ISSN 2320 -2602

International Journal of Advances in Computer Science and Technology (IJACST), Vol.3 , No.11, Pages : 01-05 (2014)
Special Issue of ICCAAC 2014 - Held on December 08,2014,Hyderbad, India

FTH with Dynamic Sub-tasking for Large Scale Unreliable Environments

Mampi Bhowmik[1], N. R. Wankhade[2]
[1]PG Student,L.G. N. Sapkal CoE, Nashik, India, m.mahi17@gmail.com
[2]HOD, Dept. of Computer Engineering, L.G. N. Sapkal CoE, Nashik, India, 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. But, 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 a 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.

Key words : Computational Grid, Fault Tolerant, Master Worker Paradigm.

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 [7]. However, these problems are NP-hard; they are CPU time intensive and require a huge amount of computing resources to be solved optimally.

RELATED WORK

Finkel have proposed DIB (Distributed Implementation of Backtracking) algorithm [9][10] 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][10] 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

ISSN 2320 -2602
International Journal of Advances in Computer Science and Technology (IJACST), Vol.3 , No.11, Pages : 01-05 (2014)
Special Issue of ICCAAC 2014 - Held on December 08,2014,Hyderbad, India

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] [5] [6]. 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 is used to maintain the hierarchy following the tolerance of the root-master failure. Lastly, assignment of sub problem to any node or master is done by checking its current utilization.

PROPOSED SYSTEM

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 Fig 1.

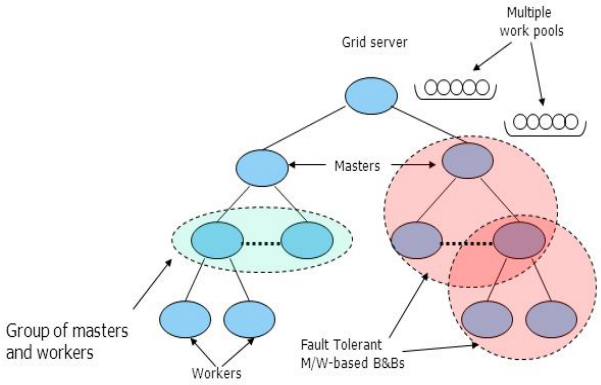

Fig 1: FTH BB Hierarchy Design

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 Fig 2. Node sends heartbeat to their master and master sends heartbeat to their nodes to check if the node is alive.

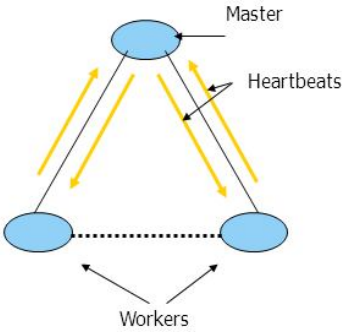

Fig 2: Heartbeat Mechanism

Work Management

The client assigns task to the grid server (root). The server sends a HB to the masters, which in turn sends the HB to the workers. The master finds out the current utilization to the workers, with the help of which the capacity of the worker is calculated. The server finds out the current utilization to the master, with the help of which the capacity of the master is calculated. 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 and assigns the task to a worker as shown in Fig 3.

ISSN 2320 -2602
International Journal of Advances in Computer Science and Technology (IJACST), Vol.3 , No.11, Pages : 01-05 (2014)
Special Issue of ICCAAC 2014 - Held on December 08,2014,Hyderbad, India

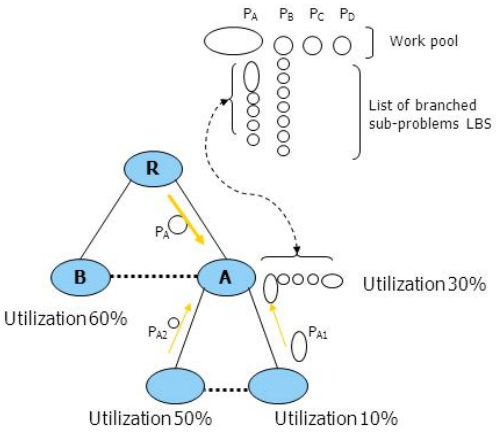

Fig 3: Distribution of Work

The division of task is done as follows:
Load = *f* (Memory available, CPU Load)

Suppose there are n number of nodes: 1, 2,…,n.

Load of Ni = Memory available at Ni + CPU Load of Ni
where i = 1, 2, … , n

Total Load (TL) = N1L1 + N2L2 +….+ NiLi
% Load of Ni = NiLi / TL

% Capacity of Ni = 100 - % Load of Ni

Number of nodes given to Ni for processing is:
Nodes Given to Ni = (% Capacity of Ni* Total number of nodes) *100

Three 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 Fig 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.

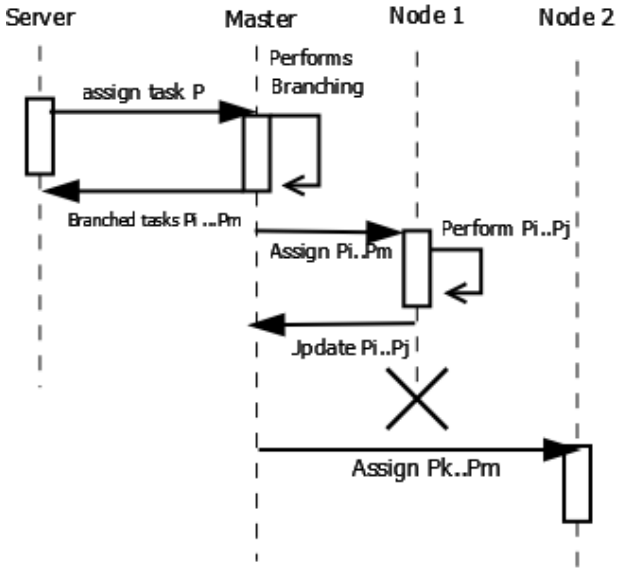

Fig 4: 3 Phase Recovery Mechanism

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 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 Fig 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 Fig 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.

ISSN 2320 -2602
International Journal of Advances in Computer Science and Technology (IJACST), Vol.3 , No.11, Pages : 01-05 (2014)
Special Issue of ICCAAC 2014 - Held on December 08,2014,Hyderbad, India

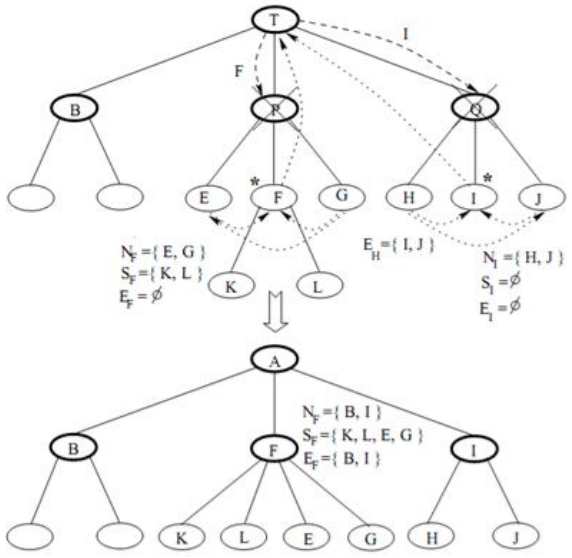

Fig 5: Master Election

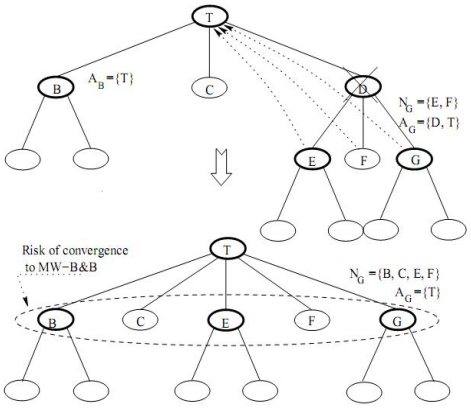

Fig 6: SCA Algorithm

ALGORITHM

1. Client assign task to grid server.
2. Grid server manages sub BB, each sub BB have one master.
3. Grid server and each master checks the availability of free resources of its subordinates by sending heartbeats.
4. Grid server distributes the task into random sized sub tasks ex. P1, P2 etc.
5. Grid server assigns the task to its intermediate masters who further divide the tasks into further sub tasks ex P1a, P1b, and P1c and so on.
6. Before assignment of task to a node, the current utilization of the node is checked and depending on the utilization master decides which sub problem to be assigned to the node.
7. If a node fails, master reschedules the task to other free node using 3 phase recovery.
8. If master fails, the task with highest identifier initiates an election algorithm ME for election of new master.
9. Hierarchy is maintained by readjustment of size of sub BB.
10. Nodes perform the task and return the results to master.
11. Master combines the result from different nodes and returns it to the server.
12. Server gives the result to the client.

RESULT ANALYSIS

FTH has been experimented on travelling salesman problem (TSP). The size of problem to be solved is considered as follows: (1) Small: 10*10 (2) Medium: 20*20, 30*30 (3) Large: 40*40, 50*50.

In the first experiment reported in Table 1, we evaluate the impact of the delay induced by the proposed FT mechanisms (the 3-phase recovery mechanism) on the performance of the algorithm. We calculate the ratio R between the effective execution time t_{Exec} and the idle time t_I recorded on the workers. The idle time includes the communication time t_C, the additional time of internal management t_M, the time masters take to perform the 3-phase recovery mechanism t_3 which includes: time of branching, time of storing sub-problems received from children and times of updating the sub-problems explored by the grandchildren.

Efficiency is calculated as:

$$R = (t_{Exec} / (t_{Exec} + t_3)) * 100$$

Table 1: Impact of FT

Instance Size	Execution Time t_{Exec} (ms)	3 Phase Time t_3 (ms)	Waiting Time t_M+t_C (ms)	Efficiency (%)
10*10	950	14	0	98.55
20*20	940	110	0	89.52
30*30	676	34	1	95.08
40*40	578	74	0	88.65
50*50	824	95	1	89.57
Average	793.6	65.4 (8.24%)	0.4 (0.05%)	92.27

Table 1 show that the use of the 3-phase recovery mechanism is not very costly in terms of execution time. Indeed, the masters and workers spend on average 8.24% of their total execution time for branching and recovering of failed processes. Moreover, the average waiting time of workers is negligible, only 0.05% of the total time. The parallel efficiency in the last column shows that the workers spend on average 92.27% of their time solving sub-problems. However, it varies from an instance size to another (98% for small instances versus 91% for large ones).

COMPARISION WITH EXISTING SYSTEM

Table 2 shows the comparison of proposed system with the existing systems.

Table 2: Comparison

Parameter	Existing System	Proposed System
Root Failure	Difficult to maintain hierarchy	Maintain Hierarchy using ME
Sub Problem Size	Fixed	Variable
Resource Utilization	Less	More
Redundant Work Done	Yes	Yes
FT Mechanism	Middleware and Application Level	Application Level

CONCLUSION

FTH-BB is an Application level Fault tolerant algorithm and hence not much 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 tries to minimize redundant work. System can handle the failure of Server or any master by master election ME. The node failure does not induce much overhead on the system, however master or server failure may induce some amount of overhead leading to increase in 3 phase time. Server and master assign the work by checking the current utilization of a system to increase the resource utilization and to get optimal solution.

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

[10] M. Bhowmik and N. R. Wankhade "fault tolerant hierarchical design in large scale unreliable environments: a review" presented at 3rd international conference on recent trends in engineering and technology, march 2014