International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 02 Issue: 09 | Dec-2015 www.irjet.net p-ISSN: 2395-0072

A Survey on Task Checkpointing and Replication based Fault Tolerance in Grid Computing

Mr.M.Nakkeeran[1].

[1] *Assistant Professor, Dept. of CSE, SVS College of Engineering, Tamil Nadu, India*

Abstract - ***A grid is a distributed computational and storage environment often composed of heterogeneous autonomously managed subsystems. As a result, varying resource availability becomes commonplace, often resulting in loss and delay of executing jobs. To ensure good grid performance, fault tolerance should be taken in to account. Commonly utilized techniques for providing fault tolerance in distributed systems are periodic job Checkpointing and replication. While very robust, both techniques can delay job execution if inappropriate Checkpointing intervals and replica numbers are chosen. This survey work provides several heuristics that dynamically adapt the above mentioned parameters based on information on grid status to provide high job throughput in the presence of failure while reducing the system overhead. This survey results on experiments are evaluated in a newly developed grid simulation environment SimGrid [2], which allows for easy modeling of dynamic system and job behavior. The workload and system parameters derived from logs that were collected from results have shown that adaptive approaches can considerably improve system performance, while the preference for one of the solutions depends on particular system characteristics, such as load, job submission patterns, and failure frequency.***

Key Words: *Checkpointing, replication, fault tolerance, throughput*

1. RESEARCH MOTIVATION

Compared to other distributed environments, such as clusters, complexity of grids mainly originates from decentralized management and resource heterogeneity. The latter refers to hardware, as well as to foreseen utilization. These characteristics often lead to strong variations in grid availability, which in particular depends on resource and network failure rates, administrative policies, and fluctuations in system load. Apparently, runtime changes in system availability can significantly affect application (job) execution. Since for a large group of time-critical or time consuming jobs delay and loss are not acceptable, fault tolerance should be taken into account. Providing fault tolerance in a distributed environment, while optimizing resource utilization and job execution times, is a challenging task. To accomplish it, two techniques are often applied: Job Checkpointing and Job Replication.

Although both methods aim to improve system performance in the presence of failure, their effectiveness largely depends on tuning runtime parameters such as the checkpointing interval and the number of replicas [1], [9], [13].

2. LITERATURE SURVEY

A large number of research efforts have already been devoted to fault tolerance in the scope of distributed environments. Aspects that have been explored include the design and implementation of fault detection services [4], [5], as well as the development of failure prediction [3], [6], [7], [8] and recovery strategies [9], [10], [11]. The latter are often implemented through job checkpointing in combination with migration and job replication. Although both methods aim to improve system performance in the presence of failure, their effectiveness largely depends on tuning runtime parameters such as the checkpointing interval and the number of replicas [12], [13]. Determining optimal values for these parameters is far from trivial, for it requires good knowledge of the application and the distributed system at hand.

3. CHECKPOINTING HEURISTICS

To tackle the Checkpointing overhead and scalability concerns, different approaches are addressed in this literature. One well-researched technique is known as incremental Checkpointing [6]. It reduces data stored during Checkpointing to only blocks of memory modified since the last checkpoint. In “Optimizing Adaptive Checkpointing Schemes for Grid Workflow Systems,” [11], another set of cooperative Checkpointing schemes is proposed that dynamically adjust the Checkpointing interval with as an objective timely job completion in the presence of failure. This schemes uses information on remaining job execution time, time left before the deadline, and the expected remaining number of failures before job termination. In "Performance and Effectiveness Trade-Off for Checkpointing in Fault-Tolerant Distributed Systems," [14], in turn, consider only dynamic Checkpointing interval reduction in case it leads to computational gain, which is quantified by the sum of the differences between the means for fault-affected and fault-unaffected job response times.

3.1 ADAPTIVE INCREMENTAL CHECKPOINT

Adaptive Incremental Checkpoint technique uses a secure hash function to uniquely identify changed blocks in memory. This algorithm is the first self-optimizing algorithm that dynamically computes the optimal block boundaries, based on the history of changed blocks which provides better opportunities for minimizing checkpoint file size. Since the hash is computed in software which do not need any system support for this. This mechanism is implemented and tested on the BlueGene/L system. It results in reduction of average checkpoint file size and adaptively towards application's memory access patterns.
The Adaptive incremental Checkpointing seems to hold better potential (w.r.t reduced average checkpoint file size, reduced Checkpointing time and automatic block-size tuning) than previous incremental Checkpointing approaches, although more experimentation on larger platforms and further fine-tuning is needed to draw very strong conclusions.

3.2 LAST FAILURE DEPENDENT CHECKPOINTING ALGORITHM

The main disadvantage of unconditional periodic job Checkpointing (Periodic) is that it performs identically whether the job is executed on a volatile or on a stable resource. The goal of LastFailureCP is to reduce the overhead introduced by excessive checkpointing in relatively stable distributed environments, i.e., the algorithm omits unnecessary checkpoints of the job *j* based on its estimated total execution time and the failure frequency of the resource r to which *j* is assigned (see Fig. -1)

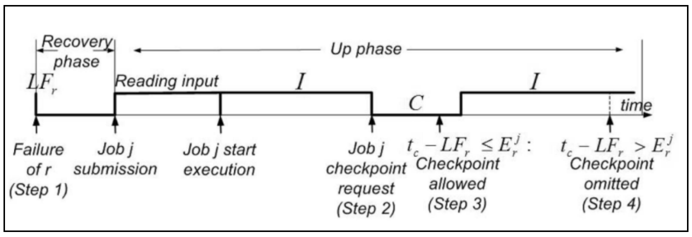

Recovery phase
Up phase
LF_r
Reading input
I
C
I
time
Failure of r (Step 1)
Job j submission
Job j start execution
Job j checkpoint request (Step 2)
$t_c - LF_r \le E_r^j$: Checkpoint allowed (Step 3)
$t_c - LF_r > E_r^j$ Checkpoint omitted (Step 4)

Fig -1: Operation of LastFailureCP on a resource running a single job

For each resource, the algorithm keeps a time stamp LF_r of its last detected failure (Step 1). When no failure has occurred, LF_r is initiated with the system start time. After an execution interval *I*, each job running on an active resource generates a checkpointing request (Step 2). The request is subsequently evaluated by the GSched and it is allowed only if the comparison $(t_c - LF_r) \le E_r^j$ evaluates to true (Step 3), where t_c is the current system time. As was previously mentioned, each checkpoint generation leads to runtime overhead C, which prolongs the execution of j (Step 3). If $(t_c - LF_r) > E_r^j$, the checkpoint is omitted to avoid the overhead as it is assumed that the resource is "stable" (Step 4). To prevent excessively long checkpoint suspension, a maximum number of omissions can be defined.

3.3 MEAN FAILURE DEPENDENT CHECKPOINTING ALGORITHM

Contrary to LastFailureCP that only considers checkpoint omissions, The MeanFailureCP algorithm dynamically modifies the initially specified Checkpointing frequency to deal with inappropriate Checkpointing intervals. (see Fig. -2)

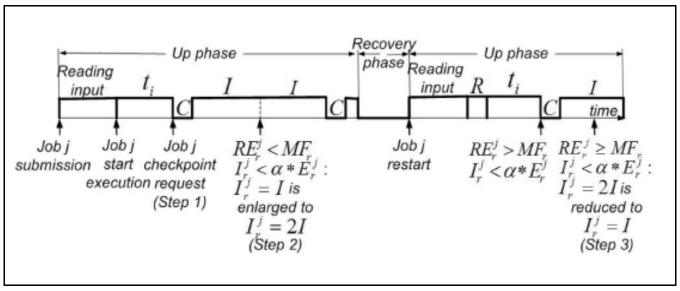

Up phase
Recovery phase
Up phase
Reading input
t_i
I
I
C
C
Reading input
R
t_i
C
I
time
Job j submission
Job j start execution
Job j checkpoint request (Step 1)
$RE_r^j < MF_r$ $I_r^j < \alpha * E_r^j$: $I_r^j = I$ is enlarged to $I_r^j = 2I$ (Step 2)
Job j restart
$RE_r^j > MF_r$ $I_r^j < \alpha * E_r^j$
$RE_r^j \ge MF_r$ $I_r^j < \alpha * E_r^j$: $I_r^j = 2I$ is reduced to $I_r^j = I$ (Step 3)

Fig-2 Operation of MeanFailureCP on a resource running a single job.

This algorithm modifies the checkpointing interval based on the runtime information on the remaining job execution time (RE_r^j) and the average failure interval (MF_r) of the resource *r* where the job *j* is assigned, which results in a customized checkpointing interval. The use of MF_r , instead of LF_r reduces the effect of an individual failure event. While PeriodicCP and LastFailureCP are first run after the expiration of the predefined Checkpointing interval, the MeanFailureCP activates Checkpointing within a fixed and preferably short time period t_i after the beginning of a job execution (Step 1). The latter approach opens the possibility to modify the Checkpointing frequency at the early stage of job processing. Each time the Checkpointing is performed, is adapted as follows: If $RE_r^j < MF_r$ and $I_r^j < \alpha \times E_r^j$, where $\alpha < 1$, the frequency of Checkpointing will be reduced by increasing the Checkpointing interval $I_r^{j_{new}} = I_r^{j_{old}} + I$ (Step 2). The first inequality in the condition ensures that either *r* is sufficiently stable or the job is almost finished, while the second limits the excessive growth of I_r^j compared to the job length. The latter can particularly be important for short jobs, for which the first condition almost always evaluates to true. On the other hand, when the above mentioned inequalities are not satisfied, it seems to be desirable to decrease I_r^j and thus to perform Checkpointing more frequently $I_r^{j_{new}} = I_r^{J_{old}} - I$ (Step 3). When reducing the Checkpointing interval, the following constraint should be taken into account: $C < I_{min} \leq I_r^{j_{new}}$. I_{min} is a predefined between consecutive checkpoints is never less overhead added by each checkpoint. In case of stable grid systems, it is desirable to choose relatively large values for I_{min} (5 percent-10 percent of the total job length) to prevent an undesirably steep decrease of the Checkpointing interval. Experiments have shown that gradually incrementing I_r^j by I ensures rapid achievement of I_{opt} in most distributed environments. However, in case of rather reliable grids, the calibration of I_r^j can be accelerated by replacing I with a desirable percentage of the job execution time.

Therefore, the total grid resource availability, which is the percentage of time during which the resource performed useful computations, can be defined as

$$A_r = \left(\left(1 - \left(\sum_{n=1}^{N} (t_{r,n}^f - t_{r,n}^r)/T_{sim}\right)\right) \times 100\right),$$

Where N is the number of resource failures; $t_{r,n}^f$ and $t_{r,n}^r$ is respectively the time stamp of the resource failure and restore; and T_{sim} is the total simulation time. From the individual resource availability, total grid availability is computed as follows:

$$A_{grid} = \left(\left(\sum_{r=1}^{R} A_r\right) / (T_{sim} \times R)\right) \times 100,$$

Where R is the number of resources in the grid. Finally, SimGrid provides a set of events to specify network links and routes (sequence of links), which form the network model of the simulator.

4. REPLICATION

Similar to deciding upon the best Checkpointing interval, finding a generally applicable procedure to calculate the optimal number of job replicas is a complicated issue. Nowadays, most of the replication-based fault-tolerant algorithms assume a fixed number of job duplicates. However, dynamic solutions have recently started to receive attention.

In [3], a dynamic replication-based method is described, called Work queue with Replication (WQR). Initially, this algorithm distributes a single copy of a job to random idle resources in First Come, First Served (FCFS) order. When the job queue is empty and the system has free resources, replication is activated to cope with varying availability of hosts. The disadvantage of this "delayed-copy" approach is that if a system is heavily loaded for a long period, which is often the case in large scientific or production grids, the replication will be significantly delayed or not activated at all. Most of the failures in distributed environments end to occur during peak hours. [7] "Performance Implications of Failures in Large Scale Cluster Scheduling" when the WQR failure prevention is turned off by definition. Whereas this algorithm dynamically vary the number of job replicas dependent on the system load, the group-based approach determines the amount of replicas taking into account the reliability of each volunteer group, which is a group of resources with similar properties.

4.1 LOAD -DEPENDENT REPLICATION FT ALGORITHM

Providing fault tolerance in distributed environments through replication has as an advantage that otherwise idle resources can be utilized to run job copies without significantly delaying the execution of the original job. Obviously, the more job copies are running on the grid, the larger is the chance that one of them will execute successfully. On the other hand, running additional replicas on a distributed environment with an insufficient number of free resources can considerably reduce throughput and prolong job execution. To deal with this dilemma, this algorithm considers the system load and postpones or reduces replication during peak hours. The algorithm requires a number of parameters to be provided in advance, i.e., the minimum Rep_{min} and maximum Rep_{max} number of job copies, and the CPU limit (CL). The latter parameter specifies the lower bound on the number of active free CPUs for replication to take place. An example of the heuristic operation is shown in Fig. 3, where the required parameters are initialized as follows: Rep_{min} and Rep_{max} are set respectively to 1 and 2; and CL is equal to two CPUs. In each iteration, the GSched consults the IS for the system status (Step 1). Based on this information, CA and CL are compared, where CA is the number of active CPUs able to execute the next job. The outcome of the comparison determines the choice for the next job to be scheduled:

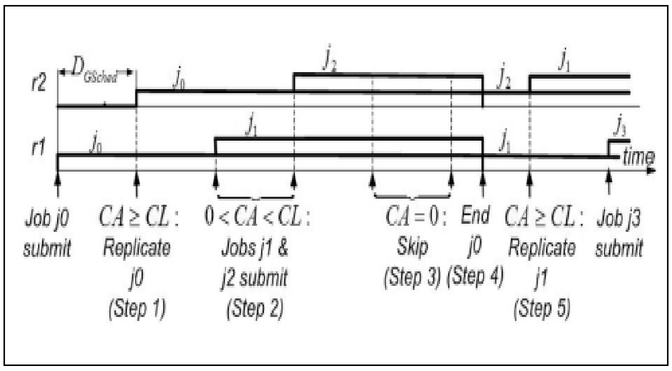

r2
r1
D_{GSched}
time
Job j0 submit
CA ≥ CL: Replicate j0 (Step 1)
0 < CA < CL: Jobs j1 & j2 submit (Step 2)
CA = 0: Skip (Step 3)
End j0 (Step 4)
CA ≥ CL: Replicate j1 (Step 5)
Job j3 submit

Fig-3 Operation of LoadDependentRep on a distributed environment consisting of two resources, each able to run two jobs simultaneously. Rep_{max} = 2, Rep_{min} = 1 and CL = 2

- ***CA ≥ CL.*** Select a job j with the earliest arrival time stamp and the number of active replicas less than Rep_{max} (Step 1).
- ***0 < CA < CL.*** Select a job j with the earliest arrival time stamp and the number of active replicas less than Rep_{min} (Step 2).
- ***CA = 0.*** Skip the current scheduling round (Step 3).

However, even if the grid system is heavily loaded, it can be desirable to consider Rep_{min} > 1, since the failure rate of resources in distributed environments increases with the intensity of the workload running on them. When one of the job duplicates finishes, other replicas are automatically canceled (Step 4). If the system load decreases before the job was executed, the remaining Rep_{max} - Rep_{min} replicas are activated (Step 5).

This algorithm assigns the selected job j to the site S with some free resources and with the smallest number of j replicas (Step 1, Step 5), since spreading replicas over different sites increases the probability that one of them will be successfully executed. If multiple sites have an equal number of job copies, a site that can provide for the fastest job execution is preferred. The speed or capacity of a site is defined as

$$\text{Speeds} = \left(\sum_{r \in S} MIPS_r\right) / \left(\left(\sum_{r \in S} n_r\right) + 1\right),$$

Where Million Instructions per Second ($MIPS_r$) is the speed of *r* and n_r is the number of jobs on *r*. In the above equation, only resources executing no other replicas of *j* are taking into account. Therefore, inside the chosen site, the job will be submitted to the fastest available resource with no identical job replicas. If no such resource exists, the distribution of j is postponed, and the next job from the GSched queue is scheduled. The resource speed is determined by

$$\text{Speedr} = MIPS_r / (n_r + 1),$$

4.2 FAILURE DETECTION AND LOAD DEPENDENT REPLICATION ALGORITHM

To increase the fault tolerance of the LoadDependentRep heuristic, this approach is combined with a failure-detection technique. The principle of failure detection is straightforward: as soon as a resource failure is discovered by the GSched, all jobs submitted to the failed resource are redistributed. The algorithm proceeds as LoadDependentRep, except that in each scheduling round, not only newly arrived jobs are considered for submission to a CR, but also all jobs distributed to failed nodes. This means that although the method offers a higher level of fault tolerance compared to solely replication-based strategies, it does not ensure job execution.

5. ADAPTIVE CHECKPOINTING AND REPLICATION-BASED FAULT TOLERANCE (COMBINED FT) ALGORITHM

In this section, a combined Checkpointing and adaptive replication-based scheduling approach is considered that dynamically switches between both techniques based on runtime information on system load. The algorithm can be particularly advantageous for grids with frequent or unpredictable alternations between peak hours and idle periods. In the *first case,* replication overhead can be avoided by switching to Checkpointing, while in the *second case;* the Checkpointing overhead is reduced by using low-cost replication. An example of the CombinedFT heuristic operation is shown in Fig.4, where the required parameters are initialized as follows: Rep_{max} and Rep_{min} are set respectively to 1 and 2; and CL is equal to two CPUs.

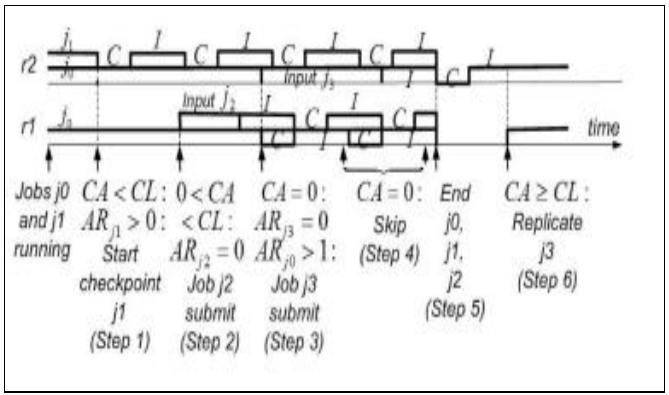

r2
r1
time
Input j_2
Input j_3
Jobs j0 and j1 running
CA < CL: $AR_{j1} > 0$: Start checkpoint j1 (Step 1)
0 < CA < CL: $AR_{j2} = 0$ Job j2 submit (Step 2)
CA = 0: $AR_{j3} = 0$ $AR_{j0} > 1$: Job j3 submit (Step 3)
CA = 0: Skip (Step 4)
End j0, j1, j2 (Step 5)
CA ≥ CL: Replicate j3 (Step 6)

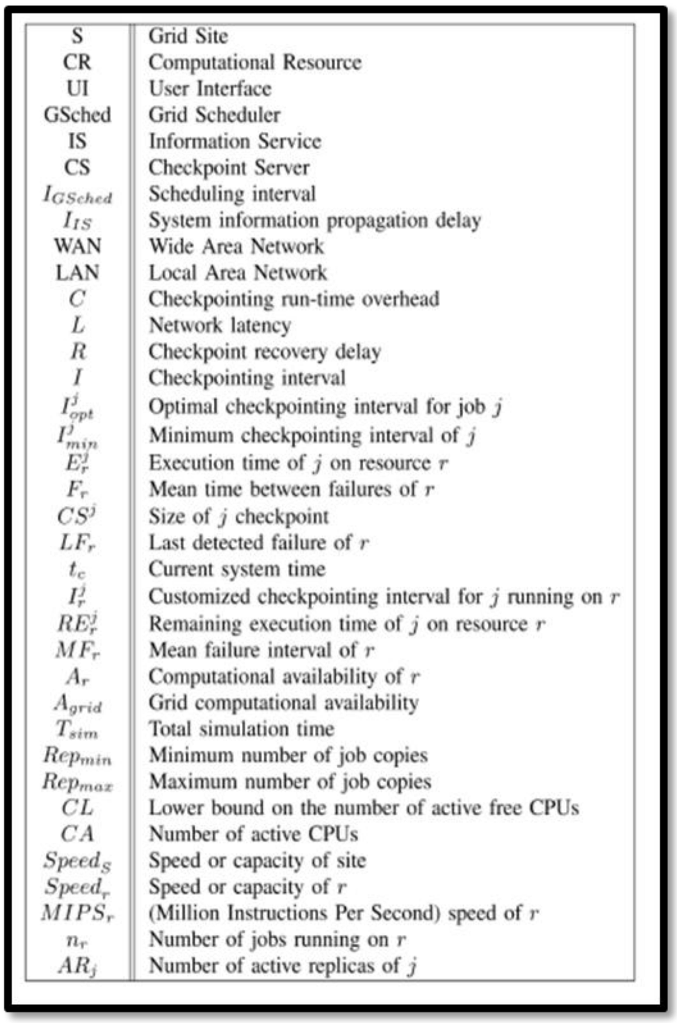
Fig-4 Operation of CombinedFT on a distributed environment consisting of two resources, each able to run two jobs simultaneously. Rep_{max} = 2, Rep_{min} = 1 and CL=2. The PeriodicCP method is applied in the Checkpointing mode

When the CPU availability is low (CA < CL), the algorithm is in Checkpointing mode (Step 1). In this mode, CombinedFT rolls back, if necessary, the earlier distributed active job replicas *(ARj)* and starts job Checkpointing. When processing the next job j, the following situations can occur:

- *ARj* > 0. Start Checkpointing the most advanced active replica and cancel the execution of other replicas (Step 1).
- *ARj= 0* and *CA > 0.* Start j on the least loaded available resource within the least loaded site, determined respectively by (2) and (1) (Step 2).
- *ARj = 0* and CA = 0 and ∃i: *ARi* > 1. Select a random replicated job i if any, start Checkpointing its most advanced active replica, cancel execution of other replicas of i, and submit j to the best available resource (Step 3).
- *ARj = 0* and CA = 0 and ¬∃i: *ARi* > 1. Skip the current scheduling round (Step 4).

The algorithm switches to replication mode when either the system load decreases or enough resources restore from failure (CA ≥ CL) (Step 5). In replication mode, all jobs with less than $\text{Rep}_{\max}$ replicas are considered for submission to the available resources, in the order defined by the Failure DependentRep algorithm. When a job *j* is selected, it is assigned to the fastest resource (with no similar job replicas) connected to a grid site *S* with the maximum Speeds and the smallest number of identical replicas. If *j* was previously in Checkpointing mode and the replication completed successfully, the Checkpointing of j is switched off (Step 6).

Table -1: List of uses symbols and acronyms



S	Grid Site
CR	Computational Resource
UI	User Interface
GSched	Grid Scheduler
IS	Information Service
CS	Checkpoint Server
I_{GSched}	Scheduling interval
I_{IS}	System information propagation delay
WAN	Wide Area Network
LAN	Local Area Network
C	Checkpointing run-time overhead
L	Network latency
R	Checkpoint recovery delay
I	Checkpointing interval
I^j_{opt}	Optimal checkpointing interval for job j
I^j_{min}	Minimum checkpointing interval of j
E^j_r	Execution time of j on resource r
F_r	Mean time between failures of r
CS^j	Size of j checkpoint
LF_r	Last detected failure of r
t_c	Current system time
I^j_r	Customized checkpointing interval for j running on r
RE^j_r	Remaining execution time of j on resource r
MF_r	Mean failure interval of r
A_r	Computational availability of r
A_{grid}	Grid computational availability
T_{sim}	Total simulation time
Rep_{min}	Minimum number of job copies
Rep_{max}	Maximum number of job copies
CL	Lower bound on the number of active free CPUs
CA	Number of active CPUs
$Speed_S$	Speed or capacity of site
$Speed_r$	Speed or capacity of r
$MIPS_r$	(Million Instructions Per Second) speed of r
n_r	Number of jobs running on r
AR_j	Number of active replicas of j

6. EXPERIMENTAL SETUP

The Grid Simulator used is SimGrid (a modified version 5.0) which has rich set of simulation facilities empowers to develop, evaluate scheduling and Fault Tolerance algorithms for heterogeneous distributed computing environments. Here, the no. of jobs =40, Total No. of P.E = 15 (3 nodes * 5 P.E). The no. of failure varies between = (4 to 9) P.E Where, P.E - Processing Elements.

7. SIMULATION RESULTS

No. of jobs =40, Total No. of P.E = 15 (3 nodes * 5 P.E)

No. of failure varies between (4 to 9) P.E

Table-2: Shows the Tabulated Result of Checkpointing and Replication with Combined FT

Parameter	LastFailureCP	MeanFailureCP	MFCP-Migration	Checkpoint-Replication	CP-Migration-Replication (CFT)
Avg.Checkpoint Interval(ms)	13.18	659.25	621.46	15.3	**627.125**
Avg.Waiting Time(ms)	23.95	5.77	4.31	4.08	**4.277**
Total Grid Resource Availablity (%)	14	26	11	37	28
Resource Speed (MIPS)	4004117	3626363	3341351	3156621	2663128

From the above statistical result it shows that Checkpoint with Migration and Replication (CombinedFT) has better Checkpoint Interval Time which considerably reducing the Average Waiting Time for the job. For Heavy Load Condition, the fully fault-tolerant Mean FailureCP results in the best system through-put compared to other heuristics. Whereas CP-Replication provides for costless fault tolerance with reduced Average Waiting Time. The preference for one of the solutions depends on particular system characteristics, such as load, job submission patterns, and failure frequency

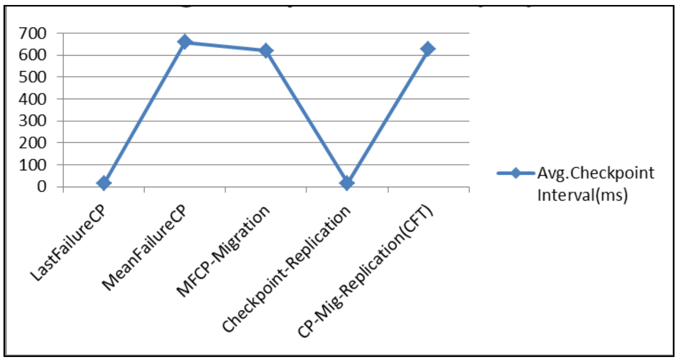


Fig-5 Average Checkpoint Interval

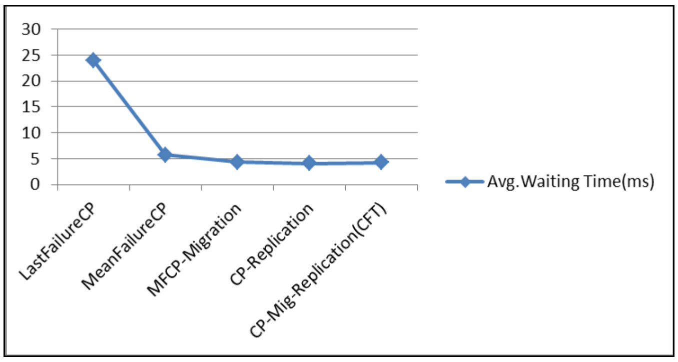


Fig-6 Average Waiting Time

8. CONCLUSIONS AND FUTURE WORKS

The problem addressed is fault tolerance in terms of resource failure in distributed environments. Thus the literature survey results achieves fault tolerance by dynamically adapting the Checkpointing frequency by Mean-Failure Dependent CP based on history of failure information and job execution time, which reduces Checkpoint overhead, and increases the throughput. Furthermore, adaptive Replication-Based Load Dependent FT can provide for the fastest job execution and throughput by dynamically varies the system load with low cost. The above two adaptive approaches are combined which improve system performance, while the preference for one of the solutions depends on particular system characteristics, such as load, job submission patterns, and failure frequency.

This work can be further extended to scheduling approach by dynamically changing estimations of job execution time which gives high job throughput in the presence of failure while reducing the system overhead.

REFERENCES

[1] Y. Li and M. Mascagni, “Improving Performance via Computational Replication on a Large-Scale Computational Grid,” Proc. Third Int’l Symp. Cluster Computing and the Grid (CCGrid ’03), May 2003.

[2] A. Legrand, L. Marchal, and H. Casanova, “Scheduling Distributed Applications: The SimGrid Simulation Framework,” Proc. Third Int’l Symp. Cluster Computing and the Grid (CCGrid ’03), May 2003.

[3] D. Silva, W. Cirne, and F. Brasileiro, “Trading Cycles for Information: Using Replication to Schedule Bag-of-Tasks Applicationson Computational Grids,” Proc. Int’l Conf. Parallel and Distributed Computing (Euro-Par ’03), pp. 169-180, Aug. 2003.

[4] Ying Z, Crishnendu C. “Energy-Aware Adaptive Checkpointing in Embedded Real-Time Systems”[C], *Proc. of the design, automation and test in Europe conference and exhibition (DATE’03), 2003.*

[5] T. Ozaki, T. Dohi, H. Okamura, and N. Kaio, “Min-Max Checkpoint Placement under Incomplete Failure Information,” Proc. Int’l Conf. Dependable Systems and Networks (DSN ’04), June-July 2004.

[6] S. Agarwal, R. Garg, M. Gupta, and J. Moreira, “Adaptive Incremental Checkpointing for Massively

Parallel Systems," Proc. 18th Ann. Int'l Conf. Supercomputing (SC '04), Nov. 2004.

[7] Y. Zhang, M. Squillante, A. Sivasubramaniam, and R. Sahoo, "Performance Implications of Failures in Large-Scale Cluster Scheduling," Proc. 10th Workshop Job Scheduling Strategies for Parallel Processing (JSSPP '04), pp. 233-252, 2004.

[8] R. De Camargo, A. Goldchleger, F. Kon, and A. Goldman, "Checkpointing-Based Rollback Recovery for Parallel Applications on the InteGrade Grid Middleware," Proc. Second Workshop Middleware for Grid Computing (MGC '04), pp. 35-40, 2004.

[9] A. Oliner, R. Sahoo, J. Moreira, and M. Gupta, "Performance Implications of Periodic Checkpointing on Large-Scale Cluster Systems," Proc. 19th IEEE Int'l Parallel and Distributed Processing Symp. (IPDPS '05), Apr. 2005.

[10] Global Grid Forum. Architecture for grid checkpoint and recovery (GridCPR) services and a GridCPR Application Programming Interface. GridCPR-WG, September 2005.

[11] Y. Xiang, Z. Li, and H. Chen, "Optimizing Adaptive Checkpointing Schemes for Grid Workflow Systems," Proc. Fifth Int'l Conf. Grid and Cooperative Computing (GCC '06), Oct. 2006.

[12] A. Ziv and J. Bruck, "Performance Optimization of Checkpointing Schemes with Task Duplication," IEEE Trans. Computers, vol. 46, no. 12, pp. 1381-1386, Dec. 2006.

[13] C. Bossie and P. Fiorini, "On Checkpointing and Heavy-Tails in Unreliable Computing Environments," SIGMETRICS Performance Evaluation Rev., vol. 34, no. 2, pp. 13-15, 2006.

[14] P. Katsaros, L. Angelis, and C. Lazos, "Performance and Effectiveness Trade-Off for Checkpointing in Fault-Tolerant Distributed Systems," Concurrency and Computation: Practice and Experience, vol. 19, no. 1, pp. 37-63, 2007.

BIOGRAPHIE

NAKKEERAN.M, working as an Assistant Professor in the Dept. of CSE, SVS College of Engineering, Coimbatore for 4 years to till date. I have completed my B.E (EEE) from GCE, Tirunelveli with First Class and M.E (CSE) in Annamalai University, Chidambaram with First Class (Distinction). I have published 11 research papers in International Journals, International Conference and National Conference in the area of Parallel and Distributed Computing, Wireless Sensor Network. And my area of interest includes Grid Computing, Cloud Computing, Green Computing and Mobile Computing.