from row $x - 1$ in which task y is scheduled on the set of processors M_j.

The complexity is this algorithm is dominated by the computation of the matrix. The computation of each value is done in $O(m)$. The overall complexity is in $O(\omega nm)$ which is pseudo polynomial.

3.3 Approximation Algorithm

The dynamic programming expression is similar to the one used in the multi subset sum problem [4]. For this problem, a $\mathcal{FPTAS}$ algorithm can be derived by using a scaling technique (also called trimming in [4]). The same technique holds on our problem and is briefly recalled below.

The approximation algorithm is in three steps. First, from the instance I of the problem, it derives an approximated instance I' by using the following transformation where $x > 1$: $p'_{ij} = \lfloor \frac{p_{ij}}{x} \rfloor$, $\omega' = \lceil \frac{\omega}{x} \rceil$ and $P'(i, j) = P(i, j)$. Second, it solves instance I' using the dynamic programming algorithm presented in Section 3.2. Finally, it returns the solution of I' as a solution of I.

The transformation of the instance ensures the following properties:

PROPOSITION 1. *All valid solutions in the original instance I are still valid solutions of instance I': Let π be a schedule such that $\sum_i \max_{j \in \pi(i)} p_{ij} \leq \omega$ then $\sum_i \max_{j \in \pi(i)} p'_{ij} \leq \omega'$.*

PROPOSITION 2. *All valid solutions of the transformed instance I' do not exceed ω too much in the original instance I: Let π be a schedule such that $\sum_i \max_{j \in \pi(i)} p'_{ij} \leq \omega'$ then $\sum_i \max_{j \in \pi(i)} p_{ij} \leq \omega + (n + 1)x$.*

Both proofs are simple arithmetic and thus are omitted but they can be found in [4]. Using $x = \frac{\omega \epsilon}{n+1}$ as the scaling factor, the following theorem is obtained.

THEOREM 3. *The described algorithm is a* $\langle \overline{1+\epsilon}, 1 \rangle$*-approximation for scheduling a chain on heterogeneous processors subject to failure.*

The optimality on the reliability is obtained from Proposition 1. This proposition basically says that all valid schedules in I are also valid in I'. Since I' is solved exactly, the solution returned by the algorithm has a better-than-optimal reliability (this is not contradictory since the solution returned by the approximation algorithm may not match the constraint on the makespan). The approximation $1+\epsilon$ on the makespan comes from Proposition 2 and the well chosen value of x.

The time complexity of the algorithm is $O(\frac{n^2 m}{\epsilon})$. The values of the makespan are bounded by $\omega^{min} = \sum_i \min_j p_{ij}$ and $\omega^{max} = \sum_i \max_j p_{ij}$. A $((1+\epsilon)(1+\rho), 1)$-approximation of the Pareto set can be constructed in $O(\log_{1+\rho}(\frac{\omega^{max}}{\omega^{min}})\frac{n^2 m}{\epsilon})$ using the technique recalled in Section 2.

4 Independent tasks on homogeneous processors

4.1 Model

In this section, we consider the problem of scheduling independent tasks on homogeneous processors. Formally, the processing time of a task does not depend on its allocation $\forall i \in T, p_{ij} = p_i$ and all processors have the same failure rate $\forall j \in M, \lambda_j = \lambda$.

Since there is no dependencies between tasks, there is no need to insert idle times. Thus, the makespan is not expressed in the same way as before. With independent tasks, we have: $C_{max}(\pi) = \max_{j \in M} \sum_{i|j \in \pi(i)} p_i$. The expression of the reliability does not really change, the only difference is that $\forall i, \forall j, P(i,j) = P(i)$ since the failure rate of all the processors are homogeneous.

4.2 Two-phase approximation

Optimizing the makespan, even without duplication and reliability, is a $\mathcal{NP}$-hard problem which is classically solved using approximation algorithms. The most classical one is List Scheduling proposed by Graham [7] which is a 2-approximation algorithm. Its principle is to consider each task iteratively and schedule it on the least loaded processor. The approximation ratio comes from the following bound on the makespan of a solution produced by List Scheduling: $C_{max} \leq max_i p_i + \frac{\sum_i p_i}{m}$.

With duplication to improve the reliability, the optimization is much harder since the number of copies of each task has to be determined. The result of Graham shows that there are two major parameters in the bound of the makespan namely, the size of the longest task and the total volume of computations. However, the size of the longest task does not depend on the scheduling process. We only focus on the maximum reliability which can obtained with a total computation volume less than C. The exact information we are interested in is how many times each task should be replicated. The optimal reliability is given by the following recursive expression:

$R(C,n) = \max_{j \in M} \left(R(C - jp_n, n-1)\left(1-(1-P(n)^j)\right)\right)$. $R(C,0) = 1$ if $C \geq 0$ and 0 otherwise.

As previously, this expression leads to a pseudo-polynomial optimal algorithm of time complexity in $O(Cnm)$ that provides how many time each task should be replicated. The same scaling technique as before leads to a $(1+\epsilon)$-approximation of the total computation time, in time complexity $O(\frac{n^2 m}{\epsilon})$.

4.3 Approximation algorithm

This dynamic programming approach only states how many times a task should be replicated but does not provide the actual allocation on the processors. Two replicas should not be scheduled on the same processor. Thus, a classical List Scheduling algorithm will not produce valid solutions. However, the same princi-

ple can be used to construct the allocation function π. The algorithm LSR (List Scheduling with Replication) is described as follows. For each task i, schedule all its $\mid \pi(i) \mid$ replicas on the $\mid \pi(i) \mid$ least loaded processors. In the following the load of processor j is denoted by $C^j_{max}(\pi) = \sum_{i \mid j \in \pi(i)} p_i$. The following proposition states that the schedule is well balanced.

PROPOSITION 4. *π is well balanced:* $\forall j, j', C^j_{max}(\pi) - C^{j'}_{max}(\pi) \leq \max_i p_i$.

Proof. The proof is done by proving by induction that after the allocation of task i, the following property is valid $\forall j, j', C^j_{max}(\pi) - C^{j'}_{max}(\pi) \leq \max_{i' \leq i} p_{i'}$. It is straightforward to verify that the property is valid when a single task is allocated. Suppose that the property is true for the first i tasks. We are going to prove that it is also verified for the first $i+1$ tasks. $\pi_{i'}$ denotes the allocation of the first i' tasks only.

Let j and j' be two processors such that $C^j_{max}(\pi_i) - C^{j'}_{max}(\pi_i) \leq \max_{i'} p_{i'}$. The difference does not change if a replica is allocated on both processors. Consider that at the $(i+1)$th step, a replica of $i+1$ has been allocated to j but not to j' : $C^j_{max}(\pi_{i+1}) = C^j_{max}(\pi_i) + p_{i+1}$, $C^{j'}_{max}(\pi_{i+1}) = C^{j'}_{max}(\pi_i)$. By construction $C^j_{max}(\pi_i) \leq C^{j'}_{max}(\pi_i)$. We finally conclude that $C^j_{max}(\pi_{i+1}) - C^{j'}_{max}(\pi_{i+1}) = C^j_{max}(\pi_i) + p_{i+1} - C^{j'}_{max}(\pi_i) \leq p_{i+1}$ and $C^{j'}_{max}(\pi_{i+1}) - C^j_{max}(\pi_{i+1}) = C^{j'}_{max}(\pi_i) - C^j_{max}(\pi_i) - p_{i+1} \leq \max_{i'} p_{i'}$. □

Using this allocation algorithm, we are able to construct a schedule of makespan less than $C_{max}(\pi) \leq \frac{C}{m} + \max_i p_i$ and of optimal reliability. Given a feasible value of the makespan ω, the overall algorithm is: compute how many times each task should be replicated using the dynamic programming with $C = \omega m$ and then allocate the replicas using LSR. Since there is no solution for $C < m \max_i p_i$, the overall process is an approximation algorithm.

THEOREM 5. *The procedure described in this section is a $\langle \overline{2+\epsilon}, 1 \rangle$-approximation algorithm of time complexity $O(\frac{n^2 m}{\epsilon})$.*

On the makespan, the approximation factor should be decomposed as $(1+\epsilon)+1$ where $1+\epsilon$ comes from the non-optimal solving of the dynamic program-

ming and a part of 1 comes from the inaccuracy of the LSR algorithm ($\max_i p_i$ is smaller than ω). The reliability depends only on the dynamic programming which provides a better-than-optimal value for the reliability (once again, this is not contradictory since the makespan threshold is not matched).

The complexity of this algorithm is dominated by the dynamic programming part. Indeed, LSR can be implemented using a heap with a complexity in $O(nm \log m)$. The values of the makespan are bounded by $\omega^{min} = \frac{\sum_i p_{ij}}{m}$ and $\omega^{max} = \sum_i p_i$. A $((1+\epsilon)(1+\rho), 1)$-approximation of the Pareto set can be constructed in $O(\log_{1+\rho}(\frac{\omega^{max}}{\omega^{min}})\frac{n^2 m}{\epsilon})$ using a technique similar to the one of Section 2.

5 Conclusion

Optimizing the reliability at the same time as the makespan is a difficult problem. In order to reach this objective, we analyzed two core scheduling problems with replication, namely, a chain of tasks which is the simplest precedence constraint on heterogeneous processors and independent tasks on identical processors. Since almost optimal solutions can be constructed for the two core problems, some interesting extensions should be studied such as scheduling applications with arbitrary structure on identical processors and independent task on uniform processors. This work could be the milestone for the design of good heuristics for more general and realistic problems since almost optimal solutions can be constructed for the two core problems.

References

[1] E. Angel, E. Bampis, and A. Kononov. A FPTAS for approximating the unrelated parallel machines scheduling problem with costs. *In Proc. of ESA*, pages 194–205, 2001.

[2] I. Assayad, A. Girault, and H. Kalla. A bi-criteria scheduling heuristic for distributed embedded systems under reliability and real-time constraints. In *Proc. of DSN*, pages 347–356, 2004.

[3] A. Benoit, V. Rehn-Sonigo, and Y. Robert. Optimizing latency and reliability of pipeline workflow applications. In *Proc. of HCW*, 2008.

[4] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to algorithms*. MIT press, 1990.

[5] A. Dogan and F. Özguner. Matching and scheduling algorithms for minimizing execution time and failure probability of applications in heterogeneous computing. *IEEE TPDS*, 13(3):308–324, 2002.

[6] J. Dongarra, E. Jeannot, E. Saule, and Z. Shi. Bi-objective scheduling algorithms for optimizing makespan and reliability on heterogeneous systems. In *SPAA '07*, pages 280–288. ACM press, June 2007.

[7] R. L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, 45:1563–1581, 1966.

[8] E. Jeannot, E. Saule, and D. Trystram. Bi-objective approximation scheme for makespan and reliability optimization on uniform parallel machines. In *Euro-Par 2008*, pages 877–886, 2008.

[9] A. Oliner, R. Sahoo, J. Moreira, M. Gupta, and A. Sivasubramaniam. Fault-aware job scheduling for bluegene/l systems. In *Proc. of IPDPS*, 2004.

[10] C. Papadimitriou and M. Yannakakis. On the approximability of trade-offs and optimal access of web sources. In *Proc. of FOCS*, pages 86–92, 2000.

[11] S. Shatz and J. Wang. Task allocation for maximizing reliability of distribued computer systems. *IEEE TC*, 41(9):1156–1169, 1992.

[12] V. T'kindt and J. Billaut. *Multicriteria Scheduling*. Springer, 2007.