FTH-BB with Dynamic Sub Tasking for Large Scale Unreliable Environments

Ms. Mampi Bhowmik
Department of Computer Engineering
Late G. N. Sapkal College of Engineering
Nashik
m.mahi17@gmail.com

Prof. N. R. Wankhade
Department of Computer Engineering
Late G. N. Sapkal College of Engineering
Nashik
nileshrw_2000@yahoo.com

***Abstract*— Computational grids use Branch and Bound (BB) algorithm that requires a huge amount of computing resources. Most of existing grid-based BB algorithms are based on the Master-Worker paradigm. In traditional Master/Worker-based parallel BB (MWBB) algorithms, a single master decomposes the initial problem to be solved into multiple smaller sub problems and distributes them among multiple workers. The workers then 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. I thereby use FTH-BB with Dynamic Sub Tasking, a fault tolerant hierarchical BB. FTH-BB with Dynamic Sub Tasking is based on different mechanism that enables to efficiently build and maintain balanced the hierarchy, and to store and recover work units (sub-problems). 3-phase recovery mechanism is used to overcome the failure of any node or master. Moreover, this approach ensures to maintain a balanced and safe hierarchy during the lifetime of the algorithm. In proposed system, utilization of any node will be maximized by splitting the work units into random sized sub problems and assigning the sub problems to the nodes by analyzing their current utilization. In addition, an efficient restart of the application is ensured by a distributed check pointing mechanism in case of a global failure.**

***Index Terms*- Computational Grid, Branch and Bound, Fault Tolerant, Master Worker Paradigm**

I. INTRODUCTION

Grid computing provides the users the facility of large scale computational and data handling capabilities by employing large-scale sharing of resources. The importance of grid computing lies in the fact that it provides enormous computational power for users at a reduced cost. The grid is a heterogeneous system as compared to the traditional clusters or supercomputers. Computational grids are loose network of computers linked to perform grid computing. A large computational task is divided up among individual machines, which then run calculations in parallel and then return the results to the original computer. The individual machines that run the calculations are nodes in a network, which may belong to multiple administrative domains that are geographically distributed. Computational grids use computing resources that are highly unreliable, volatile, and heterogeneous. The heterogeneous and dynamic nature of grids requires balancing the workload in order to maximize the resource utilization and efficiency. Combinatorial optimization problems (COPs) are solved by finding the optimal solution from a large set of feasible solutions. However, these problems are NP-hard; they are CPU time intensive and require a huge amount of computing resources to be solved optimally.

II. PARALLEL BRANCH AND BOUND

These algorithms are used to find the optimal solution in solving combinatorial optimization problems. Branch-and-bound [2] (refer to as BB) is an intelligent search method often used for optimization problems. It decomposes the original problem into smaller disjoint sub problems, while reducing (pruning) the search space by recognizing unpromising problems before starting to solve them. A BB algorithm consists of a sequence of iterations in which it performs four basic operations over a list of problems which are called as active problems:

Branch: Splits a problem into a set of new sub problems .A problem that cannot be split (either because it has no solution or because a solution is found) is fathomed. A problem decomposed into new sub problems is branched.

Bound: Computes a lower bound value and the upper bound value on the optimal solution of sub problem. This bound value will be used by Select and Eliminate operations.

Select: Selection depend on bound values, if the lower bound of any sub problem is higher than the upper bound, it is not explored further.

Eliminate: Eliminate the problems that cannot lead to an optimal solution.

III. RELATED WORK

Finkel have proposed DIB (Distributed Implementation of Backtracking) algorithm [9] based on multiple pool collegial strategy and use depth first search approach for exploring the tree. Here a problem is divided into sub problems and assigned to available machines. Each machine maintains two tables, workGotten and workGiven. Also a heap is maintained with each machine with the list of sub problems yet to be completed. User can assign priorities with each sub problem which can lead to optimal solution. The sub problems are stored in the heap according to priority. If heap is empty, the machine sends request for work to other machines. A machine

always sends a fixed part of its work, usually half to the requesting machine. This minimizes the number of messages but by increasing the message size. However there is no mechanism to reduce the redundant work done.

Iamnitchi have proposed a fully decentralized parallel BB (Branch and Bound) algorithm [2] using a multiple pool collegial strategy. Each process maintains its local work pool and sends requests to others when this pool is empty. The process receiving a work request and having enough work in its pool sends a part of its work to the requester. Best known solution is circulated to each process using frequently sent messages and each process updates the value of best known solution. FT mechanism does not attempt to detect failures of processes and to restore their data, but rather focuses on detecting not yet completed problems knowing completed ones. Each 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 processing a fixed number of sub-problems, the list is sent to a set of 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. There is no central authority for quality control or operational management. There is no mechanism to reduce the redundant work done.

Dai proposed a single-level hierarchical M/W paradigm [8]. It uses the divide and conquer strategy for exploring the problems. A main master only communicates with some sub-masters, and each sub-master manages a set of workers using multiple pool collegial strategy. Both the middleware level and application-level FT mechanisms are used. The main drawback of this approach is the use of middleware-level FT and then the redundant processes which will replace the failed ones leading to the loss of computing power. Also the middleware FT mechanism is only for the main master and not for inner master. If an inner master fails, a new master is elected from among the sub workers. Moreover, no solution is proposed to minimize the redundant work.

Mezmaz have proposed BB Grid for large scale BB algorithm using the master-worker Paradigm [3]. A single work pool strategy is used for work distribution. Two main modifications done here are, to evaluate several optimal solutions instead of single one and to evaluate sub space according to several objectives. A list of active nodes is generated i.e. node created but not yet treated. Each active node covers a set of tree nodes. Each node in the tree is assigned a number. The numbers of the set of nodes covered by an active node forms an interval. Fold and unfold operators are used to establish relation between interval and active nodes. Fold operator deduces interval from list of active nodes and unfold operator vice versa.

Djamai [4], in order to overcome the limits of BB Grid by Mezmaz in terms of scalability, designed a pure P2P approach for the algorithm. It provides fully distributed algorithms to deal with BB mechanisms like work sharing; best upper bound sharing and termination detection. FT (fault tolerance) is ensured by a check pointing mechanism. However, this FT mechanism has been only applied to the original BB Grid and has not been extended to the P2P distributed version.

In this paper, we present an extension of these works: first, the fault recovery mechanisms are presented in detail. Second, Master Election and SCA are used to maintain the hierarchy following the tolerance of the root-master failure. Third, a distributed check pointing is proposed allowing reconstituting the unexplored sup-problems after a global failure. Lastly, assignment of sub problem to any node or master is done by checking its current utilization.

IV. PROPOSED SYSTEM

A. Hierarchical Design

The FTH-BB with dynamic sub tasking is based on Hierarchical Master Worker (HMW) paradigm [1]. Grid Server is the root and has a centralized control of the hierarchy. The hierarchical design deals with the scalability issue. It is composed of several Fault Tolerant Master/Worker sub BB. Each sub BB is having one master and several workers. The workers in a sub BB can act as a master for lower level sub BB. Each FTMW-BB performs parallel recursive branching of the task. Masters (inner nodes) then assign the tasks to worker by checking their utilization. Each master owns a single work pool. The Workers (Leaves) actually perform the sub tasks in parallel. The architecture is as shown in figure 1.

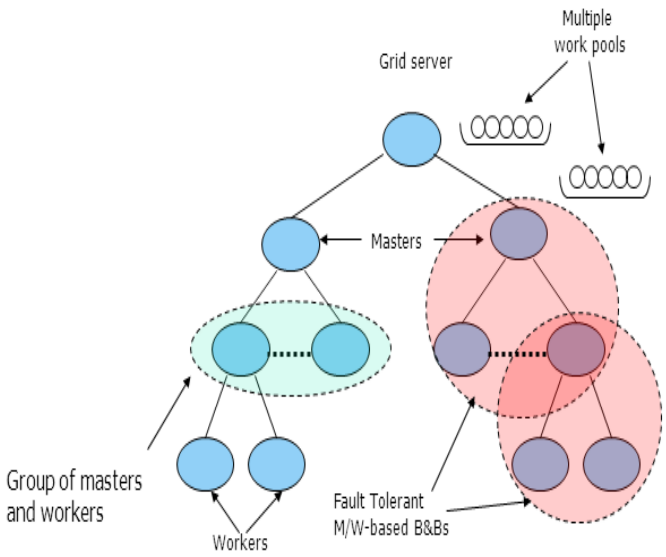

Fig 1: FTH BB Hierarchy Design

B. Concept of Heartbeat

The computing resources used in FTH-BB are unreliable. Any failure of the computing resources must be detected to ensure the connectivity of the grid and proper execution of the task. The heartbeats [1] in the system enables the Grid Server, Masters and Nodes to stay connected with each other and

ensure the availability of resources as shown in figure 2. The heartbeats are classified as follows:

Simple HB: Nodes heartbeat their master and master heartbeat their nodes to check if the node is alive.

Ack HB: Node sends Ack HB to its master to update the master about the work completed.

Control HB: Used initially by the grid server and masters to get the CPU and RAM utilization. After getting the current utilization the utilization table entries are updated at the masters and the server.

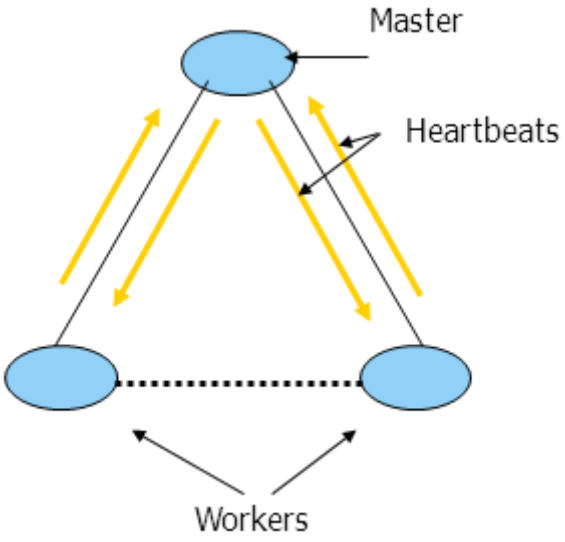

Fig 2: Heartbeat Mechanism

Each master and the grid server maintain two tables:

Utilization Table: This table stores the current utilization of RAM and CPU of each worker that is collected initially by sending Control HB. Table 1 shows an example of utilization table.

Cost Table: This table has predefined values for the amount of data that can be processed by any resource for a certain percentage of utilization. Table 2 shows an example of cost table.

Table 1: Utilization Table

CPU Utilization in %	RAM Utilization in %
50	50
50	60
20	10

Table 2: Cost Table

Utilization in %	Amount of Data
50	100 MB
60	80 MB
20	250 MB

C. Work Management

The client assigns task to the grid server (root). The server sends a control HB to the masters, which in turn sends the control HB to the workers. The worker sends the current utilization to the masters, with the help of which the utilization of the master is calculated and sent to the grid server. The server then divides the task into random sized sub tasks and assigns them to the master by checking its utilization. The masters further divides the task into further sub tasks, check the most promising options in the cost and utilization table and assigns the task to a worker. Each master and server maintains a list of branched sub problems LBS as shown in figure 3.

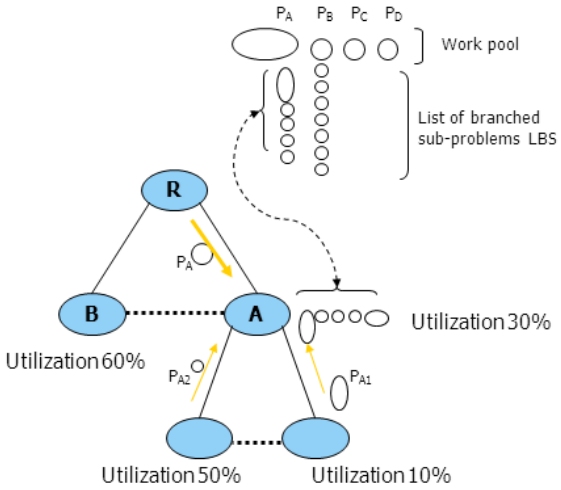

Fig 3: Distribution of Work

D. 3 Phase Recovery Mechanism

Both the masters and the nodes are vulnerable to failure. In case of failure of node, the task assigned to it must be rescheduled to a new worker. 3 phase recovery mechanism [1] guarantees the rescheduling of task in case of failure of worker or master without doing redundant work as shown in figure 4. It is divided into 3 phases as follows:

Phase 1 (between a master and its children): A master assigns a problem to its children. The child performs branching and sends back the branched sub-problems to the master.

Phase 2 (between a master, its children and its parent/server): Each time a worker finishes a sub problem, it updates the master which in turn updates its master or the server. The master and server know at any time the unexplored parts of a given problem.

Phase 3 (between a master and a new free node): When a process fails the parent of the failed process detects its failure and saves the unexplored part of its sub-problem. When a new safe process connects, the parent reschedules it the unexplored part of the sub-problem.

E. Maintenance of Hierarchy

Failure of any node in the hierarchy can lead to an unbalance hierarchy. The failure of a leaf node has not a great

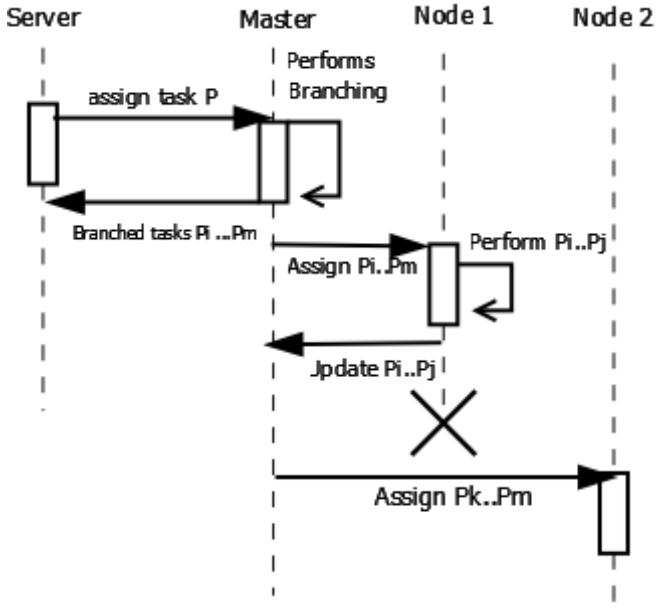

Fig 4: 3 Phase Recovery Mechanism

impact because it is located in a leaf of the hierarchy. No other process depends on it and its task can be partially rescheduled by its parent using the 3- phase mechanism. But a master failure can isolate some parts of the hierarchy because the inner masters represent intermediary links. When an inner master fails, the sub-BB it represents crashes and the link between its descendants and the rest of the hierarchy is lost. As a result orphan branches may be created leading to the failure of the algorithm. Hence, it is necessary to rebuild the hierarchy. Master Election ME algorithm [1] is used to maintain the hierarchy in case of failure shown in figure 5. When a master fails, the nodes under that master elect a new master among them using bully algorithm. Each node is having a unique identifier assigned to it. When a node p_i detects the failure of its master, it initiates an election by sending an election message to all its neighbours with higher identifier. If no process responds, p_i becomes the master and announces its success to the other nodes. If one of the nodes answers, it means that there is at least a safe node which can be a master, then p_i ends its election. When a node p_j receives an election message from a node p_i with a lower identifier, it answers and initiates a new election algorithm. A newly elected master considers all its neighbours as its children. The new master then connects to its closest safe ascendant using simple connection to ascendant (SCA) shown in figure 6. It is informed by its new parent about its neighbours. This method tolerates the failure of the server of the hierarchy. When the server fails, the masters of the first level select a master between them and this new selected master behaves as the server.

F. Distributed Check pointing

Distributed Check Pointing [1] makes the whole algorithm more reliable even if some inner masters or the server fail. Each master and server process uses a back-up file to store a

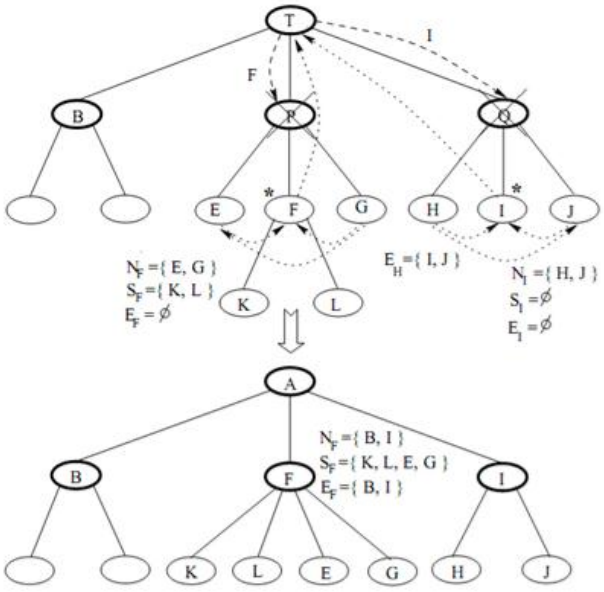

Fig 5: Master Election

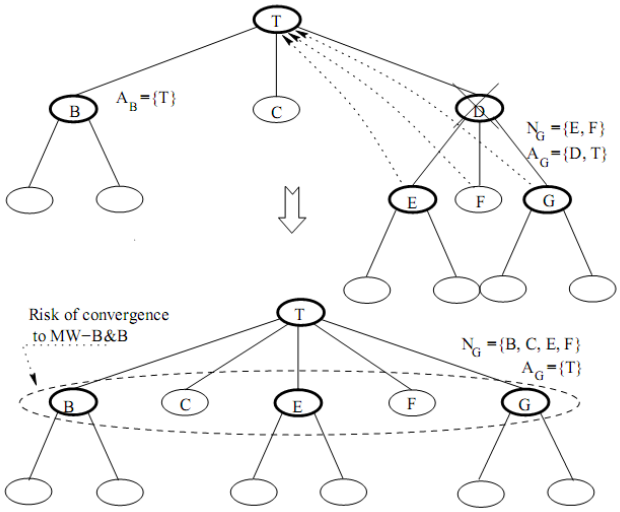

Fig 6: SCA Algorithm

sequence of unexplored branches. When a master fails, the newly designated master reads the back-up file, reconstitutes the locally unexplored sub-problems, and then the exploration process carries on from the last consistent local state of the sub-BB. The drawback is the large number of generated back up files as the number of master increase. Large number of generated back up files can degrade the performance of the system. To overcome this drawback we are restricting the number of masters which can participate in check pointing.

Only the level 1 masters and the server can perform check pointing operations. The masters who perform check pointing i.e. the level 1 master are called as active masters and others are called as passive masters. Each sub-MW-BB performs distributed check pointing independently. There is no global backup file and each master handles its own local backup file. Figure 7 explains the concept of distributed check pointing.

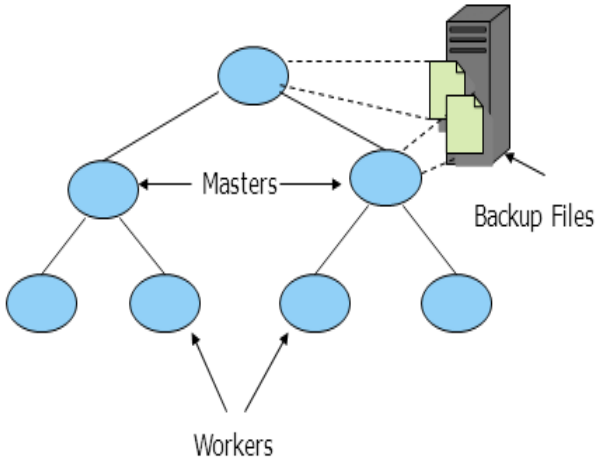

Fig 7: Distributed Check pointing

V. CONCLUSION

FTH-BB is an Application level Fault tolerant algorithm and hence no additional overheads are induced to the execution time of the algorithm. Several FT-MW-based BBs are launched hierarchically to address the issue of scalability. Each master performs FT mechanism. 3-phase recovery mechanism distribute, store, and recover work units in case of failures and also minimize redundant work. System can handle the failure of Server or any master by master election ME. Server and master assign the work by checking the current utilization of a system to increase the resource utilization and to get optimal solution. Also distributed check pointing mechanism is done by server and level 1 master to recover from failure.

REFERENCES

[1] A. Bendjoudi N. Melab and E-G. Talbi FTH-B&B: a Fault-Tolerant Hierarchical Branch and Bound for Large Scale Unreliable Environments IEEE TRANSACTIONS ON COMPUTERS

[2] Adriana Iamnitchi. A problem-specific fault-tolerance mechanism for asynchronous, distributed systems. In Proceedings of the International Conference on Parallel Processing 2000, pages 414, 2000.

[3] M. Mezmaz, N. Melab, and E-G. Talbi. A Grid-based Parallel Approach of the Multi-Objective Branch and Bound. In Fifteen Euromicro Conference on Parallel, Distributed and Network-based Processing, Naples, Italy, February. 7-9 2007. IEEE Computer Society Press

[4] M. Djamai, B. Derbel, and N. Melab. Distributed BB: A Pure Peerto- Peer Approach. In Proc. of 25th IEEE LSPP/IPDPS, Anchorage, (Alaska) USA, Mai 16th-20th 2011.

[5] A. Bendjoudi, N. Melab, and E-G. Talbi. Fault-Tolerant Mechanism for Hierarchical Branch and Bound Algorithm. IEEE International Symposium on Parallel and Distributed Processing Workshops and PhdForum (IPDPSW), 1806 1814, 2011.

[6] M. Mezmaz, N. Melab, and E-G. Talbi. A Grid-enabled Branch and Bound Algorithm for Solving Challenging Combinatorial Optimization Problems. In Proc. of 21th IEEE Intl. Parallel and Distributed Processing Symposium, Long Beach, California, March 26th -30th 2007.

[7] K. Aida, Y. Futakata, and T. Osumi. Parallel Branch and Bound Algorithm with the Hierarchical Master-worker Paradigm on the Grid (Grid). IPSJ Trans. on High Performance Computing Systems, 47(12):193206, 20060915.

[8] Z. Dai, F. Viale, X. Chi, D. Caromel, and Z. Lu. A Task-Based Fault-Tolerance Mechanism to Hierarchical Master/Worker with Divisible Tasks. High Performance Computing and Communications, 10th IEEE International Conference on, 0:672677, 2009

[9] R. Finkel and Udi Manber. Dib a distributed implementation of backtracking. ACM Trans. Program. Lang. Syst., 9(2):235256, 1987