Fault-Tolerant Mechanism for Hierarchical Branch and Bound Algorithm

A. Bendjoudi
Centre de Recherche sur l’Information Scientifique et Technique CERIST, DTISI
3 rue des frères Aïssou, 16030 Ben-Aknoun, Algiers, Algeria
Université Abderrahmane Mira de Béjaïa
Route de Targa Ouzemmour, 06000 Béjaïa, Algeria
Email: ahcine.bendjoudi@gmail.com

N. Melab and E-G. Talbi
Université des Sciences et Technologies de Lille, France
LIFL/UMR CNRS 8022
59655 - Villeneuve d’Ascq cedex - France
Email: {noureddine.melab,el-ghazali.talbi}@lifl.fr

Abstract—Solving exactly large instances of Combinatorial Optimization Problems (COPs) using Branch and Bound (B&B) algorithms requires a huge amount of computing resources. These resources can be offered by computational grids and the scalability can be achieved using Hierarchical Master/Worker-based B&B pushing the limits of the traditional Master/Worker paradigm. However, the resources offered by grids are most of the time unreliable, volatile, and heterogeneous. Therefore, they must take into account fault tolerance. In this paper, we present FTH-B&B, a fault tolerant hierarchical B&B, in order to deal with the fault tolerance issue. It is composed of several fault tolerant Master/Worker-based sub-B&Bs organized hierarchically into groups and perform independently fault tolerant mechanism. Beside, a fault recovery mechanism is introduced to recover and avoid redundant exploration of sub-problems in case of failures. In addition, we propose a mechanism to maintain the hierarchy safe and balanced during the lifetime of the algorithm. Our algorithm is applied to the Flow-Shop scheduling problem (FSP) and implemented on top of the ProActive grid middleware. It has been promisingly experimented on the Grid’5000 French nation-wide grid and shows its ability to remain efficient even in presence of failures.

Keywords—Fault tolerance; Hierarchical Master-Worker; Grid Computing; Parallel Branch and Bound; Large scale Experiments;

I. INTRODUCTION

Solving exactly combinatorial optimization problems (COPs) exactly consists in finding the optimal solution among a large set of feasible solutions (search space). However, these problems are NP-hard [13], they are CPU time-intensive and require a huge amount of computing resources to be solved optimally. Branch and Bound (B&B) algorithms are the best known method for an exact resolution of COPs. They perform an implicit enumeration of all the search space instead of exhaustive one. Based on a pruning technique, they reduce considerably the computation time required to explore the whole search space. However, such a technique is not sufficient for very large problem instances. Therefore, high performance computing such as grid-based parallel computing is required.

Nowadays, computational grids offer a huge amount of computing resources that can provide the power required by parallel B&B algorithms to solve large COP instances. Nevertheless, the computing resources are highly unreliable, volatile, and heterogeneous. Therefore, the traditional parallel B&B algorithms must be rethought to meet these characteristics. The presence of node failures in the computational grid requires dealing with fault tolerance issue to correctly perform the computation and to ensure reliable execution. Additionally, parallel algorithms must be adapted to deal with the large scale environments where a huge amount of spawned work units must be executed and/or communicated. Consequently, bottlenecks can be created on some parts of the network or some computational resources.

Traditional Master/Worker-based parallel B&B (MW-B&B) algorithms have proven their inefficacy [6][5] to deal with the scalability issue. In such algorithms, a single master decomposes the initial problem to be solved into multiple smaller sub-problems and distributes them among multiple workers. The workers, on their side, perform the exploration of the different sub-problems. However, this approach is strongly limited regarding scalability in large scale environments. Indeed, the central master process is subject to bottlenecks caused by the large number of requests submitted by the different workers. A high scalability can be reached using a hierarchical design. Indeed, using a Hierarchical Master/Worker (HMW) design allows pushing the limits of MW-B&B.

However, designing such hierarchical parallel B&B algorithms is not straightforward since they must also deal with the unreliability of the computing resources. Fault tolerance can be achieved at two levels: application-level or middleware-level. Several middlewares offer fault tolerant mechanisms to reliable execution of applications: ProActive [2][7][10], XtremWeb [4], and Condor [1][15][22]. Nevertheless, they are generally costly in terms of execution time and slow down the application at the expense of the execution time. The second strategy consists in introducing the fault tolerance inside the algorithms [20][21][18][12][16][17][11].

In this paper, we propose FTH-B&B (Fault Tolerant
Hierarchical Branch and Bound) designed for unreliable architectures. To avoid additional costs of middleware fault tolerant mechanisms, we focus on application-level ones and introduce a distributed fault tolerance mechanism. Indeed, FTH-B&B is composed of several fault tolerant MW-B&Bs organized hierarchically into groups forming a hierarchy. A fault recovery mechanism is introduced to avoid the loss of work units and to gain in terms of execution time. In this sense, work units are stored so that one can recover at each instant the sub-problems assigned to dead processes using a 3-phase communication mechanism (presented in Section IV-A). Moreover, our approach, ensures to maintain the hierarchy safe and balanced during the lifetime of the algorithm in order to have a valid functioning. Finally, a relaunching from scratch is ensured by a distributed checkpointing mechanism.

FTH-B&B has been applied to Flow-Shop scheduling problem (FSP) and developed on top of the ProActive middleware but without using its fault tolerance features to avoid slowing down the algorithm. It has been experimented on the Grid’5000 French nation-wide grid [3]. The experiments show the ability of FTH-B&B to deal with the fault tolerance of grids. Indeed, the overall efficiency of the algorithms reaches easily 98.90% using the fault tolerance mechanism. Additionally, the recovered and rescheduled sub-problems participate in the gain in terms of execution time. Finally, our approach has proved its ability to maintain and to balance the hierarchy during its lifetime.

The rest of this paper is organized as follows: Section 2 presents an overview of B&B algorithms and their parallelization. Section 3 presents existing fault tolerant mechanisms of parallel B&B algorithms. The proposed FTH-B&B describing the work management, the fault recovery, the maintenance of the hierarchy, and the distributed checkpointing are detailed in Section 4. Section 5 presents the experimental results using the Grid’5000 real nation-wide grid infrastructure. In Section 6, the concluding remarks of the paper are drawn and some perspectives are presented.

II. Branch and Bound

Branch-and-Bound (B&B) algorithms are well-known techniques for an exact resolution of COPs. They make an implicit enumeration of all solutions of the search space instead of an exhaustive enumeration which is obviously impractical due to the exponential growth of the potential solutions. B&B algorithms are characterized by four operators: branching, bounding, selection and elimination. Using the branching operator, the search space of a given problem is partitioned into a set of smaller sub-problems. The bounding operator is used to compute the lower bound of the optimal solution. When a new upper bound is identified, it is compared to the actual lower bound in order to decide whether or not it is necessary to branch the sub-problem. The elimination operator allows to identify the nodes which are not likely to lead to the optimal solution and eliminates them. Sub-problems are explored according to a given selection strategy which can be: depth-first search, breadth-first search, best-bound search, etc. A serial implementation of the algorithm consists of a sequential execution of the above four operations.

The sub-trees generated when executing a B&B algorithm can be easily explored independently because the only global information in the algorithm is the value of the upper bound. Its parallelization may be designed according to the architecture of the processing machine, synchronization, granularity of generated tasks, communication between different processes, and the number of computing processors [14]. Gendron et al. [14] classified the parallelization strategies into three classes according to the degree of parallelization: parallelism of type 1 introduces parallelism when performing the operations (generally the bounding operation) on generated sub-problems (e.g. executing the bounding operation in parallel for each sub-problem). In type 2 the search tree is built in parallel by performing operations on several sub-problems simultaneously. The type 3 also implies the building of several trees in parallel. The information generated when building a given tree can be used for the construction of another. Therefore, the tree is explored concurrently.

The processes involved in the computation of the parallel algorithm select their tasks from a work pool which is a memory where the processes select from and store in their work units (generated and not yet explored sub-problems) [14]. Two types of work pool can be distinguished: single work pool and multiple work pool. The first type is more adequate for applications based on the master-worker paradigm. In the second type, there are several memory locations where processes find and store their work units. Multiple work pool is divided into three sub-classes: collegial, grouped and mixed. In a collegial algorithm, each work pool is associated with exactly one process. In a grouped organization, processes are partitioned into groups, and each group of processes has its own work pool. In a mixed one, each process has its associated work pool, but there is also a global work pool, shared by all processes.

III. Related Work

No loss of data is tolerated in exact methods designed to COPs. A loss of one or several sub-problem(s) can cause the loss of one or several optimal solutions. Therefore, each parallel B&B designed to be executed in an unreliable environment must take into account the fault tolerance issue. Fault tolerance can be achieved at the middleware level [2][4][1] or at the application level [20][21][18] [12][16][17][11]. The fault tolerance mechanisms introduced at the middleware level cause additional delays to the execution time of the algorithms. For example the ProActive middleware proposes a fault tolerance mechanism using a
replication of the process involved in the computing. This forces the developers to reserve nodes especially for process replication which leads to lose in terms of execution power. In our case, we focus on fault tolerance at the application-level which allows us to gain in terms of execution power. In the following, we present the state-of-the-art approaches for application-level fault tolerance.

Iamnitchi et al [18] have proposed a fully decentralized parallel B&B algorithm. Each process maintains its local work pool and sends requests to others when it is empty. The process receiving a work request and having enough work in its pool, sends a part of its work to the requester. The fault tolerant mechanism does not attempt to detect failures of computers and restore their data, but rather focuses on detecting not completed problems knowing completed ones. The approach is based on the fact that the generated sub-problems of a B&B are tree nodes. Therefore, sub-problems are represented uniquely by their position in the tree. Given a set of nodes of the tree, its complement can be easily found.

Every process maintains a list of new locally completed sub-problems and a table of the completed problems. When a problem is completed, it is included in the local list. After a period of time or after a fixed number of sub-problems, the list is sent to $m$ other processes, selected randomly, as a work report message. When a process receives a work report, it stores it. When a process runs out of work, it chooses an uncompleted problem and solves it.

Finkel et al. [12] have proposed DIB (Distributed Implementation of Backtracking) which is a fault tolerant hierarchical algorithm designed for tree algorithms like B&B. Its failure recovery mechanism is based on keeping track of machines responsible of each unsolved problem. Each machine memorizes the problems for which it is responsible, as well as the machines to which it sends problems or from which it receives problems. The completion of a problem is reported to the problem source machine. Hence, each machine can determine whether the work for which it is responsible is still unsolved, and can redo it in case of failure.

Z. Dai et al. [11] have proposed a single-level hierarchical master-worker designed for divisible tasks which is similar to divide and conquer paradigm. In their framework, a main master only communicates with some sub-masters, and each sub-master manages a set of workers. Nodes of the hierarchy are organized into groups. Both the middleware-level and application-level fault tolerance mechanisms are used. The middleware-level mechanism ensured by ProActive [2] is applied only to the main-master and the authors do not specify if they also use it to sub-masters and workers. They are confronted to orphan workers when a sub-master fails. They use two techniques to circumvent this issue: the election of a sub-master from the remaining sub-workers and the use of redundant sub-workers. The elected sub-workers are a replica of sub-masters, when a sub-master fails they are used to generate a new sub-master.

Mezmaz et al. [20][21] have proposed B&B@Grid for large scale B&B algorithm using the master-worker paradigm. The authors rethought the representation of the search space and then minimized the quantity of flowing information in the network in order to alleviate the master process. Their approach is based on an efficient coding of work units. A list of sub-problems is represented by a unique interval defined by only two integers. Fold and unfold operators are defined to deduce a search sub-space from an interval and vice versa. Therefore, the transferred and stored information in the grid is an interval instead of a list of sub-problems. In order to push the limits of B&B@Grid in terms of scalability, Djamai et al. ?? designed a pure P2P approach for the algorithm. It provides fully distributed algorithms to deal with B&B mechanisms like work sharing, best upper bound sharing and termination detection.

Fault tolerance, is ensured by a checkpointing mechanism. The optimal solution and the not yet explored sub-problems are stored as intervals in a backup file. When the master fails, the file is checked and then the not yet explored sub-problems are deduced using the folding operator. Workers update the intervals regularly and inform the master of any new solution. When a worker fails, the last copy of its interval is either fully allocated to another B&B process or shared between several B&B processes.

IV. THE FTH-B&B ALGORITHM

The FTH-B&B we propose is a fault tolerant parallel B&B algorithm based on the Hierarchical Master/Worker paradigm in order to deal with the fault tolerance issue while ensuring the scalability issue in large scale environments using the hierarchical design. The proposed approach is composed of several fault tolerant Master/Worker-based sub-B&B algorithms (see Figure IV), where inside each sub-B&B one master manages several workers and performs failure recovery. The sub-B&Bs are launched in parallel and
act independently on different sub-trees. They are organized hierarchically in several levels. The root node and the inner nodes of the hierarchy designate masters and the leaves designate workers. The masters perform a parallel recursive branching in order to decrease the size of sub-problems until reaching sufficiently fine-grained sub-problems which can be explored sequentially by the workers. In addition, they locally store the branched and assigned sub-problems for further rescheduling. Each sub-B&B is composed of one master and a group of workers that can also be masters for another sub-B&B at the lower level. Each sub-B&B is developed using the framework (P2PBB) presented in [8] where one master manages a dynamic group of communicating workers. Therefore, the algorithm is composed of communicating groups of fault tolerant Master/Worker-based sub-B&B processes.

FTH-B&B is designed to run on computational Grids offering huge amount of computing resources which are highly unreliable. Therefore, for a valid execution of the algorithm, any failure must be detected and handled. The fault detection is the responsibility of all processes of the algorithm. Both masters and workers are responsible to detect failures of the process they depend on or which depends on them. Masters send heartbeats to their children every Heartbeat Period \( HP \). If a worker is dead or suspected to have failed, it is removed from its list of children. The not yet explored part of its sub-problem is saved and rescheduled to another free safe worker (see Section IV-A for more details). Workers also send heartbeats to their masters every \( HP \times G \), \( G \) being the size of the group forming one sub-B&B. The period of heartbeats is increased \( G \) times to avoid flooding their masters by their heartbeats.

In order to minimize communication overhead, masters do not inform their children about any worker failure. In the same sub-B&B group, workers do not heartbeat their neighbors but rather they detect failures when they broadcast their upper-bounds. When that happens, they remove the failed worker from their list of neighbors, so that they will never send them messages.

When executing FTH-B&B in a faulty environment, one must take into account three important points. First, one must ensure fault recovery to avoid the loss of work units and to gain in terms of execution time in minimizing the redundant work. Second, one must ensure to maintain the same topology during the lifetime of the algorithm in order to have a valid functioning of FTH-B&B. Third, one must ensure an efficient relaunching from scratch of a great number of FTH-B&B processes.

### A. Work management with task recovery

Each process has its local work pool represented by a list of active sub-problems. An active sub-problem is generated during the branching of the sub-tree received from the master in the upper level but not yet visited. It covers all sub-problems that can be explored from each element of the list. A collegial strategy is considered since multiple work pools are used and each process has its own work pool. During the search, the local pool evolves continuously and when it is empty, the process sends a work request to its master. A master receiving such a request consults its local work pool and asks its upper master if its work pool is empty.

Since no loss of work is tolerated in exact resolution of COPs, masters manage a list of assigned sub-problems which is a mapping between the assigned sub-problem and the process assigned to it. A sub-problem has a unique identifier in the local work pool the parent problem. When a failure is detected, a part of the sub-problem is rescheduled to another safe worker. To avoid redundant responses, only the first result is taken into account.

In order to optimize the overall execution time and avoiding redundant exploration of sub-problems, when facing failures, the masters also manage a list of branched sub-problems \( LBS \). It represents the sub-problems being explored by their grandchildren (see Figure 2). It is updated each time a grandchild finishes the exploration of its assigned sub-problem. When the process exploring a sub-problem fails, the master identifies the sub-problem assigned to it and the branched sub-problems in \( LBS \) and reschedules only the not yet explored part. For example, in Figure 2, when \( B \) fails, \( A \) does not reschedule all the sub-problem \( P_B \) but only the sub-problems \( P_{G}, P_{H}, P_{I} \), and \( P_{J} \) to the newly connected node \( C \) because \( P_{E} \) and \( P_{F} \) are already explored. The size of \( LBS \) does not affect the masters because it is static and depends only on the size of the sub-problem and the used branching method. Moreover, it is removed when the parent sub-problem is totally explored.

The task distribution is performed respecting a fault recovery mechanism using the 3-phase communication mechanism between the current master, its children and its grandchildren (see the sequence diagram in Figure 3).

- **Phase 1**: It allows a master to assign problems to its children and to receive back the branched sub-problems. In the sequence diagram (see Phase 1 in Figure 3), \( M_i \) assigns a problem \( P_{ij} \) to its child
Phase 1: After the initial branching and assignments, the 3-phase mechanism starts with the maintenance of the hierarchy by the master, denoted by $M_{ij}$, which informs its children about the updated sub-problems.

Phase 2: Each child, represented by $C_{ijk}$, updates its sub-problem exploration and informs its parent $M_i$ about the updated sub-problems. This phase also involves the maintenance of sub-problems already explored by the grandchildren.

Phase 3: The master reschedules the not-yet explored sub-problems to safe processes. In case of worker failure, the master in charge of the failed worker will reschedule its sub-problem to another safe worker. This phase involves the reconnection of disconnected branches and the continuation of the algorithm.

B. Maintenance of the hierarchy

It is important to maintain the same topology from the beginning to the end of the execution of the algorithm in order to have a reliable execution. Indeed, all communication and work management are based on the used topology (in our case the hierarchy). Additionally, failures can isolate some parts of processes from the rest of the hierarchy leading to loss of computing power and/or data. Moreover, at each instant, the hierarchy must be balanced.

Masters and workers are both subject to failures. A worker failure has not a great impact because it is localized in a leaf of the hierarchy. In fact, no other process depends on it and its task can be partially rescheduled by its parent (see Section IV-A). However, a master failure can isolate some parts of the hierarchy because the inner masters represent intermediary links. Indeed, when an inner master fails, the sub-B&B it represents crashes and the link between its descendants and the rest of the hierarchy is lost. Therefore, orphan branches may be created and can cause a dysfunction in the global algorithm. The following presents how to ensure no loss of data (work units), in case of worker failure, and how to maintain the hierarchy, in case of inner master failure.

In the case of worker failure, the master in charge of it detects its failure and reschedules the not-yet explored part of its task to another safe worker. Its neighbors will detect its failure in the next upper-bound communication. When a master fails, the hierarchy is reconstituted with the creation of new links between the descendants and the ascendants of the failed master. In the following, every process $P_i$ holds a list of all its ascendants $A_i$, which contains at most $\log_k(N)$ elements, $N$ being the size of the computational pool and $k$ the size of a sub-B&B group. Two rebuilding strategies are considered: Simple connection to ascendants and Balanced hierarchy.
1) Simple connection to ascendants: In this strategy, when a master \( M_i \) fails, all its children \( C_i[1..g] \) connect to one of its closest safe ancestors \( A_i[1..(l-1)] \), \( l \) is the current level (see Figure 4). This strategy is simple to implement and to manage. Nevertheless, its main drawback is that the closest safe master rapidly becomes a bottleneck when it is faced to several failures among its children. The safe master receives an exponential number of connections in a short period of time. Indeed, if \( f \) masters fail, it receives \( f^k \) connections from its descendants, \( k \) is the size of sub-B&B groups. Therefore, the FTH-B&B rapidly converges to a simple MW-B&B over time as the number of failed masters increases.

2) Balanced hierarchy: In this strategy, when a master crashes, orphan processes migrate to another safe sub-B&B. Their migration is done according to a migration algorithm which assigns new orphan processes to a safe non-full sub-B&B. A sub-B&B is full if the number of processes composing it reaches the threshold group size a master can manage. The migration algorithm purpose is to avoid the convergence of FTH-B&B to a simple MW-B&B and to maintain the hierarchy balanced at each instant. That is done with the readjustment of the sub-B&Bs group sizes at each master failure. Orphan processes connect to their closest ascendant. To maintain a balanced hierarchy, the closest safe master holds the orphan processes only when the group it manages is not full. Otherwise, it dispatches them over its safe children (see Figure 5). Therefore, the hierarchy of FTH-B&B changes over the time as soon as masters fail. That avoids creating new bottlenecks after several failures and no risk to convergence during the lifetime of the system. Moreover, at each instant the hierarchy is balanced whatever the number of failures since the new orphan processes are dispatched over all the hierarchy.

The main drawback of this strategy is the overhead caused by the management of the hierarchy which relatively degrades the efficiency in terms of execution time. Additionally, the safe masters must manage the task of the both failed and orphan processes.

C. Distributed check-pointing

In a simple Master/Worker-based B&B, the reliability can be achieved through checkpointing operation, but this approach assumes that there exists at least one reliable process (the master). In our approach, there exists many levels of checkpointing. Each level of masters in the hierarchy perform distributed asynchronous checkpointing operation independently from others. That makes the whole algorithm more reliable although some inner masters fail. Each sub-B&B has its own back-up file managed by the sub-B&B master. Therefore, at the level of each sub-B&B, a simple fault tolerant Master/Worker-based B&B process is launched, where the master saves periodically (or in asynchronous way) the performed sub-problems into its local back-up file. The master saves a sequence of not yet explored branches which are represented with a vector of sub-solutions. When a master fails, the new considered safe master checks the back-up file, it reconstitutes the not yet explored sub-problems, and then the exploration process carries on the last valid state of the sub-B&B. The global checkpointing is based on these simple sub-B&B checkpointing operations which are performed independently.

The main drawback of this approach is the large number of generated backup files when the number of launched sub-B&Bs increases. That can degrade the overall performance of the algorithm. In fact, when several masters are launched on the same grid cluster the multiple disk accesses can slow down the cluster. One can remedy to this drawback by uniformly launching sub-B&Bs on different grid clusters, or by limiting the number of masters which can perform checkpointing. Therefore, two types of masters are distinguished: masters which perform checkpointing (active masters) and those which do not (passive masters). The active masters are situated at the upper levels of the hierarchy and passive at the lower levels. We can decide dynamically if a master can or not participate in the checkpointing process.

V. PERFORMANCE EVALUATION

FTH-B&B has been experimented on the Flow-Shop scheduling problem (FSP) which consists in scheduling \( n \) jobs on \( m \) machines with respect to two constraints: (a) any two jobs cannot be assigned simultaneously to the same machine; (b) all the jobs are scheduled in the same order on all the machines. The cost function to be optimized is the total completion time i.e. \( \text{makespan} \) (\( C_{Max} \)) cost function. The objective is to find the optimal solution which minimizes the \( \text{makespan} \). The root of the tree generated by the B&B algorithm represents a configuration where no job is assigned to any machine. An inner tree node with depth \( d \)
has a partial solution with \( d \) assigned jobs. The effectiveness of B&B algorithms resides in the use of a good estimation of the optimal solution. We have opted for the best known lower bound for the FSP proposed by Lenstra et al. [19].

FTH-B&B has been implemented on top of the ProActive middleware [2][7][10]. ProActive is an open source Java library aiming at simplifying the programming of multi-threaded, parallel and distributed applications for Grids, multi-core systems, clusters and data-centers. It allows concurrent and parallel programming and offers distributed and asynchronous communication, a deployment framework, and fault tolerance features. However, FTH-B&B does not use any of the ProActive’s fault tolerance features to avoid slowing it down.

We experimented our approach on Grid’5000 which is composed of more than 5000 processor cores geographically distributed among 9 sites. In all the reported results FTH-B&B is experimented at large scale. Indeed, between 1900 and 8900 FTH-B&B processes are deployed in the different experiments. In order to obtain deeper hierarchy, multiple processes are launched on the same processor.

In the following, we experimentally evaluate the ability of FTH-B&B to deal with the fault tolerance issue in large scale unreliable environments. Indeed, we measure the efficiency of FTH-B&B which is the effective CPU time taken by the workers solving their tasks taking into account fault tolerance (the time of storing and updating sub-problems). We also evaluate the benefit of the task recovery mechanism on the efficiency of the algorithm in terms of the number of explored sub-problems during 40 minutes of execution solving the FSP instances Ta021 - Ta027 described above using 1120 grid nodes (therefore, 112 masters and 1008 workers). Failures are obtained killing processes chosen uniformly during their lifetime and the hierarchy is rebuilt at each instant using the balanced hierarchy approach. The table shows respectively, the percentage of dead processes, the number of rescheduled sub-problems (redundant work) and the runtime of task recovery mechanism and without using it, and finally the speedup between the two strategies.

The task recovery mechanism adopted by FTH-B&B aims at minimizing the redundant work (rescheduled sub-problems) when the processes handling them fail. Table II represents a comparison between two strategies of work management: with and without task recovery in presence of failures. We measure the number of rescheduled sub-problems during 40 minutes of execution solving the FSP instances Ta021 - Ta027 described above using 1120 grid nodes (therefore, 112 masters and 1008 workers). Failures are obtained killing processes chosen uniformly during their lifetime and the hierarchy is rebuilt at each instant using the balanced hierarchy approach. The table shows respectively, the percentage of dead processes, the number of rescheduled sub-problems (redundant work) using the 3-phase fault recovery mechanism and without using it, and finally the speedup between the two strategies.

On average 54% of the processes are killed during the exploration process. We note that without using the 3-phase mechanism for task recovery, the masters reschedule more sub-problems than when they use it. Therefore, more

<table>
<thead>
<tr>
<th>Bench</th>
<th>Execution Time ( T_{\text{Exec}} )</th>
<th>3-Phase Time ( T_{3-\text{Phase}} )</th>
<th>Delay Time ( T_{\text{Delay}} )</th>
<th>Efficiency ( R )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ta021</td>
<td>46996.603</td>
<td>120.507 (0.25%)</td>
<td>367.501</td>
<td>98.97%</td>
</tr>
<tr>
<td>Ta022</td>
<td>41843.986</td>
<td>126.265 (0.40%)</td>
<td>401.372</td>
<td>98.31%</td>
</tr>
<tr>
<td>Ta023</td>
<td>47180.395</td>
<td>130.493 (0.27%)</td>
<td>381.944</td>
<td>99.02%</td>
</tr>
<tr>
<td>Ta024</td>
<td>57050.001</td>
<td>128.745 (0.22%)</td>
<td>384.204</td>
<td>99.10%</td>
</tr>
<tr>
<td>Ta025</td>
<td>60370.203</td>
<td>129.536 (0.21%)</td>
<td>421.354</td>
<td>99.09%</td>
</tr>
<tr>
<td>Ta026</td>
<td>50753.190</td>
<td>117.435 (0.23%)</td>
<td>423.435</td>
<td>98.94%</td>
</tr>
<tr>
<td>Ta027</td>
<td>54855.053</td>
<td>128.871 (0.23%)</td>
<td>362.219</td>
<td>99.11%</td>
</tr>
<tr>
<td>Average</td>
<td>508.404.055</td>
<td>125.187 (0.25%)</td>
<td>385.206</td>
<td>98.92%</td>
</tr>
</tbody>
</table>

Table I

<table>
<thead>
<tr>
<th>Bench</th>
<th>(% Dead processes)</th>
<th>Redundant work (% of time)</th>
<th>Speedup (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ta021</td>
<td>50.89%</td>
<td>2736535518</td>
<td>10.48</td>
</tr>
<tr>
<td>Ta022</td>
<td>52.76%</td>
<td>2056609549</td>
<td>09.56</td>
</tr>
<tr>
<td>Ta023</td>
<td>42.58%</td>
<td>5550936476</td>
<td>05.76</td>
</tr>
<tr>
<td>Ta024</td>
<td>58.39%</td>
<td>4235792214</td>
<td>04.34</td>
</tr>
<tr>
<td>Ta025</td>
<td>53.66%</td>
<td>7049753500</td>
<td>07.07</td>
</tr>
<tr>
<td>Ta026</td>
<td>50.98%</td>
<td>5543751206</td>
<td>05.84</td>
</tr>
<tr>
<td>Ta027</td>
<td>51.16%</td>
<td>1198024644</td>
<td>10.48</td>
</tr>
<tr>
<td>Average</td>
<td>54.89%</td>
<td>7642972397</td>
<td>6.36</td>
</tr>
</tbody>
</table>

Table II
redundant work is done on all the tested instances, slowing down the algorithm. The speedup between the two strategies is then calculated and shows that on average, it is equal to 6. That means that when using the task recovery mechanism, FTH-B&B is 6 times more rapid than another approach which does not adopt a task recovery mechanism in an unreliable environment.

As mentioned in Section IV-B, it is important to maintain the hierarchy balanced during the lifetime of the algorithm in order to obtain a valid execution and to avoid isolating some parts of the hierarchy when inner masters are subject to failures. In the following, we experiment the two proposed approaches in this sense: Balanced hierarchy and the simple connection to ascendants one. To evaluate the robustness of the hierarchy in presence of failures, we record the degree of the different masters forming the hierarchy.

Figure 6 shows the degrees recorded on the four most loaded masters in the algorithm when using the simple connection to ascendants approach. The x-axis represents the percentage of the processes having failed in time in seconds and the y-axis determines the degrees of the masters. First, we note clearly that the degrees of the masters increase over time (with the number of dead processes) slowing them down. Second, the root master process rapidly becomes bottleneck passing from a degree equal to 10 at the beginning of the execution, to 1368 at the end. Hence, the algorithm rapidly converges to a classical Master/Worker paradigm.

The curve in Figure 7 shows the average degrees of the different masters sorted by their levels in the hierarchy, when using the balanced hierarchy approach. To obtain a deeper hierarchy, the size of a group is fixed to 10 and the number of launched processes on the same processor is duplicated to acquire more processes. In order to evaluate the robustness of the approach and its behavior in extreme situations, up to 8 masters are killed every 1 second. Second, we note that masters of the hierarchy resists to failures and do not become bottlenecks during all the experiment. Finally, the average degree decreases softly for the three first levels and more rapidly for the fourth one. That can be explained as follows: at the beginning, all sub-B&Bs are initially full because of the balanced construction of the hierarchy. After the first failures, no master can acquire orphan processes. Therefore, it uniformly dispatches them among its children, and so on until achieving the leaves of the hierarchy (workers). These workers change their behavior and become masters and each one has only the new migrated master as a unique child (degree=1), the fact that influences the overall degree of the masters in the down levels of the hierarchy. After many failures, the new masters acquire more and more migrated orphan processes. Therefore, the size of sub-B&B groups converges to the maximum size of a group and then the global degree converges to its original value.

VI. CONCLUSIONS

Solving combinatorial optimization problems exactly using parallel B&B algorithms requires a huge amount of computing resources. That can be achieved with their execution in large scale on computational grids. The scalability can be ensured using Hierarchical Master/Worker pushing the limits of the traditional Master/Worker paradigm. However, the resources offered by such grids are volatile and highly unreliable. Therefore, fault tolerance must be taken into account to ensure no loss of data during the execution of the algorithm which is intolerable in exact solving of COPs.

In this paper, we have proposed FTH-B&B (Fault Tolerant Hierarchical B&B algorithm) in order to deal with the fault tolerance issue of grids. The fault tolerance is introduced at the application level. Indeed, it is composed of several fault tolerant Master/Worker-based sub-B&Bs launched in parallel and organized into a hierarchy so that each sub-B&B is composed of a unique master managing several workers and performing a set of fault tolerant mechanisms locally. A 3-phase communication approach between a master, its children, and its grandchildren is adopted to distribute, store,
and recover work units in case of failures. This mechanism allows one to gain in terms of execution time because the master does not reschedule the entire sub-problems but only the not yet explored parts. The approach also proposes a mechanism to maintain the hierarchy safe and balanced during the lifetime of the algorithm. Finally, a distributed checkpointing mechanism is proposed in which each master performs its checkpointing independently from others.

We have implemented our approach on top of ProActive and we have experimented it on the Grid'5000 real nationwide grid environment. The different experiments demonstrate the ability of FTH-B&B to ensure fault tolerance while maintaining high execution efficiency. Indeed, the approach enables to achieve an efficiency of 98,92%. Moreover, the task recovery allows to gain on average 4,35% of the global execution time. Experiments also showed that the hierarchy can be maintained safe and balanced, thanks to the balanced hierarchy maintenance strategy.

As perspectives, we intend to develop an enhanced version which takes into account a best effort mechanism that allows to exploit the maximum number of resources on the used computational grid. Second, we plan to improve the realism of the simulated failures with the use of realistic model of failures based on real statistics of grid failures. FTH-B&B can also be improved with the use of a fully decentralized topology eliminating the weakness at the level of the root master.

REFERENCES