EAI Endorsed Transactions
on Energy Web
Research Article EAI.EU

Resubmission Based Fault Tolerance Approach to Schedule Jobs in GRID Environment

Ravinder Ahuja[1,*] and Alisha Banga[2]

[1]Computer Science and Engineering department, Jaypee Institute of Information Technology Noida-201304, Uttar Pradesh, India
[2]Electrical and Electronics Engineering department, Satyug Darshan Institute of Engineering and Technology, Faridabad-121003, Haryana, India

Abstract

Grids are the type of distributed machines in which aggregation of resources is done to provide services. Grid environment needs to be fault tolerance so as to obtain maximum performance of the tasks that are being processed on it. A novel fault tolerance approach is designed by us. Fault tolerance approaches mainly lie into two categories, Replica Based approach, and Check-Pointing approach. The main limitation of Replica Based Approach (RBA) is its non-applicability to cost-based resources whereas Check-Pointing Approach (CPA) suffers from the inherent disadvantages of taking Check-Point which incur overhead and wastage of time. In our approach, the large task is divided into various subtasks on the basis of data flow and control flow dependencies. The experimental results show that the proposed approach is efficient than a Check-Point approach in terms of various parameters like the number of Grid lets successfully completed an average execution time of a task. GridSim ToolKit 4.0 is used for simulation.

Keywords: Check-Point Approach, Dependency Graph, Grid Computing, Job Scheduling, Replica-based Approach.

Received on 13 September 2019, accepted on 13 July 2019, published on 17 July 2019

doi: 10.4108//eai.11-6-2019.159603

*Corresponding author. Email:ahujaravinder022@gmail.com

1. Introduction

The grid is "A type of parallel and distributed system that enables the sharing, selection, and aggregation of geographically distributed autonomous and heterogeneous resources dynamically at runtime depending on their availability, capability, performance, cost and users' quality-of-service requirements"[1]. Grid computing enables aggregation and sharing of geographically distributed resources and data into a single virtual machine for solving the large-scale problems, which requires more computational power. The grid can be used for many applications like medical imaging and diagnosis, biometrics, satellite image processing. The grid has many design issues like scheduling, data management, resource management, security, load balancing and fault tolerance [2, 3]. Grid jobs are very large and many grid environments are most likely to fail, so fault management becomes more important to give desired Quality of Service (QoS) to grid user. Due to the large size of jobs, the cost and difficulty of finding and recovering from faults in Grid applications are higher than normal applications. In grid environment resources may enter and leave at any time which may cause a fault.

There are two approaches of fault tolerance in a grid environment: proactive and reactive [4, 5]. Proactive fault tolerance approach considers the failure of resources before scheduling jobs, like replica approach. On the other hand, the reactive mechanism takes appropriate action after the job failure, like check pointing approach. Our approach will prove several advantages over two main approaches, one is a failure of one subtask does not

affect the other independent tasks, only failed task need to be resubmitted not the complete task.
Section 2 contains literature review, section 3 contains proposed approach, section 4 contains simulation parameters, section 5 contains results and discussions, and section 6 contains conclusions and future scope.

2. Literature Survey

2.1. Replica-Based Approach

In this approach [6] multiple copies of the task are created, which increases the probability of success. If a replica fails then it does not restart and success depends upon success on other tasks. Obviously, it creates overhead in workload due to multiple executions of the task. If n copies of w workload then the total workload is wxn. But due to its good performance, it is used in many grid solutions.
Communication between replicas is important in this approach for that it uses TOM (Total order multicast) [7] protocol for communication. The grid is divided into clusters. Clusters are controlled by a head node called coordinator. Coordinators act as an interface between internal nodes and external nodes in other clusters. Group membership services are used to manage processes. The basic operations provided are joined, leave and, execute. Two primitives operations are used for communication send_multicast and receive_multicast. The number of replicas required can be calculated by using a fuzzy approach. Some algorithms use a fixed number of replicas and increase workload whenever a new job arrives for execution. So an optimal approach can be used for deciding number of replicas required to give the optimal result at any given time.

2.2. Check-Pointing Approach

In Check-pointing Approach [8] running state of the process is saved to an image file and program is restarted from that file. Address space and register set of a process are saved. Fault tolerance can be done with a high level and low-level check pointing which are available in the market. Each provides different services and interfaces and due to technical reasons are application dependents. A low-level check pointing has various components like low-level check pointer, execution manager, local resource manager, GRB. Adaptive check pointing approach [9] is used for economy based Grid due to various limitation of economy based grid-like, If a fault occurs at grid resource then job is rescheduled and failed to satisfy the QoS requirements like budget and deadline because resubmitted job takes more time and more budget and In such environment there are resources that fulfill the criteria of QoS but have more tendencies toward fault. But GRB again and again selects these resources which makes the scenario worst. Faults mainly affect the resource management strategy. The main aim of this approach is to optimize the user-centric metrics (like a number of the task executed within and with exceeding deadline and budget) in the presence of a fault. Here fault occur means a grid resource is unable to complete the task within given time and budget. When such a fault is detected by the Grid Resource Broker (GRB), the fault occurrence information of that resource is updated. This fault occurrence information is used during decision making in allocating the resources to the job. This is implemented as a fault index. Fault index is maintained and updated when an allocated job completes [1, 10]. This fault index indicates the resource vulnerability to fault i.e. higher the fault index is, the higher the failure rate. The fault index of a resource increases every time when the resource fails to complete the assigned task within deadline and budget. Similarly, the fault index decreases every time when resource successfully completes the assigned job within deadline and budget.
This approach has the same components as in low-level check-pointing packages [11], but with some additional one. The components are Grid Resource, Fault Tolerance Schedule Manager, Grid Resource Broker, Grid Information Service (GIS) etc. When GRB receive a grid job from the user, it gets the contact information of available grid recourse from the GIS and then contacts with resources and tells them to send their current workload condition. Based on current workload condition of the resources, it prepares a list of resources that can execute the task with user required QoS[11]. Then GRB collects the fault index of selected resources from the Fault Tolerance schedule manager. Depending on the fault index of the resources GRB implements the following Algorithm to take the appropriate decision. In paper [12] authors proposed an efficient persistent state restoration approach. Previous approaches only considered volatile state. In paper [13], authors have analyzed nine fault tolerance job scheduling algorithms and different faults.

3. Proposed Approach

A process is said to be successfully executed on a resource if and only if it starts when the resource is up and finishes before resource goes down. If a process finish time is greater than resource failure start time, then the process is called as failed as shown in table 1. The Grid jobs (Gridlet) are large in size, so they require more computing time, which increases the probability of failure of the job. As the execution time of a process is large, larger is the probability that its finish time is greater than resource failure start time as shown in figure 1.

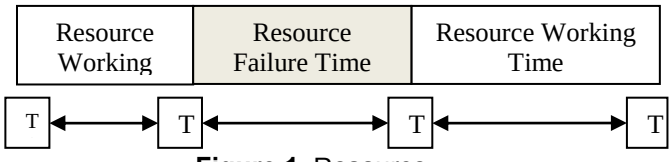

Figure 1. Resource lifetime

Table 1. Relationship between Job status and Resource Life Time

Job Status	Job Start Time(St)	Job End Time(Et)
Success	T1 < St <T2	Et<T2
Failure	T1<St	Et>T2
Success	T3<St<T4	Et<T4

The Resubmission Based approach divides a large task into a number of small size subtasks, which can be executed on grid resources and execution of all the subtasks have a similar effect as execution of the large task. Some of the subtasks are independent of each other, can be executed in parallel to decrease turnaround time which helps in achieving deadline QoS factor.

3.1. System Model

Various Components of System Model are shown in figure 2:

3.1.1 Grid User (GU): This component submits Grid jobs (Grid lets) into the grid environment. The Grid is constructed to fulfil the requirement of grid user. GU is the key component for which grid is constructed. Grid looks like a supercomputer to the grid user. GU thinks he is the only user which is using the grid so that each user is not aware of the existence of another user. GU directly communicates to Grid Resource Broker for executing its large size jobs. Grid Broker returns the results of jobs after the execution of the job.

3.1.2 Grid Resources Broker (GRB): This is the main component, which is responsible for providing a virtual image of a supercomputer to grid the user by hiding internal details. It receives Grid lets from grid user and executes them on grid infrastructure and returns back the result of Gridless. Grid Resource Broker first

Communicates to Grid Information Service to get the information of grid resources in the grid. Grid Information Service contains all static information of all resources which may be local or global to it after getting the information about all resources. GRB selects one of them on the basis of scheduling algorithm used, and assigns the grid job to that resource and uses heartbeat detection approach to detect failure of a resource. Grid resource returns the result of the job after execution completed and GRB hands over the results to Grid user.

Subtask Generator (SG): It is a part of Grid Resource Broker who deals with dividing the large size task into small size subtasks and constructs the flow graph for subtasks. It also gives a guarantee that execution of all subtasks results in the same effect as execution of large size task. SG first divides the task into the basic block on the basis of control flow dependency. Then it starts picking up each basic block and divides them into subtasks on the basis of data flow dependency. Subtask size checker is one part of SG. Subtask size is an important factor, which must be considered in this approach. If the subtask size is too large then it does not fulfil the working principle and if the subtask size is too small then most of the time is spent in scheduling the subtask, which increases the overhead as well as the execution time of the task. Subtask upper size limit and lower size limit are two metrics which are used to decide the subtask size. If subtask size is less then subtask lower limit then small subtasks at the same level of flow graph are merged to make a subtask whose size lies between the limits. If the subtask size is larger than the subtask upper limit then subtask is divided into smaller and equal size sub-subtasks so that each sub-subtask's size lies between the limits. All this functionality is done inside subtask size checker.

Subtask Manager(SM): It is also a part of Grid Resource Broker which manages execution of subtasks. It decides which subtask has to execute on which resource and at what time. It also maintains the sequence of execution of subtasks in such a way that execution of all subtasks results in the same effect as execution of large size task. In short, this part works as the scheduler of subtasks. It schedules the subtasks on the basis of the flow graph. It is also responsible for the generation of output after all subtasks successfully executed. If a particular resource fails, it resubmits all the currently executing subtasks on that resource to some other available resource

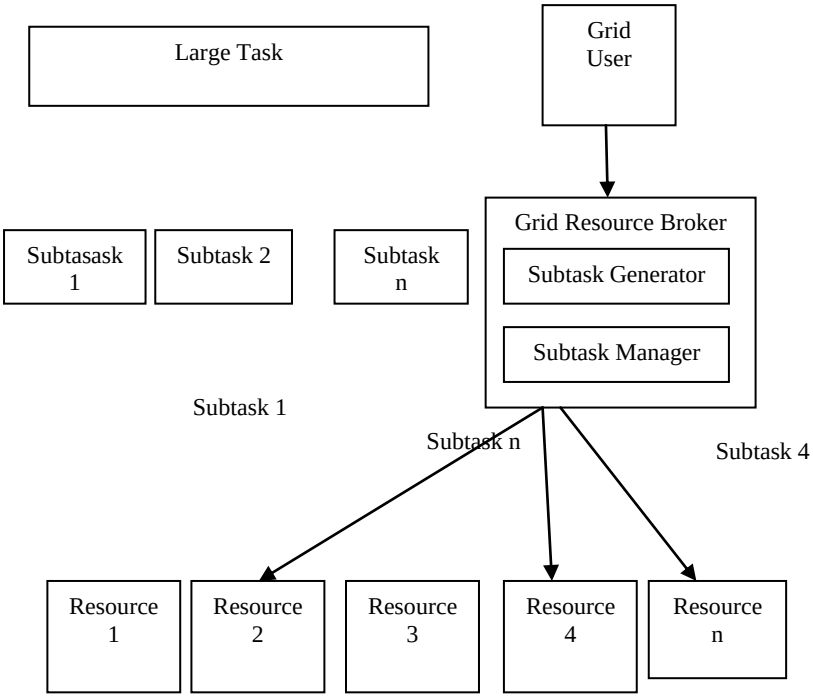

Figure 2. System Architecture

Grid Resource (GR): This is the component, on which tasks are executed. A GR consists of multiple processing elements. GRs are geographically distributed. Each GR is connected with a Grid Information Service. When a new grid resource is connected or a failed resource is recovered, first of all, it communicates with the Grid Resource Information Service to register itself. It provides its static information like a number of processing elements, the speed of processing elements, cost of resource etc. Static information of a resource is such information regarding a resource which does not change over a short time of period. Grid Resource Broker directly communicates to GR and assigns grid jobs on them and takes back the result which Grid Resource Broker hands over to grid user. Figure 3 shows the interaction between different components of the model. It tells the order in which different components interact with each other when a grid job is submitted in the grid.

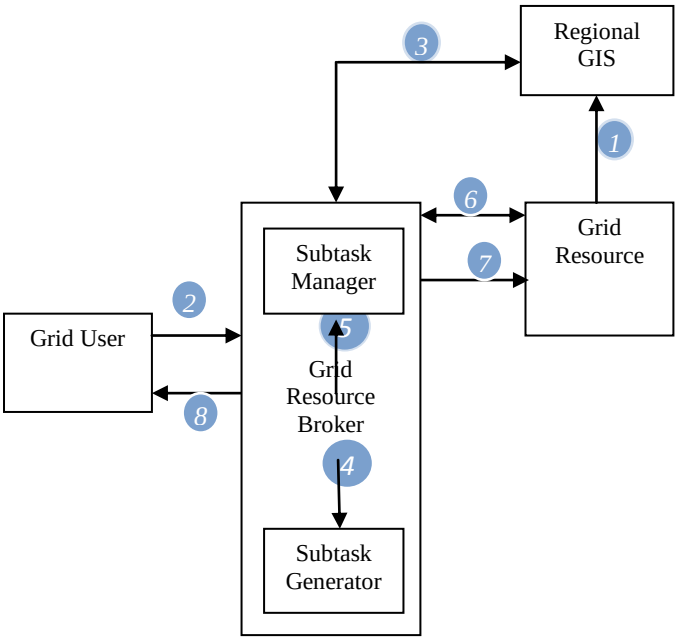

Figure 3. Interaction of different components in the proposed approach

The explanation of the interaction of different components is as follows:

1. Each resource registers itself to the designated GIS to inform its availability and its current static information.
2. Grid user sends a job, deadline time and initial budget to the Grid Resource Broker.
3. Grid Resource Broker queries GIS for the available list of resources.
4. Then Grid Resource Broker calls its Subtask Generator to divide the large task into small subtask depending on the control and data flow dependency.
5. Then Grid Resource Broker calls its Subtask Manager to schedule the subtasks.
6. Subtasks are submitted to grid resource for execution.
7. Resource executes the subtasks and results are sent back to the Grid Resource Broker.
8. Grid Resource Broker hand-over the results to Grid User.

Algorithm Subtask Generator

Step 1: Large tasks are divided into basic blocks depending on the control flow dependency.
Step 2: Repeat step3 to step6 for each instruction in every basic block.
Step 3: Compare the input variables with input set and output variables with output set of each subtask generated for the basic block in the system till now.
Step 4: If no match is found then the instruction does not depend on any existing subtask, so a new subtask is created with that instruction and input set is initialized with input variables of the instruction and output set is initialized with output variables of the instruction.
Step 5: If only one match is found then it means the instruction depends upon the instruction of matched subtask so that instruction is appended in matched subtask and input variables and output variables are added in an input set an output set of the matched subtask.
Step 6: If more than one match is found then it means instruction depends upon more than one subtask and a common child node is found in flow graph and instruction is added in that node.
Step 7: Flow graph and input and an output set of subtasks are returned.

Subtask Manager

Step 1: Repeat the step 2 to step 9 until all subtasks are executed.
Step 2: Find the subtasks in the flow graph that does not depend on any of other subtasks.
Step 3: Repeat step 4 to step 6 until all independent subtasks are scheduled.
Step 4: Find the resources which can satisfy the budget and deadline factor for the subtask.
Step 5: If no such resource is found then set the status of the job as cancel and return.
Step 6: Among the shortlisted resources, find the lightly loaded resource and schedule the subtask to that particular resource.
Step 7: Wait for the subtask to be completed.
Step 8: If the subtask is successfully executed than remove it from the flow graph and go to step 2.
Step 9: If subtask fails due to resource failure then reschedule the subtask to any other available resource.
Step 10: Check the total execution time of the job, if it is less than a deadline then set its status as successful otherwise set its status as failed.

Grid Resource Broker

Step1: Grid Job (Gridlet) is submitted by grid user to grid resource broker (GRB).
Step2: GRB divides the large task into small subtasks.
Step3: GRB schedules all independent subtasks to different resources and waits for them to complete and

then submits rest of the subtasks according to dependency sequence.

Step4: Failure of a subtask leads to resubmitting the failed subtask on another resource.

Step5: GRB maintains all the information about the subtasks of a task and when all of the subtasks are executed then GRB returns the result of the task back to the grid user.

The main strength of our approach lies in step2. In this step, first, the jobs are submitted to the subtask generator. The subtask generator first constructs basic blocks of the given task on the basis of control dependencies. Then it divides each basic block into subtasks according to their data dependency. Finally, all the constructed subtasks are passed to subtask size checker. Here we have defined two thresholds for subtask size. If subtask size is small, it will degrade the performance of the grid a lot of communication overhead is associated with the execution of subtask. Hence to alleviate this problem, the size checker

Combines different subtasks on the same level of data dependency graph until size becomes within the threshold limit. If the size of the subtask is greater than the upper threshold, it divides that subtask into continuous sub-subtasks. Finally, all the subtasks are submitted to the subtask manager (Step 3). Subtask manager schedules the subtasks of each basic block according to their data dependency graph. If any of the subtasks fails, then we have to resubmit only that particular subtask on another available resource. The benefit in Resubmission Based approach is that the subtask size is small, so we have a less computational loss and also no need to have check-pointing as tasks are already sub-divided.

4. Simulation Parameters

The Grid consists of eleven resources. Their capabilities, mean time to failure and cost are given in Table 2. Cost of the resources is in cost per million of instructions. During simulation 20 Users submit 10 jobs per user. Job size is uniformly distributed in [700,000 to 800,000] Million of instruction (MI) units. Its input and out file size are also uniformly distributed in [300 to 500] kilobytes. Resource failure is exponentially distributed with a mean time of failure of each resource. Recovery of the failed resource is also simulated using exponential distribution with a mean time of 3 minutes. In the experiment, there are three virtual organizations and each having a GIS.

5. Results and Discussions

Subtask generator component receives a large task as input and returns a set of subtasks. It also returns precedence constraints directed acyclic flow graph, input and output set of subtasks. Precedence constraints directed acyclic flow graph is created by considering control dependency and data dependency of each block. Subtask generator first converts large task into three address instruction format and then these three address instructions are provided to dependency calculating unit for further processing. The output provided by this unit is shown below:

```
==========Subtask
      0
Adjacency List is:  3 4
Parents List is:
Input Set is:  h m
Output Set is:  c
==========Subtask
      1
Adjacency List is:  4
Parents List is:
Input Set is:  b f
Output Set is:  g
==========Subtask
      2
Adjacency List is:  3
Parents List is:
Input Set is:  h i
Output Set is:  d
==========Subtask
      3
Adjacency List is:  6 8
Parents List is:  0 2
Input Set is:  a d f k l
Output Set is:  h m
```

```
==========Subtask
      5
Adjacency List is:  8
Parents List is:
Input Set is:  f i
Output Set is:  j
==========Subtask
      6
Adjacency List is:
Parents List is:  3 4
Input Set is:  b i n o
Output Set is:  a h
==========Subtask
      7
Adjacency List is:
Parents List is:
Input Set is:  k n
Output Set is:  e
==========Subtask
      8
Adjacency List is:
Parents List is: 3 4 5
Input Set is:  b c d f j l n o
Output Set is:  d f g
```

The output of the dependency calculating unit contains Adjacency List, Parents List, Input Set and Output Set of subtasks. Initially, the subtasks which are having no parents in Parents List are independent and are scheduled in parallel. After successfully executing a subtask, it is removed from Parents List of the other subtasks which are dependent on this subtask and then a subtask is searched in a system which is having empty Parents List.

Figure 4 shows the flow graph created by subtask generator unit. The node represents the subtask and a directed edge from ith node to jth node represents that jth subtask is dependent on ith subtask and jth subtask can only be scheduled after the completion of the ith subtask. Comparison of Check-Pointing approach and Resubmission Based approach.

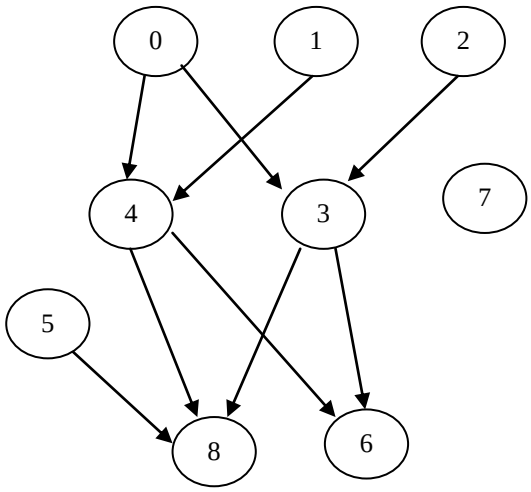

Figure 4. Precedence constraints Directed Acyclic Flow Graph.

The system models of Check-Point and Resubmission Based approach are designed and tested in GridSim Toolkit-4.0 [14]. The GridSim libraries are added to Eclipse. Eclipse is an integrated development environment (IDE) for Java. The GridSim libraries are available freely as java runtime environment (JRE), and they are linked to the Eclipse platform as an external JRE. A number of resources with different characteristics like cost. CPU rating is used to design grid infrastructure for simulation purpose as mentioned in the World Wide Grid (WWG testbed). Different numbers of Grid lets are created to evaluate these approaches. Gridlet is defined in terms of a number of instruction (in Million), input file size (in kilobyte), and output file size (in kilobytes). In the experiment, 200 Grid lets are submitted for different values of budget and deadline for measuring the performance. Grid lets are assigned to two different grids, one in which Check-Point fault tolerance approach is used and to another in which Resubmission Based fault tolerance approach is used. In both the scenarios, the first aim is to fulfill the budget and deadline QoS parameter. In Check-Point approach, we take Check-Point at regular intervals.

Different values of budget Figure 5 shows the comparison on the basis of successfully executed grid lets in resubmission based and check-point approach for different values of budget varying from 5000 to 17000₹, and deadline time is fixed to 120 seconds. Check-Point approach takes Check-Points at regular intervals, which is a time-consuming task

Figure 6 shows the number of jobs failed due to deadline time in both approaches. A number of jobs failed to finish within the deadline is plotted and it is observed that in resubmission based approach very few jobs failed to finish within deadline time as compared to the check-point approach. The check-point consumes time in saving status at each check-point at regular interval which is an overhead in the execution of the job. Let assume that 1 second is consumed in saving status at each check-point and average 10 check-points are taken during execution of a job then overall 10 seconds (1 * 10) of extra time is consumed in check-point approach. Another reason for the better performance of resubmission based approach is that it divides a large task into small subtasks which can be executed in parallel. Due to this parallelism between different subtasks, the execution time of a job is lesser as compared to execution time in check-point approach which helps the jobs to finish within deadline time. So some of the tasks do not achieve deadline and are considered as unsuccessful.

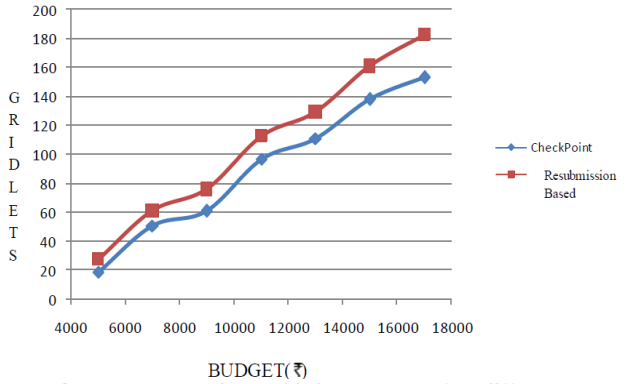

Figure 5. Number of job success for different values of budget

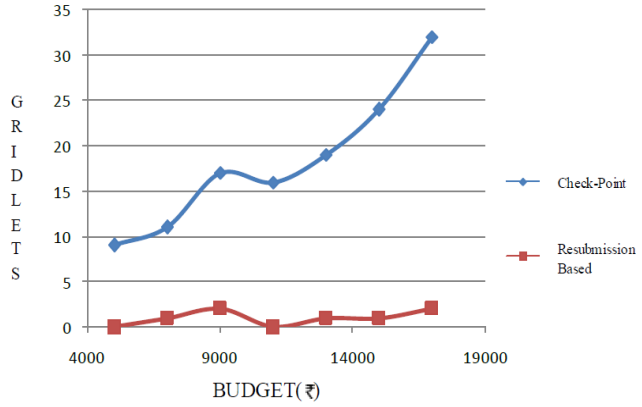

Figure 6. Number of jobs fails to achieve the deadline

But in resubmission based approach no Check-Pointing is done and independent subtasks can run in parallel, which helps in achieving deadline hence more jobs are successfully executed in resubmission based approach as compared to Check-Point approach as shown in figure 7. At the initial stage of the experiment near about 30 jobs get the resources. Rest of 170 jobs fail to satisfy QoS parameters like deadline and budgets, so these 170 jobs are considered as cancelled jobs. Out of 30 jobs in checkpoint approach near about 20 jobs successfully executed and rest of the 10 jobs failed to complete within deadline time. In resubmission based approach independent subtasks can execute in parallel so all 30 jobs are successfully executed. It also reduces average execution time in resubmission based approach.

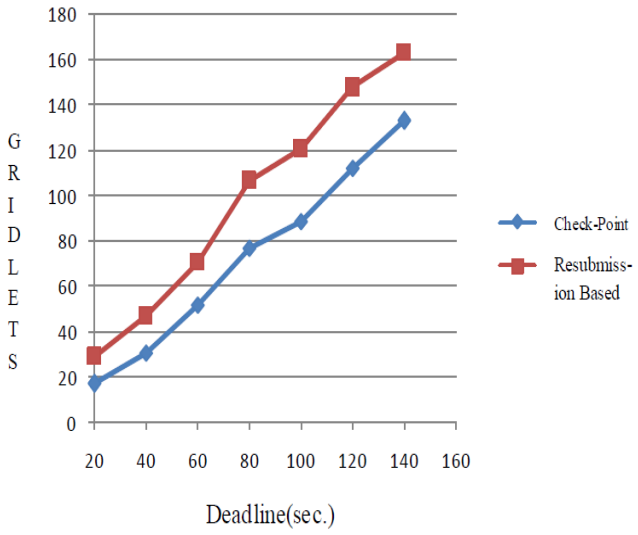

Figure 7. Number of job success for different values of the deadline

Figure 7 shows the comparison in a number of gridlet successfully executed in resubmission based and check-point based approach for different values of deadline time which varies from 20 to 150 second and budget is fixed to 12000₹. Some time is consumed in check-pointing which lead to failing a task to complete within the deadline. The graph shows that more jobs are completed within deadline time in resubmission based approach as compared to check-point based approach. At the initial stage of the experiment near about 40 jobs get the resources and rest of 160 jobs are failed to satisfy QoS parameters like deadline and budgets, so these 160 jobs are considered as a canceled job. Out of 40 jobs in checkpoint approach near about 20 jobs successfully executed and rest of the 20 jobs failed to complete within deadline time but in resubmission based approach 30 jobs are successfully executed due to independent subtasks can execute in parallel. 10 jobs are failed in resubmission based approach to finish within deadline time because deadline time is small in an initial stage of the experiment.

References

[1] Kim, Y. (2012). A Fault-tolerant Mutual Exclusion Algorithm in Asynchronous Distributed Systems. *International Journal of Contents*, *8*(4), 1-6.

[2] J Baker, M., Buyya, R., & Laforenza, D. (2002). Grids and Grid technologies for wide-area distributed computing. *Software: Practice and Experience*, *32*(15), 1437-1466.

[3] Foster, I., Kesselman, C., & Tuecke, S. (2001). The anatomy of the grid: Enabling scalable virtual organizations. *The International Journal of High-Performance Computing Applications*, *15*(3), 200-222.

[4] Krauter, K., Buyya, R., & Maheswaran, M. (2002). A taxonomy and survey of grid resource management systems for distributed computing. *Software: Practice and Experience*, *32*(2), 135-164.

[5] Medeiros, R., Cirne, W., Brasileiro, F., & Sauvé, J. (2003, November). Faults in Grids: Why are they so bad and What can be done about it?. In *Grid Computing, 2003. Proceedings. Fourth International Workshop on* (pp. 18-24). IEEE.

[6] Litke, A., Tserpes, K., Dolkas, K., & Varvarigou, T. (2005, February). A task replication and fair resource management scheme for fault-tolerant grids. In *European Grid Conference*(pp. 1022-1031). Springer, Berlin, Heidelberg.

[7] Erciyes, K. (2006, December). A replication-based fault tolerance protocol using group communication for the grid. In *International Symposium on Parallel and Distributed Processing and Applications* (pp. 672-681). Springer, Berlin, Heidelberg.

[8] Mandal, P. S., & Mukhopadhyaya, K. (2006). Performance analysis of different checkpointing and recovery schemes using stochastic models.

[9] Nazir, B., Qureshi, K., & Manuel, P. (2009). Adaptive checkpointing strategy to tolerate faults in the economy based grid. *The Journal of Supercomputing*, *50*(1), 1-18.

[10] Jiang, C., Wang, C., Liu, X., & Zhao, Y. (2007, July). A fuzzy logic approach for secure and fault-tolerant grid job scheduling. In *International Conference on Autonomic and Trusted Computing* (pp. 549-558). Springer, Berlin, Heidelberg.

[11] Riteau, P., Lebre, A., & Morin, C. (2009, May). Handling persistent states in process checkpoint/restart mechanisms for HPC systems. In *Cluster Computing and the Grid, 2009. CCGRID'09. 9th IEEE/ACM International Symposium on* (pp. 404-411). IEEE.

[12] Jankowski, G., Januszewski, R., Mikolajczak, R., & Kovacs, J. (2008). Improving the fault tolerance level within the GRID computing environment-integration with the low-level checkpointing packages. *Core Grid TR-0158 June*, *16*.

[13] Balpande, M., & Shrawankar, U. (2014, April). Robust fault tolerant job scheduling approach in a Grid environment. In *Circuits, Systems, Communication and Information Technology Applications (CSCITA), 2014 International Conference on* (pp. 259-264). IEEE.

[14] GridSim toolkit. [Online] Available: http://www.gridbus.org/gridsim/

Table 2. Simulation Parameters

Resource Name	***(Location)***	**Nodes**	**Rating**	**Policy**	**GIS**	**Cost(₹)**	**Mean Time to Failure**
RAL	(UK)	41	4900	Space-Shared	2	490	60
Imp.	College	52	6200	Space-Shared	2	620	100
NorduGrid	(Norway)	17	2000	Space-Shared	2	200	340
NIKHEF	(Netherlands)	18	2100	Space-Shared	0	210	340
Lyon	(France)	12	1400	Space-Shared	0	140	450
CERN	(Switzerland)	59	7000	Space-Shared	0	700	75
Milano	(Italy)	5	7000	Space-Shared	1	700	55
Torino	(Italy)	2	300	Time-Shared	1	30	130
Rome	(Italy)	5	600	Space-Shared	1	60	110
Padova	(Italy)	1	100	Time-Shared	1	10	150
Bologna	(Italy)	67	8000	Space-Shared	1	80	150