Iso-Level CAFT: How to Tackle the Combination of Communication Overhead Reduction and Fault Tolerance Scheduling

Mourad Hakem

LIFC Laboratory, Université de Franche-Comté, Belfort, France
Mourad.Hakem@lifc.univ-fcomte.fr

Abstract. To schedule precedence task graphs in a more realistic framework, we introduce an efficient fault tolerant scheduling algorithm that is both contention-aware and capable of supporting ε arbitrary fail-silent (fail-stop) processor failures. The design of the proposed algorithm which we call Iso-Level CAFT, is motivated by (i) the search for a better load-balance and (ii) the generation of fewer communications. These goals are achieved by scheduling a chunk of ready tasks simultaneously, which enables for a global view of the potential communications. Our goal is to minimize the total execution time, or latency, while tolerating an arbitrary number of processor failures. Our approach is based on an active replication scheme to mask failures, so that there is no need for detecting and handling such failures. Major achievements include a low complexity, and a drastic reduction of the number of additional communications induced by the replication mechanism. The experimental results fully demonstrate the usefulness of Iso-Level CAFT.

1 Introduction

With the advent of large-scale heterogeneous platforms such as clusters and grids, resource failures (processors/links) are more likely to occur and have an adverse effect on the applications. Consequently, there is an increasing need for developing techniques to achieve fault tolerance, *i.e.*, to tolerate an arbitrary number of failures during execution. Scheduling for heterogeneous platforms and fault tolerance are difficult problems in their own, and aiming at solving them together makes the problem even harder. For instance, the latency of the application will increase if we want to tolerate several failures, even if no actual failure happens during execution.

In this paper, we introduce the Iso-Level Contention-Aware Fault Tolerant (Iso-Level CAFT) scheduling algorithm (a new version of CAFT [4] that were initially designed to address both problems of network contention and fault-tolerance scheduling) that aims at tolerating multiple processor failures without sacrificing the latency. Iso-Level CAFT is based on an active replication scheme to mask failures, so that there is no need for detecting and handling such failures. Our choice for the active replication scheme is motivated by two important

Y. Dou, R. Gruber, and J. Joller (Eds.): APPT 2009, LNCS 5737, pp. 259–272, 2009.

advantages. On the one hand, the schedules obtained are static, thus it is easy to have a guarantee on the latency of the schedule. On the other hand, the deployment of the system does not require complicated mechanisms for failure detection. Major achievements include a low complexity, and a drastic reduction of the number of additional communications induced by the replication mechanism.

We suggest to use the bi-directional one-port architectural model, where each processor can communicate (send and/or receive) with at most one other processor at a given time-step. In other words, a given processor can simultaneously send a message, receive another message, and perform some computation. The bi-directional one-port model seems closer to the actual capabilities of modern networks (see the discussion of related work in [4,5,6]). Indeed, it seems to fit the performance of some current MPI implementations, which serialize asynchronous MPI sends as soon as message sizes exceed a few megabytes [4].

The rest of the paper is organized as follows: Section 2 presents basic definitions and assumptions. Then we describe the principle of the new Iso-Level CAFT algorithm in Section 3. We experimentally compare Iso-Lvel CAFT with its initial version CAFT in Section 4; the results assess the very good behavior of the new algorithm. Finally, we conclude in Section 5.

The review of related work on fault tolerance scheduling is provided in [4].

2 Framework

The execution model for a task graph is represented as a weighted Directed Acyclic Graph (DAG) $G = (V, E)$, where V is the set of nodes corresponding to the tasks, and E is the set of edges corresponding to the precedence relations between the tasks. In the following we use the term node or task indifferently; $v = |V|$ is the number of nodes, and $e = |E|$ is the number of edges. In a DAG, a node without any predecessor is called an *entry* node, while a node without any successor is an *exit* node. For a task t in G, $\Gamma^-(t)$ is the set of immediate predecessors and $\Gamma^+(t)$ denotes its immediate successors. A task is called *ready* if it is unscheduled and all of its predecessors are scheduled. We target a heterogeneous platform with m processors $\mathcal{P} = \{P_1, P_2, \ldots, P_m\}$, fully interconnected. The link between processors P_k and P_h is denoted by l_{kh}. Note that we do not need to have a physical link between any processor pair. Instead, we may have a switch, or even a path composed of several physical links, to interconnect P_k and P_h; in the latter case we would retain the bandwidth of the slowest link in the path for the bandwidth of l_{kh}. For a given graph G and processor set $\mathcal{P}$, $g(G, \mathcal{P})$ is the *granularity*, *i.e.*, the ratio of the sum of slowest computation times of each task, to the sum of slowest communication times along each edge. $\mathcal{H}(\alpha)$ is the head function which returns the first task from a sorted list α, where the list is sorted according to tasks priorities (ties are broken randomly). The number of tasks that can be simultaneously ready at each step in the scheduling process is bounded by the width ω of the task graph

(the maximum number of tasks that are independent in G). This, implies that $|\alpha| \leq \omega$.

Our goal is to find a task mapping of the DAG G on the platform $\mathcal{P}$ obeying the one-port model. The objective is to minimize the latency $\mathcal{L}(G)$, while tolerating an arbitrary number ε of processor failures. Our approach is based on an active replication scheme, capable of supporting ε arbitrary fail-silent (a faulty processor does not produce any output) and fail-stop (no processor recovery) processor failures.

3 The Iso-Level CAFT Scheduling Algorithm

In the previous version of CAFT algorithm [4], we consider only one ready task (the one with highest priority) at each step, and we assign all its replicas to the currently best available resources. Instead of considering a single task, we may deal with a chunk of several ready tasks, and assign all their replicas in the same decision making procedure. The intuition is that such a "global" assignment would lead to better load balance processor and link usage.

We introduce a parameter B for the chunk size: B is the maximal number of ready tasks that will be considered at each step. We select the B tasks with the higher bottom levels $b\ell(t)$ (the length of the longest path starting at t to an exit node in the graph) and we allocate them in the same step. Then, we update the set of ready tasks (indeed some new tasks may have become ready), and we sort them again, according to bottom levels. Thus, we expect that the tasks on a critical path will be processed as soon as possible.

The difference between CAFT and the new version, which we call Iso-Level CAFT (or ILC), is sketched in Algorithm 3.1. With CAFT we take the ready task with highest priority all allocate all its replicas before proceeding to the next ready task. In contrast, with Iso-Level CAFT, the second replicas of tasks in the same chunk are allocated only after all first replicas have been placed. Intuitively, this more global strategy will balance best resources across all tasks in the chunk, while CAFT may assign the $\varepsilon + 1$ best resources to the current task, at the risk of sacrificing the next one, even though it may have the same bottom level.

We point out that we face a difficult tradeoff for choosing an appropriate value for B. On the one hand, if B is large, it will be possible to better balance the load and minimize communication costs. On the other hand, a small value of B will enable us to process the tasks on the critical path faster. In the experiments (see Section 4) we observe that choosing $B = m$, the number of processors, leads to good results.

Theorem 1. *The time complexity of Iso-Level CAFT is*

$$O\left(em(\varepsilon+1)^2 \log(\varepsilon+1) + v \log \omega\right)$$

Proof. The proof is similar to that given in [4]. Note that since $\varepsilon < m$, we can derive the upper bound $O\left(em^3 \log m + v \log \omega\right)$.

Algorithm 3.1 CAFT vs Iso-Level CAFT (ILC)

```
1: initialization; U ← V;
2: while U ≠ ∅ do
3:    T ← H(α); ILC: repeat B times (*CAFT: |T| = 1  |  ILC: |T| = B*)
4:    for 1 ≤ i ≤ ε + 1 do
5:       for t ∈ T do
6:          allocate task-replica t^(i) to processor with shortest finish time
7:       end for
8:    end for
9: end while
```

Notice that, allocating many copies of each task will severely increase the total number of communications required by the algorithm: we move from e communications (one per edge) in a mapping with no replication (fault free schedule), to $e(\varepsilon+1)^2$ with replication (fault tolerant schedule), a quadratic increase. In fact, duplicating each task $\varepsilon+1$ times is an absolute requirement to resist to ε failures, but duplicating each precedence edge $e(\varepsilon+1)^2$ times is not mandatory. We can decrease the total number of communications from $e(\varepsilon+1)^2$ down to $e(\varepsilon+1)$ as it was proved in [4]. Unfortunatly, this reduction does not work all the time. The linear number of communications $e(\varepsilon+1)$ holds only in special cases, typically for tasks having a unique predecessor, or when every replica of all predecessors are mapped onto distinct processors or when all the replicas belonging to the same processor communicate with only the same successor-replica.

The problem becomes more complex when tasks have more than one predecessor and several replicas of predecessors mapped on the same processor communicate with different successor-replicas. In the following, we show how to reduce this overhead in the design of Iso-Level CAFT.

3.1 Reducing Communication Overhead

When dealing with realistic model platforms, contention should be considered in order to obtain improved schedules. We account for communication overhead during the mapping process by removing some of the communications. To do so, we propose the following mapping scheme.

Let t be the current task to be scheduled. Consider a predecessor t_j of t, $j \in \Gamma^-(t)$, that has been replicated on $\varepsilon+1$ distinct processors. We denote by $\mathcal{D}_u$ the set of replicas assigned to processor $\mathcal{P}_u$, and $\eta_u = |\mathcal{D}_u|$ its cardinality. The maximum cardinality is $\eta = \max_{1\le u\le m} \eta_u$. Also we denote by $\mathcal{N}$ the number of processors involved/used by all replicas of tasks in $\Gamma^-(t)$.

We would like to reduce the number of communications from all predecessors t_j to t when possible. The idea is to attempt to place each replica on the non-locked (locked processors are already either involved in a communication with a replica of t, or processing it) processor which currently contains the most predecessor replicas. To this purpose, we sort processors by non increasing order of number of replicas $\eta_u, 1 \le u \le m$, assigned to them. At each step in

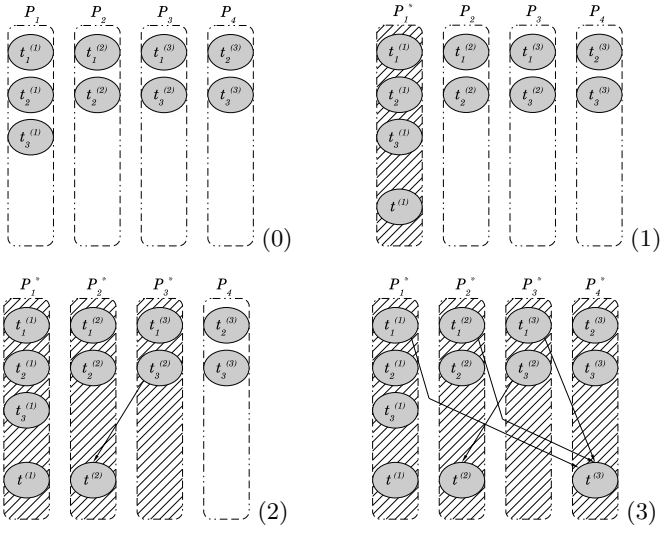


Fig. 1. Iso-Level CAFT Scheduling Steps

the mapping process, we try to take communications from replicas belonging to the non-locked processors, whenever possible. If not, we insert ε additional communications.

Fig. 1 illustrates this procedure. We set $\varepsilon = 2$ in this example. At step (0), no processor is blocked. The three predecessors of the current task t, namely t_1, t_2 and t_3, are assigned. At step (1), we place the first replica $t^{(1)}$ on P_1, which becomes locked. This is represented in the figure with a superscript $*$, and the processor is also hatched in the figure. No communication is added in this case. At step (2), we need to add a communication from P_3 to P_2, and thus we have three locked processors. At step (3), we place replica $t^{(3)}$ on the only non-locked processor which is P_3, and we need to add extra communication since all processors are locked.

Theorem 2. *The schedule generated by Iso-Level CAFT algorithm is valid and resists to ε failures.*

Proof. The proof is similar to that of CAFT (see [4])

In the following, we give an analytical expression of the actual number of communications induced by the *Iso-Level CAFT* algorithm. First we give an interesting upper bound for special graphs, and then we derive an upper bound for the general case.

Special graphs

First, we bound the number of communications induced by Iso-Level CAFT for special graphs like classical kernels representing various types of parallel algorithms [1]. The selected task graphs are:

(a) LU: LU decomposition
(b) LAPLACE: Laplace equation solver
(c) STENCIL: stencil algorithm
(d) DOOLITTLE: Doolittle reduction
(e) LDM^t: LDM^t decomposition

Miniature versions of each task graph are given in Fig. 2.

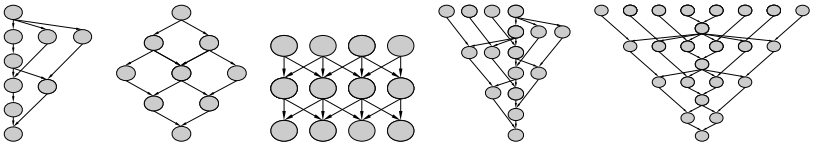


Fig. 2. Classical kernels of parallel algorithms

Theorem 3. *The number of messages generated by Iso-Level CAFT for the above special graphs is at most*

$$V_2(\varepsilon+1) + V_3\left(\varepsilon\left\lceil\frac{(\varepsilon+2)}{2}\right\rceil + 2\right),$$

where $V_2 \leq \lfloor\frac{\varepsilon}{2}\rfloor$ *is the number of nodes of in-degree* 2 *and* $V_3 \leq \lfloor\frac{\varepsilon}{3}\rfloor$ *is the number of nodes of in-degree* 3 *in the graph.*

Proof. One feature of the special graphs is that the in-degree of every task is at most 3. At each step when scheduling current task t, we have three cases to consider, depending upon its in-degree (the cardinal of $\Gamma^-(t)$). Recall that processors are ordered by non increasing η_u values, where η_u. is the number of replicas already assigned to P_u, hence which do not need to be communicated again.

(1) $|\Gamma^-(t)| = 1$. In this case, in order to pay no communication, we just need to place each replica of t with a replica of its predecessor.

(2) $|\Gamma^-(t)| = 2$. The two redecessor tasks of t are denoted t_1 and t_2. If replicas of t_1 and t_2 are mapped on the same processor ($\mathcal{P}(t_1^{(z)}) = \mathcal{P}(t_2^{(z')}) = \mathcal{P}$ for some $1 \leq z, z' \leq \varepsilon+1$), then there is no need for any additional communication. Other replicas of t_1 and t_2 which does not satisfy the previous property are thus mapped onto singleton processors. We perform the *one-to-one mapping* algorithm to allocate the corresponding other replicas of t. For each replica, at most one communication is added.

(3) $|\Gamma^-(t)| = 3$. Here we consider the number of replicas allocated to processor $\mathcal{P}_u$, denoted as η_u.

- We place a replica on each processor with $\eta_u = 3$, thus no communication need to be paid for
- Consider a processor with $\eta_u = 2$. When allocating a replica of t on such a processor $\mathcal{P}_u$, we need to receive data from the third predecessor allocated to $\mathcal{P}_v \neq \mathcal{P}_u$. $\mathcal{P}_v$ may be either a singleton processor ($\eta_v = 1$) or it may handle two predecessors ($\eta_v = 2$).
  - if $\eta_v = 1$, then we need only one communication for mapping the replica of t. In this case $\mathcal{P}_v$ communicates only to $\mathcal{P}_u$.
  - if $\eta_v = 2$, then we may need to add extra communications. For the first $\left\lceil \frac{\varepsilon+1}{2} \right\rceil$ replicas of t, we add only one communication per replica, and lock processors accordingly. But for the remaining set $\left\lfloor \frac{\varepsilon+1}{2} \right\rfloor$ of replicas, we will have to generate $\varepsilon + 1$ communications for each of these replicas. Overall, the number of communications is at most $\left\lceil \frac{\varepsilon+1}{2} \right\rceil + (\varepsilon + 1) \left\lfloor \frac{\varepsilon+1}{2} \right\rfloor$
  Let $X = \left\lceil \frac{\varepsilon+1}{2} \right\rceil + (\varepsilon + 1) \left\lfloor \frac{\varepsilon+1}{2} \right\rfloor$. Let $Y = \varepsilon \left\lceil \frac{(\varepsilon+2)}{2} \right\rceil + 1$. If $\varepsilon = 2k$ is even, then $X = 2k^2 + k + 1 \leq 2k^2 + 2k + 1 = Y$. If $\varepsilon = 2k + 1$ is odd, then $X = 2k^2 + 2k + 1 \leq 2k^2 + 3k + 1 = Y$. In all cases $X \leq Y$, hence the number of communications is at most Y.
- Now, all remaining processors have at most one replica ($\eta = 1$). Thus task t needs its data from two other replicas. So we have to take at most two communications for each replicas mapped. Thus for the mapping of $\varepsilon + 1$ replicas, we will have at most a number of communications equal to $2(\varepsilon+1)$. Note that $2(\varepsilon + 1) \leq Y + 1 = \varepsilon \left\lceil \frac{(\varepsilon+2)}{2} \right\rceil + 2$ for all ε, hence the result.

General graphs

Theorem 4. *For general graphs, the number of messages generated by Iso-Level CAFT is at most*

$$e\left(\varepsilon \left\lceil \frac{(\varepsilon+2)}{2} \right\rceil + 1\right)$$

Proof. At each step when scheduling current task t:

(i) For the first $\left\lceil \frac{\varepsilon+1}{2} \right\rceil$ replicas, we generate at most $\sum_{u=1}^{\left\lceil \frac{(\varepsilon+1)}{2} \right\rceil} (|\Gamma^-(t)| - \eta_u)$ communications (recall that η_u is the number of replicas already assigned to P_u, hence which do not need to be communicated again). Altogether, we have at most $\left\lceil \frac{(\varepsilon+1)}{2} \right\rceil |\Gamma^-(t)|$ communications for these replicas.

(ii) We still have to map the remaining $\left\lfloor \frac{\varepsilon+1}{2} \right\rfloor$ of t replicas. In the worst case, each replica placed will generate $\varepsilon+1$ communications (this is because processors may be locked in this case).

Thus for this remaining set of replicas, the number of communications is at most $(\varepsilon + 1) \sum_{u=\left\lceil \frac{(\varepsilon+1)}{2} \right\rceil + 1}^{\varepsilon+1} (|\Gamma^-(t)| - \eta_u) \leq (\varepsilon + 1) \left\lfloor \frac{\varepsilon+1}{2} \right\rfloor |\Gamma^-(t)|$

From *(i)* and *(ii)*, we have a total number of communications of $|\Gamma^-(t)|X$, where $X = \left\lceil \frac{\varepsilon+1}{2} \right\rceil + (\varepsilon + 1) \left\lfloor \frac{\varepsilon+1}{2} \right\rfloor$. As in the proof of Theorem 3, we knwo that $X \leq Y$, where $Y = \varepsilon \left\lceil \frac{(\varepsilon+2)}{2} \right\rceil + 1$. Hence the number of communications is at most Y.

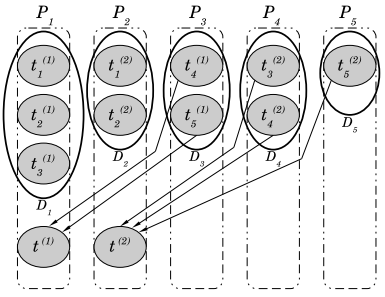


Fig. 3. Complementary/disjoint sets of replicas

Thus, summing up for all the v tasks in G, the total number of messages is at most $\sum_{u=1}^{v} |\Gamma^-(t)| \left(\varepsilon \left\lceil \frac{(\varepsilon+2)}{2} \right\rceil + 1\right) = e \left(\varepsilon \left\lceil \frac{(\varepsilon+2)}{2} \right\rceil + 1\right)$.

The following last Theorem deals with disjoint and complementary replica sets. In fact, the number of communications can be drastically reduced in such a case:

Theorem 5. *For general graphs, if at each step when scheduling a task t, we can determine replica sets $\mathcal{D}_u$ that are both disjoint ($\mathcal{D}_u \cap \mathcal{D}_{u'} = \emptyset$ if $u \neq u'$) and complementary ($\sigma_{u=1}^{m} |\mathcal{D}_u| = |\Gamma^-(t)|$, or in other words $\cup_{1 \leq u \leq m} \mathcal{D}_u$ contains a replica of each predecessor of t), then the number of messages is at most $e(\varepsilon+1)$.*

Proof. We map a replica on $\mathcal{D}_u$ and add communications from all complementary sets, which generates at most $|\Gamma^-(t)| - |\mathcal{D}_u| = |\cup_{1 \leq u' \leq m, u' \neq u} \mathcal{D}_{u'}| \leq |\Gamma^-(t)|$.

Thus, for the mapping of $\varepsilon+1$ replicas, and summing up for the set V of tasks in G, the total number of messages is at most $\sum_{t \in V} |\Gamma^-(t)|(\varepsilon+1) = e(\varepsilon+1)$.

Fig. 3 illustrates, for the mapping of the first replica $t^{(1)}$ we have $|\Gamma^-(t)| - |\mathcal{D}_1| = 5 - 3 = 2 = |\mathcal{D}_3|$. In addition, both $\mathcal{D}_1$ and $\mathcal{D}_3$ are mutually complementary/disjoints and they form a complete instance of all predecessors. Also, for the mapping of the second replica $t^{(2)}$, we have $|\Gamma^-(t)| - |\mathcal{D}_2| = 5 - 2 = 3 = |\mathcal{D}_4 \cup \mathcal{D}_5|$. Similarly, the condition of complementarity/disjunction of the sets $\mathcal{D}_2$, $\mathcal{D}_4$ and $\mathcal{D}_5$ holds.

4 Experimental Results

We assess the practical significance and usefulness of the Iso-Level CAFT algorithm through simulation studies. We compare the performance of Iso-Level CAFT with its initial version CAFT algorithm. We use randomly generated graphs, whose parameters are consistent with those used in the literature [4]. We characterize these random graphs with three parameters: (i) the number of tasks, chosen uniformly from the range $[80, 120]$; (ii) the number of incoming/outgoing edges per task, which is set in $[1, 3]$; and (iii) the granularity of

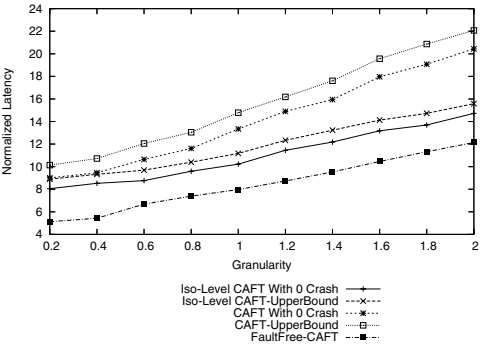
the task graph $g(G)$. We consider two types of graphs, with a granularity (a) in $[0.2, 2.0]$ and increments of 0.2, and (b) in $[1, 10]$ and increments of 1. Two types of platforms are considered, first with 10 processors and $\varepsilon = 1$ or $\varepsilon = 3$, and then with 20 processors and $\varepsilon = 5$ (a full set of results is available in the dedicated research report [3]). To account for communication heterogeneity in the system, the unit message delay of the links and the message volume between two tasks are chosen uniformly from the ranges $[0.5, 1]$ and $[50, 150]$ respectively. Each point in the figures represents the mean of executions on 60 random graphs. The fault free schedule is defined as the schedule generated without replication, assuming that the system is completely safe. Recall that the upper bounds of the schedules are computed as explained in [2].

Each algorithm is evaluated in terms of achieved latency and fault tolerance overhead $\frac{\text{CAFT}^0|\text{Iso-Level CAFT}^0|\text{CAFT}^c|\text{Iso-Level CAFT}^c - \text{CAFT}^*}{\text{CAFT}^*}$, where the superscripts $*$, c and 0 respectively denote the latency achieved by the fault free schedule, the latency achieved by the schedule when processors effectively fail (crash) and the latency achieved with 0 crash. We have also compared the behavior of each algorithm when processors crash down by computing the real execution time for a given schedule rather than just bounds (upper bound and latency with 0 crash).

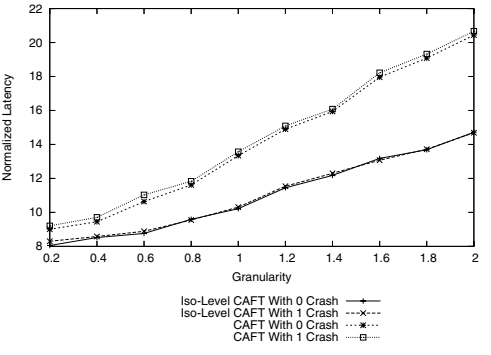
Comparing the results of Iso-Level CAFT to the results of CAFT, we observe in Fig. 4 and 5 that Iso-Level CAFT gives the best performance. It always improves the latency significantly in all figures. This is because the Iso-Level CAFT algorithm tries incrementally to ensure a certain degree of load balancing for processors by scheduling a chunk of ready tasks before considering their corresponding replicas. This better load balancing also decreases communications between tasks. Consequently, this leads to minimize the final latency of the schedule.

We also find in Fig. 6 and 7 that the performance difference between CAFT and Iso-Level CAFT increases when the granularity increases. This interesting result comes from the fact that larger granularity indicates that we are dealing with intensive computations applications in heterogeneous platforms. Thus, in order to reduce the latency for such applications, it is important to better parallelize the application. That is why we changed the backbone of CAFT to perfectly balance the load of processors at each step of the scheduling process.

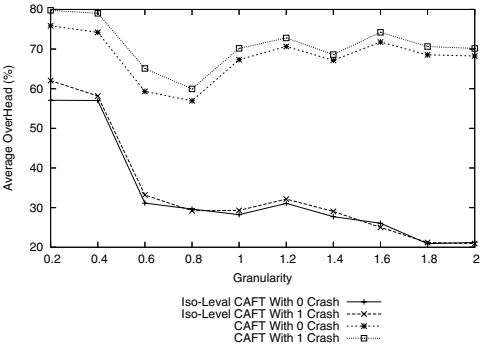
Finally, we readily observe from all figures that we deal with two conflicting objectives. Indeed, the fault tolerance overhead increases together with the number of supported failures. We also see that latency increases together with granularity, as expected. In addition, it is interesting to note that when the number of failures increases, there is not really much difference in the increase of the latency achieved by CAFT and Iso-Level CAFT, compared to the schedule length generated with 0 crash. This is explained by the fact that the increase in the schedule length is already absorbed by the replication done previously, in order to resist to eventual failures.



(a) Latency bounds

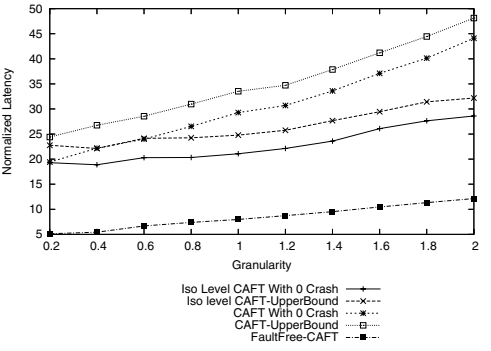


(b) Latency achieved with crash

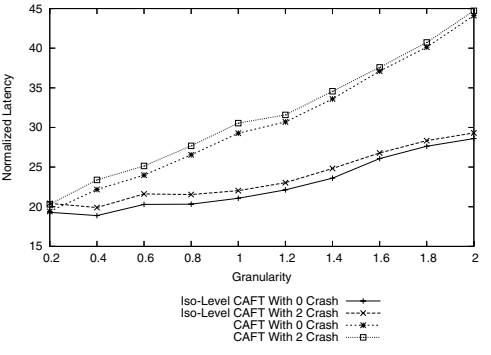


(c) Fault tolerance overhead

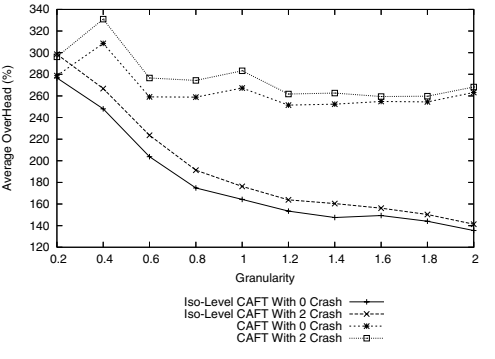
Fig. 4. Average normalized latency and overhead comparison between Iso-Level-CAFT and CAFT (Bound and Crash cases, $m = 10, \varepsilon = 1$)



(a) Latency bounds

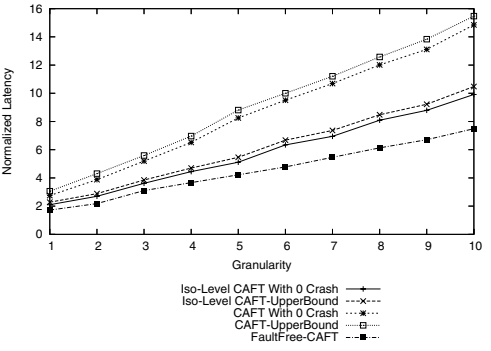


(b) Latency achieved with crash

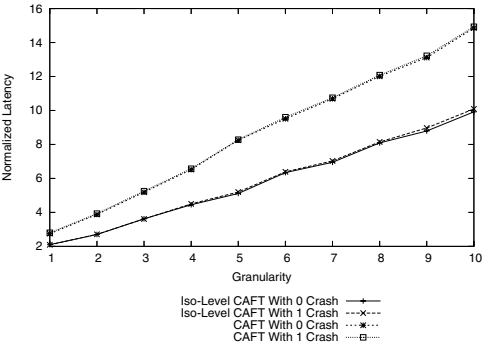


(c) Fault tolerance overhead

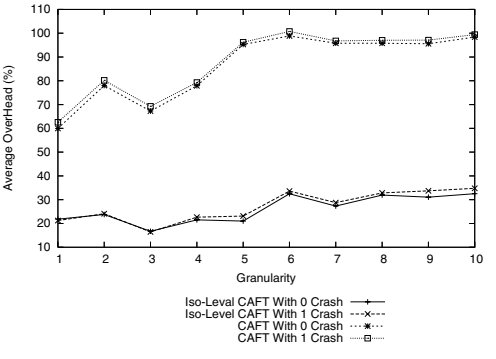
Fig. 5. Average normalized latency and overhead comparison between Iso-Level-CAFT and CAFT (Bound and Crash cases, $m = 10, \varepsilon = 3$)



(a) Latency bounds

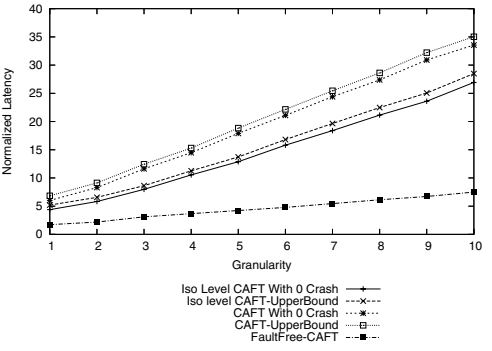


(b) Latency achieved with crash

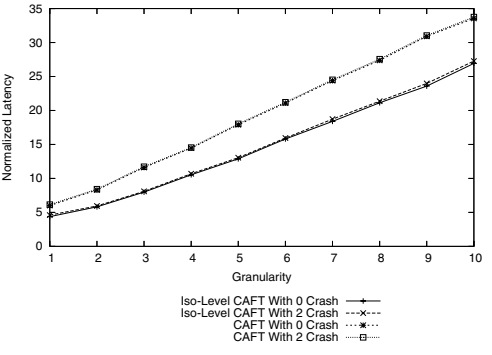


(c) Fault tolerance overhead

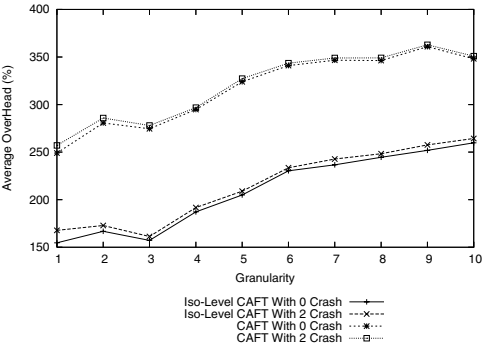
Fig. 6. Average normalized latency and overhead comparison between Iso-Level-CAFT and CAFT for coarse grain graphs $g(G) \geq 1$ (Bound and Crash cases, $m = 10, \varepsilon = 1$)



(a) Latency bounds



(b) Latency achieved with crash



(c) Fault tolerance overhead

Fig. 7. Average normalized latency and overhead comparison between Iso-Level-CAFT and CAFT for coarse grain graphs $g(G) \geq 1$ (Bound and Crash cases, $m = 10, \varepsilon = 3$)

5 Conclusion

In this paper, an efficient fault-tolerant scheduling algorithm (Iso-Level CAFT) for heteorgeneous systems is studied and analysed. Iso-Level CAFT is based on an active replication scheme, and is able to drastically reduce the communication overhead induced by task replication, which turns out a key factor in improving performance when dealing with realistic, communication contention aware, platform models. The design of Iso-Level CAFT is motivated by (i) the search for a better load-balance and (ii) the generation of fewer communications. These goals are achieved by scheduling a chunk of ready tasks simultaneously, which enables for a global view of the potential communications. To assess the performance of Iso-Level CAFT, simulation studies were conducted to compare it with CAFT, which seems to be its main direct competitor from the literature. We have shown that Iso-Level CAFT is very efficient both in terms of computational complexity and quality of the resulting schedule.

An extension of Iso-Level CAFT would be to extend it to the context of pipelined workflows made up of collections of identical task graphs (rather than dealing with a single graph as in this paper). We would then need to solve a challenging tri-criteria optimization problem (latency, throughput and fault-tolerance).

References

1. Beaumont, O., Boudet, V., Robert, Y.: A realistic model and an efficient heuristic for scheduling with heterogeneous processors. In: Proc. of the 11th Heterogeneous Computing Workshop HCW 2002 (2002)
2. Benoit, A., Hakem, M., Robert, Y.: Fault tolerant scheduling of precedence task graphs on heterogeneous platforms. In: Proc. of the 10th Int. Workshop in Advances Parallel and Distributed Computational Models APDCM 2008, pp. 1–8 (2008), http://graal.ens-lyon.fr/~abenoit/
3. Benoit, A., Hakem, M., Robert, Y.: Iso-Level CAFT: How to Tackle the Combination of Communication Overhead Reduction and Fault Tolerance Scheduling. In: RR 2008-25, LIP, ENS Lyon, France (July 2008), http://graal.ens-lyon.fr/~mhakem/
4. Benoit, A., Hakem, M., Robert, Y.: Realistic models and efficient algorithms for fault tolerance scheduling on heterogeneous platforms. In: Proc. of the 37th IEEE Int. Conference on Parallel Processing ICPP 2008, pp. 246–253 (2008), http://graal.ens-lyon.fr/~abenoit/
5. Sinnen, O., Sousa, L.: Experimental evaluation of task scheduling accuracy: Implications for the scheduling model. IEICE Transactions on Information and Systems E86-D(9), 1620–1627 (2003)
6. Sinnen, O., Sousa, L.: Communication contention in task scheduling. IEEE Trans. on Parallel and Distributed Systems 16(6), 503–515 (2005)