

On Demand Check Pointing for Grid Application Reliability using Communicating Process Model

Baghavathi Priya.S*, Chandrasekaran Subramaniam**, Ravichandran.T ***

*Research Scholar, Jawaharlal Nehru Technological University, Hyderabad, Andhra Pradesh, India.

** Dean & Professor, Information Technology Department, Rajalakshmi Engineering College, Chennai, India.

*** Principal, Hindustan Institute of Technology, Coimbatore, Tamil Nadu, India.

baghavathipriya@yahoo.com , chandrasekaran_s@msn.com, dr.t.ravichandran@gmail.com

Abstract—The objective of the work is to propose an on-demand asynchronous check pointing technique for the fault recovery of a grid application in communicating process approach. The formal modelling of processes using LOTOS is done wherein the process features are declared in terms of possibilities of rollback and replicas permitted to accept the assigned tasks as decided by the scheduler. If any process is tending to be faulty in run time that will be detected by check pointing mechanism through the Task Dependency Graph (TDG) and their respective worst case execution time and dead line parameters are used to decide the schedulability. The Asynchronous Check Pointing On Demand (ACP-OD) approach is used to enhance the grid application reliability through the needed fault tolerant services. The scheduling of concurrent tasks can be done using the proposed Concurrent Task Scheduling Algorithm (CTSA) algorithm to recover from the faulty states using replication or rollback techniques. The check pointing and replication mechanisms have been used in which the synchronization between communicating processes is needed to enhance the efficiency of check pointing mechanism. The model is tested with a number of rollback variables treating the application as a Stochastic Activity Network (SAN) using Mobius.

Keywords— Check pointing, Reliability, Process, Rollback, Replication.

I. INTRODUCTION

The application reliability in a grid environment depends on the synchronized behaviour of many heterogeneous hardware and software resources deployed in a distributed manner. The efficient scheduling of such resources can be achieved in a multi-tier component or service mapping between the processes and the replicated applications and possible rollbacks. A large scale infra structure services and fault report services are needed to claim the maximum reliability for the application deployed or demanded over the grid. Irrespective of the service, the grid components ranging from grid node to grid broker must communicate in asynchronous or synchronous manner to complete the submitted tasks from the grid users. The Replica Resource Selection Algorithm (RRSA) proposed in 2010 uses a check point setter through the mean time to release (MTTR) the service or resource is optimized [1]. But the mechanism for the faster recovery from the faulty status of the particular process is not considered. A fault tolerant load balancing model was proposed using a fault

detector and manager services through which the check pointing technique was compared [2]. Even with standard interfaces and communications protocols in place, resource heterogeneity and dynamism will likely lead to component interactions that result in faults and failures which when executing grid user applications. Another well-known failure detection problem occurs in asynchronous distributed systems where management functions are decentralized and they may be subject to failure. One important issue is finding efficient methods for check pointing many concurrent, intercommunicating processes, so that in the event of failure, they can resume processing from a common saved state. Early efforts in using checkpoint and process migration in large-scale grids were reported in the Legion system and Cactus. A work flow approach is used in the Linked Environment for Atmospheric Discovery (LEAD) infrastructure to establish a fault tolerant computational grid and the success probability was determined [3]. A Stochastic modelling was introduced to analyse the performance of different check pointing schemes where the time complexity of synchronous and asynchronous check pointing was focused. In Synchronous Check pointing if all the processes take check points at the same time instant, the set of check points would be consistent. But since globally synchronized clocks are very difficult to implement, processes may take check points within an interval. Processes synchronize through system messages before taking checkpoints. These synchronization messages contribute to extra overhead [4]. This model of analysis compared both the pessimistic and optimistic approaches to find out the minimum message logging [5]. A decentralized fault tolerant framework for grid computing was proposed in which node-bound proactive maintenance strategies are identified to reduce the service failures [6]. A flexible failure handling framework for grid services was proposed in which fault notification generation was utilized to determine the state of the task in progress as Grid-WFS[7]. The Grid Resource Management System GRID-RMS addresses the need for adaptability and scalability of the grid resources to meet the specific needs demanded by the grid users[8]. The main focus of the paper is to formally specify the behaviour needed for the various fault tolerant grid services through check pointing strategy. The proposed check pointing techniques invoke the necessary replicas or rollbacks in order to meet the user application reliability requirements in the case of faults

occurred either in the processes and or in the communication channels.

The paper is organized as follows: Section II discusses a generic Grid architecture needed for enhancing the application reliability through check pointing strategy and the essential component services. Section III explores the need for a feasible mapping between available application replicas and the reliable roll backs of the erroneous tasks within the grid environment and its LOTOS specification for verification purposes. The formal specification enables not only to avoid the faults but also to trace the failed processes in a segmented way. Section IV describes the computational model of concurrent activation of processes with the help of their corresponding stacks and the collaboration or various grids component services and the application reliability analysis through process failure probabilities. Section V focuses on the validity of the model based on the cost and complexity of the replicas and rollbacks respectively and concludes the work.

II. GENERIC GRID ARCHITECTURE FOR FAULT TOLERANT GRID APPLICATION

Grid resources include processor clusters, supercomputers, storage devices, and related hardware, together with operating system and other software for managing these resources. Grid resources can also include dedicated software components that may be engaged by users to perform various functions, such as data mining and other analysis. A third category of grid resources are data stores used in data and multimedia grids. The most common technique of check pointing can be applied to improve the fault tolerant behaviour of the application by having suitable number of replicas for the critical tasks in the

computational scheme. In the case of data grid services, there is a need for roll back mechanism since the data from the previous node may be faulty or to recover from the faulty state of the forwarding nodes in a cluster as shown in Figure 1. The Grid jobs are executed by the computational grid as follows: (i) The Grid users submit their jobs to the scheduler Manager their requirements i.e., deadline in which the users want their jobs to be executed. (ii) The Scheduler Manager component submits user jobs to the Resource Information Service for getting optimized resources. (iii) The Resource Information Service submits job with matched resource to the Job Dispatcher to a Resource Pool which is a member of a grid and it offers computing services to grid users.

1) *Schedule Manager:*

The Scheduler Manager plays a vital role in the grid in receiving the jobs from grid users. It selects feasible resources by submitting jobs to Resource Information Service. Then it generates job-to-resource mapping and the entities of scheduler are the Schedule Manager providing the Resource Information Service, Job Dispatcher service. A job dispatcher dispatches the jobs one by one to the Central Checkpoint Unit.

2) *Checkpoint Unit:*

It receives the scheduled jobs from the schedule manager and sets checkpoint based on the failure rate of the resource on which it is scheduled. Then it submits the job to the resource. Central Checkpoint Unit receives job completion message or job failure message from the resource pool and responds to that accordingly. During execution, if job failure occurs, the job is reached.

3) *Checkpoint Controller:*

On each checkpoint set by the Central checkpoint Unit, job status is reported to the checkpoint Controller. Checkpoint Controller save the job status and return it on demand i.e., during job/resource failure.

4) *Checkpoint Rollback or Check Point Replication:*

When a new checkpoint is created, the Checkpoint Controller initiates Checkpoint Rollback mechanism. It will perform the possible rollback operations for a faulty process and find the list of possible states from recovery is possible. If rollback is not possible, the Checkpoint Controller initiates Checkpoint Replication services from where once the application replicas may be called. If not available the on demand replication is to be carried out and the critical services are replicated. The information service logs on the details like the time of initiation of the replicas and their identity are stored in the buffer inside the Checkpoint Controller.

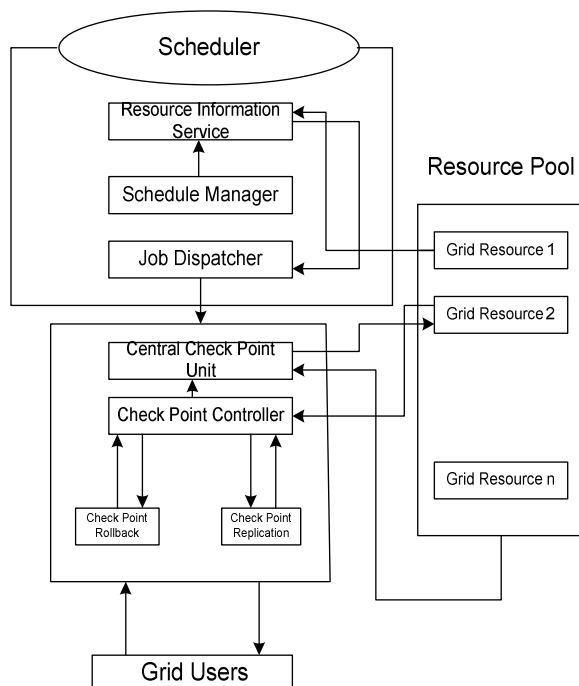


Figure1. Checkpoint architecture with Replication and Rollback

III. PROCESS MAPPING ACROSS REPLICAS & ROLLBACKS

The check pointing can be carried out in an asynchronous manner either on demand or on supervision by the grid manager service. Based on the nature of the tasks periodicity and possibility of rollbacks, the processes may request to find a mapping from replica to rollbacks. The start time and

deadline parameters of the submitted tasks, the check point controller identifies by running the feasibility and schedule analysis of all the tasks. Once a set of solutions are identified, the controller checks the availability of the needed resources from the respective service and dispatches the jobs to them for early completion with or without replication of the application.

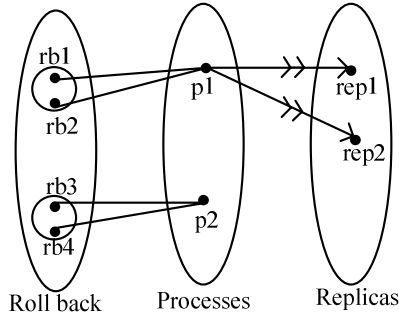


Figure2. Mapping of Replicas and Rollbacks

The mapping of the process across the rollbacks and replicas are shown in Figure 2 and if there is no mapping available, the application may be replicated on demand or the grid manager starts negotiating with the user regarding the slippage in the scheduling for further actions as in the proposed concurrent task scheduling algorithm (CTSA) mentioned below.

Concurrent Task Schedulability Algorithm::

Tasks: n

Independent tasks:m

Concurrent : n-m

No of replica : n_{re}

nN of rollback : n_{ro}

$P(\text{selection replica or rollback}) = 0.5$

select one of n_{re} or one of n_{ro}

probability of failure of one task P_x^y

if $x \in n-m$ // failed task is concurrent

$$P_{\text{failure}} = P_x^y * \sum_{i=1}^{n-m} P_{x+a}^y$$

// probability that concurrent process fail one after another//
else // failed task is independent

$$P_{\text{failure}} = P_x^y$$

Communication failure= $P(CF)$

// failure in synchronous or asynchronous
mode of communication//

$$P_{nre}^{\text{tot failure}} = P_{\text{failure}} * P(CF)$$

$$\left\{ \sum_{j=1}^q \sum_{k=1}^{m+1} 1/n-m * 0.5 [P(r_k / r) + P(re_j / re)] \right.$$

$$* \sum_{nre}^{q=1} 1/(n-m-q) \} * P(CF) \text{ // Concurrent}$$

$$\left\{ \sum_{j=1}^q \sum_{k=1}^{m+1} 1/m * 0.5 [P(r_k / r) + P(re_j / re)] \right.$$

$$* P(CF) \text{ // Independent}$$

Figure3. Concurrent Task Schedulability Algorithm

The mapping of the processes and their permitted rollbacks and possible replicas are shown which are under the control of the scheduler. The replication of a process is executed in parallel with the identical state but on different resources so that one replica is guaranteed to finish the process correctly.

specification

grid[a1,a2,b1,b2,c1,c2,d1,d2,e1,e2,f1,f2,g1,g2,h1,h2] := noexit

//behavioural specification

behaviour

process1[sequence of tasks]

after execution of process1

process2[sequence of tasks]

interleaving possible

process3 [sequence of tasks], process4 [sequence of tasks]
after execution of process3 and process4

process5[sequence of tasks]

interleaving possible

process6[sequence of tasks], process7[sequence of tasks],
process8[sequence of tasks]

(p1[a1,a2] >> p2[b1,b2] >> p3[c1,c2] ||| p4[d1,d2] >>
p5[e1,e2] >> p6[f1,f2] ||| p7[g1,g2] ||| p8[h1,h2])

where

//process instantiation

process p1 [a1,a2] noexit:=

(

Define atomic | compound

a1: task(start time, worst case execution time,
deadline, cost)

a2: task(start time, worst case execution time,
deadline, cost)

// features available

feature1:

rollback? rollback permitted: nat

feature2:

replication? replication permitted: nat

)

endproc

Figure4. LOTOS Specification for Check pointing(Asynchronous)

As shown above, the formal behavioural specifications are given in LOTOS for each process with their associated attributes that are to be declared initially. For example, process P1 has associated attributes a1 and a2 and for process P2, b1 and b2 are associated attributes and so on. Next the behavioural specification of each process has been defined as sequence of tasks that are to be undertaken. After execution of process the next process will be executed. Here interleaving is possible. Process 3 and Process 4 are interleaving processes. Process 6, Process 7 and Process 8 are also interleaving processes. The process instantiation can also be done and the attributes like a1 may consists of the start time, worst case execution time, deadline and cost of the individual tasks may

IV. COMPUTATIONAL MODEL OF CONCURRENT ACTIVATION OF PROCESSES

```

graph TD
    GS[Grid Supervisor] <--> GM[Grid Monitor]
    GM <--> GR[Grid Requestor]
    GM <--> TE[Time estimator]
    TE <--> AL[Allocato]
    AL <--> EX[Executor]
    EX --> TE
    
    AL --> S["S stack  
(a1) | | | | |"]
    AL --> W["W stack  
Worst | | | | |  
(a1)"]
    AL --> D["D stack  
Dead lines | | | | |  
(a1)"]
    AL --> C["C stack  
Cost | | | | |  
(a1)"]
    AL --> RS["Rollback stack  
rb(a1) | | | | |"]
    AL --> R["Replica stack"]
    
    EX --> C1{"Checking worst case  
execution time"}
    C1 -- Y --> ROLL[Rollback]
    C1 -- N --> C2{"Checking deadline"}
    C2 -- Y --> ROLL
    C2 -- N --> C3{"Checking cost"}
    C3 -- Y --> ROLL
    C3 -- N --> SUM["Summation of  
Check Point time"]
    
    SUM --> C4{"Checkpoint checking"}
    C4 -- Y --> SR[Start replication]
    C4 -- N --> ROLL
    
    SR --> NO[No. of rollbacks]
    NO --> RW[Rewrite rollback variables]
    RW --> CM[Correction mechanism]
    CM --> FF[Feed forward]
    FF --> SUM
    
    ROLL --> SUM
    
    SUM --> JC[Job completion]
  
```

If the task is a dependent one then it will start performing the rollback mechanism by calling the FALL methods in the appropriate session in the protected mode. Then the recovery or the correction mechanism can be initiated so as to forward the status to the time estimator service component through the global grid interface. The time estimator determines the most probable time for calling the replicas using CALL methods through the private interface allotted for that process to minimise the replication time. In the coordinated checkpointing mechanism, the processes will attempt to synchronize the call or demand for check points to ensure individual saved states are consistent with respect to each other. This requires a precise accounting for messages that are in-transit when the

The figure illustrates the architecture of the proposed framework, divided into two main components: Rollback and Replication.

Rollback Path (Top Diagram):

- Inputs:** Timeed_Activity and Recovery:Rollback.
- Processing:** The inputs are processed by four parallel units (P1, P2, P3, P4). These units feed into two intermediate nodes, rb1 and rb2.
- Intermediate Processing:** The outputs from rb1 and rb2 feed into a node labeled "Rewrite all rollback variables".
- Correction:** The output from "Rewrite all rollback variables" feeds into a node labeled "Correction".
- Output:** The final output is "Feed Forward".

Replication Path (Bottom Diagram):

- Inputs:** Timeed_Activity and Recovery:Replication.
- Processing:** The inputs are processed by four parallel units (P1, P2, P3, P4). These units feed into two intermediate nodes, re1 and re2.
- Output:** The outputs from re1 and re2 feed into a node labeled "Feed Forward".

The checkpoint time can be calculated as the summation of checkpoint start time, status collection time, decision time, rollback time, rewrite rollback time, correction time, feed forward time and restart time of individual processes for the assigned tasks. The Task Dependency XML service is shown below:

```
name>Check Pointing Grid Application</name>
<declarations>
  <declaration environment="true" name="x1"
type="urgent chan" value="10" />
</declarations>
</system>
<tasktable>
  <task C="13" D="200" T="45" behaviour="P"
name="task1" />
  <task C="12" D="300" T="65" behaviour="P"
name="task2" />
  <task C="11" D="25" P="2" T="25" name="t" />
  <task C="5" D="15" P="1" T="15" name="tas" />
  <task C="10" D="20" P="3" T="20" depend=
/ >
  <task C="8" D="10" P="4" T="10" name="tas" />
  <task C="9" D="11" P="2" T="11" depend="t" />
  <task C="7" D="15" P="1" T="15" name="tas" />
  <task C="6" P="4" name="task11" />
  <task C="10" D="350" T="350" behaviour="I"
name="task3" />
</tasktable>
```

The grid application reliability can be calculated based on the successful negotiation and the synchronization between various component services, The reliability calculation also considers the cost factor on which the service agreement was drawn with grid users, The on demand check pointing will identify the failed processes and activate the application replicas at the detected node or from the other node identified by the global detection service component. The reliability of the grid application was determined by injecting random faults in the task and in the attribute level and the worst case and computing time are tabulated as shown in Table 1&2.

$$R_{app}^g(\text{time}, \text{cost})$$

$$R_{app}^g(\text{time}) = 1 - P_{total}^{time}(\text{failure})$$

$$= 1 - P_{total}(\text{process_faulty})$$

$$= 1 - \sum_{i=1}^n P_f(\text{task})$$

$$= 1 - \left[\sum_{i=1}^n P(\text{concurrent tasks}) + \sum_{j=1}^m P(\text{independent tasks}) \right] \quad (1)$$

$$R_{app}^g(\text{cost}) = 1 - P_{total}^{cost}(\text{task})$$

$$= 1 - \left[\sum_{k=1}^{n+m} P(\text{negotiation}) * P(\text{synchronization}) \right] \quad (2)$$

$$R_{app}^g(t, c) = R_{app}^g(t) + R_{app}^g(\text{cost}) \quad (3)$$

TABLE 1. WORST CASE RESPONSE TIME

Name	Computing time(ms)	WCRT (ms)	Deadline (ms)
Task1	10	42	350
Task2	5	31	250
Task3	7	17	150
Task4	9	26	200
Task5	6	37	300
Task6	5	42	450
Task7	15	23	200
Task8	12	45	350
Task9	7	34	300
Task10	9	56	100

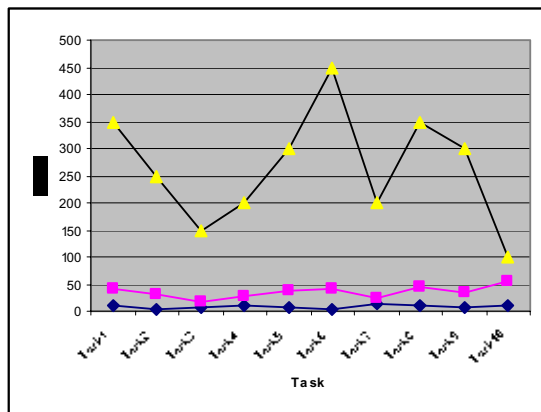


Figure7. Worst Case Response Time

TABLE 2. WORST CASE RESPONSE TIME

Name	Computing time(ms)	WCRT (ms)	Deadline (ms)
Task1	10	42	350
Task2	5	31	250
Task3	7	17	150
Task4	50	32	40
Task5	6	37	300
Task6	5	42	450
Task7	15	23	200
Task8	35	12	15
Task9	7	34	300
Task10	9	56	100

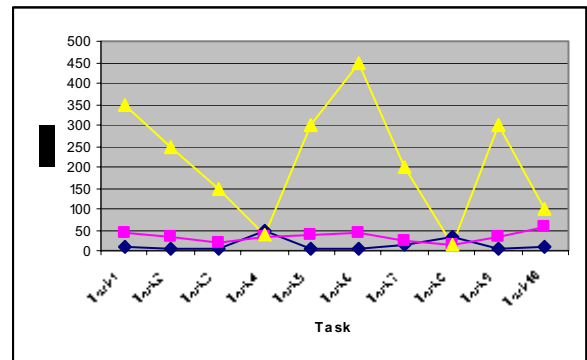


Figure8. Worst Case Response Time

V. CONCLUSION

The check pointing strategy is used to detect the failed process in the grid environment using the on demand approach and the recovery action is carried with the help of concurrent task scheduling algorithm. The formal specification and verification is carried out using LOTOS in order to avoid the faults in the processes after the jobs are assigned. The computational model of collaboration of grid component services using attribute stacks is introduced. The task dependency is considered to identify the best scheduling and the corresponding replicas are activated to improve the reliability of grid application through fault tolerant techniques.

The technique is tested as a stochastic activity network and the schedulability is shown as a timing graph. The work is having a limitation on the number of concurrent tasks that should always be maintained as lower than that of the independent tasks. The size of the attribute stacks should be increasingly large if the numbers of grid requests are more than that of the available resources at any point of run time. The future work focuses on the enhancement of the proposed algorithm in the case of resource conflicts when two processes are demanding the same resource at the same time when there is no application replicas exist in the grid environment.

REFERENCES

- [1] Malarvizhi Nandagopal, V.Rhymend Uthariaraj, "Fault Tolerant Scheduling Strategy for Computational Grid Environment", International Journal of Engineering Science and Technology, Vol.2(9), 4361-4372, 2010.

- [2] J.Jaybharathy and Ayeshaa Parveen.A,"A Fault Tolerant Load Balancing Model for Grid Environment", Internatinal Journal of Recent trends in Engineering, Vol 2, No.2, 162-164,2009.
- [3] Gopi Kandaswamy, Anirban Mandal, and Daniel A.Reed, "Fault Tolerance and Recovery of Scientific Workflows on Computational Grids", IEEE Computer Society, 2008.
- [4] Partha Sarathi Mandal, Krishnendu Mukhopadhyaya, "Performance analysis of different checkpointing and recovery schemes using stochastic model", Journal of Parallel and Distributed Computing, 66, 99-107,2006.
- [5] Youcef Derbal,"A new fault-tolerance framework for grid computing", An International Journal on Multiagent and Grid System, 2,115-133, 2006.
- [6] Jia Yu and Rajkumar Buyya,"A Taxonomy of Scientific Workflow Systems for Grid Computing", SIGMOD Record, Vol.34, No.3, 2005.
- [7] Soonwook Hwang and Carl Kesselman, "A Flexible Framework for Fault Tolerance in the Grid", Journal of Grid Computing 1,251-272,2003.
- [8] Klaus Krauter, Rajkumar Buyya and Muthucumaru Maheswaran, " A taxonomy and survey of grid resource management systems for distributed computing ", Software – Practice and Experience 32, 135-164,2002.