

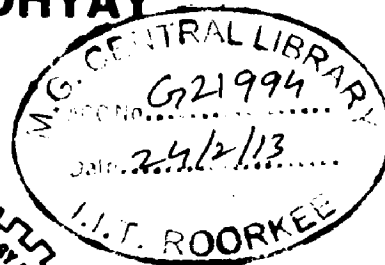
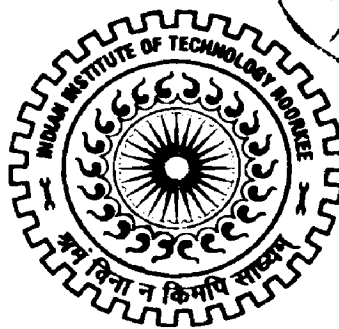
# ADAPTIVE CHECKPOINTING BASED FAULT TOLERANCE IN GRID ENVIRONMENT

## A DISSERTATION

*Submitted in partial fulfillment of the  
requirements for the award of the degree  
of*  
**MASTER OF TECHNOLOGY**  
*in*  
**COMPUTER SCIENCE AND ENGINEERING**

By

**NEERAJ UPADHYAY**



**DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING  
INDIAN INSTITUTE OF TECHNOLOGY ROORKEE  
ROORKEE - 247 667 (INDIA)  
JUNE, 2012**

## CANDIDATE'S DECLARATION


---

I hereby declare that the work, which is being presented in the dissertation entitled “**Adaptive checkpointing based fault tolerance in grid environment**” towards the partial fulfillment of the requirement for the award of the degree of **Master of Technology in Computer Science** submitted in the Department of Electronics and Computer Engineering, Indian Institute of Technology Roorkee, Roorkee, Uttarakhand (India) is an authentic record of my own work carried out during the period from July 2011 to June 2012, under the guidance of **Dr. Manoj Misra, Professor**, Department of Electronics and Computer Engineering, IIT Roorkee.

The matter presented in this dissertation has not been submitted by me for the award of any other degree of this or any other Institute.

Date: 12-06-12

Place: Roorkee

  
(Neeraj Upadhyay)

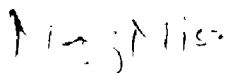
---

## CERTIFICATE

This is to certify that the above statement made by the candidate is correct to the best of my knowledge and belief.

Date: 12-06-12

Place: Roorkee

  
(Dr. Manoj Misra)

Professor

Department of Electronics and Computer Engineering

IIT Roorkee.

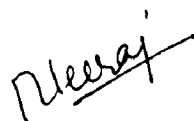
## ACKNOWLEDGEMENTS

---

First and foremost, I would like to extend my heartfelt gratitude to my guide and mentor **Dr. Manoj Misra**, Professor, Department of Electronics and Computer Engineering, Indian Institute of Technology Roorkee, for his invaluable advices, guidance, encouragement and for sharing his broad knowledge. His wisdom, knowledge and commitment to the highest standards inspired and motivated me. He has been very generous in providing the necessary resources to carry out my research. He is an inspiring teacher, a great advisor, and most importantly a nice person.

I am greatly indebted to all my friends, especially Abhishek Bafna, who have graciously applied themselves to the task of helping me with ample moral supports and valuable suggestions.

On a personal note, I owe everything to the Almighty, my parents and my sister. The support which I enjoyed from my father, mother and my sister provided me the mental support I needed.

  
**NEERAJ UPADHYAY**

## **Abstract**

Grid systems differ from traditional distributed systems in terms of their large scale, heterogeneity and dynamism. These factors contribute towards higher number of fault occurrences as large scale causes lower values of Mean Time To Failure (MTTF), heterogeneity results in interaction faults (protocol incompatibilities) between communicating disparate nodes and dynamism implies dynamically varying resource availability due to resources autonomously entering and leaving the grid and thus effecting the jobs running on them. Another factor that increases probability of failure of applications is that applications running on grid are long running computations taking days to finish. Traditional approaches for tolerating faults in distributed systems include checkpointing and replication. Incorporating fault tolerance in scheduling algorithms is one of the approaches for handling faults in grid environment. Genetic Algorithms and Ant Colony Optimization are a popular class of meta-heuristic algorithms used for grid scheduling. This work designs heuristics for adaptive checkpointing based on fault information about resources. These heuristics have been incorporated in GA and ACO. Other adaptive checkpointing techniques developed focus on online adaption of checkpoint interval based on MTBF, last failure time and fault indexes of resources. Performance comparison of adaptive checkpointing with periodic checkpointing techniques have been performed using simulated Grid environment for wide range of scenarios such as temporally and spatially correlated failures, real failure traces and real workload traces. Adaptive checkpointing techniques are found to give superior performance compared to periodic checkpointing.



# Table of Contents

1. Introduction.....	1
1.1 Introduction.....	1
1.2 Motivation.....	1
1.3 Problem Statement.....	2
1.4 Thesis Organization.....	3
2. Background and Literature Review	
2.1 Grid Computing.....	4
2.2 Fault Tolerance in Grid Environment	
2.2.1 Fault Detection and Handling Techniques.....	6
2.3 Research Gaps.....	18
3. Proposed Work	
3.1 Batch Mode Scheduling and Fault Model.....	20
3.2 Fault Tolerance Support in Metaheuristics.....	21
3.2.1 Genetic Algorithm based Fault Tolerance Techniques.....	21
3.2.2 Ant Colony Optimization based Fault Tolerance.....	29
3.3 Adaptive Checkpointing based Fault Tolerance Techniques.....	30
3.3.1 Fault Index based Periodic Skip.....	31
3.3.2 MTBF and Last Failure based Adaptive Checkpointing for correlated failures.....	32
4. Simulation Environment and Implementation details.....	34
5. Performance Evaluation and Experimental Results	
5.1 Performance Metrics.....	43
5.2 Simulation Parameters.....	44
5.3 GA-based Adaptive Fault Tolerance Using MTBF of Resources.....	45

5.4 GA-based Adaptive Fault Tolerance Using Fault Ratios.....	51
5.5 ACO-based Fault Tolerance Using MTBF of Resources.....	54
5.6 ACO-based Adaptive Fault Tolerance Using Fault Ratios.....	57
5.7 GA-based Adaptive Fault Tolerance Using Fault Ratios for Correlated Failures.....	59
5.8 ACO-based Adaptive Fault Tolerance Using Fault Ratios for Correlated Failures.....	64
5.9 Fault Index Based Periodic Skip.....	68
5.10 Adaptive Checkpointing Using MTBF and Last Failure times of resources.....	73
5.11 Performance Comparison of GA-based Checkpointing Techniques Using Failure Traces .....	77
5.12 Performance Comparison of ACO-based Checkpointing Techniques Using Failure Traces.....	82
5.13 Performance Comparison of GA-based Checkpointing Techniques Using Workload Traces.....	87
5.14 Performance Comparison of ACO-based Checkpointing Techniques Using Workload Traces.....	103
6. Conclusions and Future Work.....	118
References.....	121

## List of Figures

2.1 Illustration of how real ants find shortest path.....	15
2.2 ACO Algorithm.....	15
3.1 Batch mode scheduling.....	20
3.2 Genetic Algorithm.....	21
3.3 Representation of Chromosome.....	22
3.4 Stochastic Universal Sampling.....	23
3.5 Single point Crossover.....	23
3.6 Last Failure Time based checkpointing.....	26
3.7 Fault occurrence history table .....	28
4.1 Components of Grid environment.....	34
4.2 Event diagram to represent interaction between different GridSim entities.....	36
4.3 Modules hierarchy used for experimentation of proposed techniques.....	39
5.1 – 5.7 Comparison between GA- based adaptive checkpointing using MTBF and GA-based periodic checkpointing.....	(44 - 47)
5.8 -5.14 Comparison between GA- based adaptive checkpointing using Fault ratios and GA-based periodic checkpointing.....	(48 – 50)
5.15 – 5.21 Comparison between ACO- based adaptive checkpointing using MTBF and ACO-based periodic checkpointing.....	(51 – 53)
5.22 – 5.28 Comparison between ACO- based adaptive checkpointing using Fault ratios and ACO-based periodic checkpointing.....	(53 – 55)
5.29 – 5.38 GA-based techniques comparison for spatially and temporally correlated failures.	(57 – 60)

5.39 – 5.48 ACO-based techniques comparison for spatially and temporally correlated failures. .....	(61 – 64)
5.49 – 5.53 Fault Index based periodic skip vs periodic skip.....	(65 – 66)
5.54 – 5.62 Fault Index based periodic skip vs periodic skip for spatially and temporally correlated failures.....	(67 – 69)
5.63 – 5.70 Adaptive Checkpointing using MTBF and Last failure times of resources...	(70 – 72)
5.71 – 5.82 GA- based techniques comparison for failure traces.....	(73 – 78)
5.83 – 5.94 ACO- based techniques comparison for failure traces.....	(79 – 83)
5.95 – 5.133 GA- based techniques comparison for workload traces.....	(84 – 99)
5.134 – 5.172 ACO- based techniques comparison for workload traces.....	(99 – 113)

# Chapter 1

## Introduction and Problem Statement

---

### 1.1 Introduction

In past few decades computer network technology has taken a big leap with channel capacity no longer being a barrier to distributed computing. This same period saw the emergence of highly compute and data intensive applications with resource requirements hard to be fulfilled by a single computer, cluster or even a supercomputer. All these factors lead to the inception of a new paradigm called high performance distributed computing (HPC). Grid computing is one such distributed HPC environment.

The vision of Grid computing is a computing environment as pervasive as electric Grid where desired resources as electric power in electric grids would be available on-demand and in the desired amount making computational Grid a utility similar to electric Grid. The Grid as it exists today collaborate distributed computing resources with resource owners contributing their idle CPU cycles. These idle cycles are used for running tasks of massively parallel scientific applications. Thus today's Grid can be seen as an ensemble of resources from multiple domains on a large scale to make a distributed supercomputer (with capabilities larger than a supercomputer). Grid is a popular platform for running scientific applications such as weather forecasting, drug design, multimedia applications, physics particle accelerator experiments etc.

However maintaining such a massive infrastructure while it continues to provide non trivial qualities of service [4] is not that easy. Several issues such as security, job scheduling, load balancing, failover techniques etc have to be dealt with. This dissertation report focuses on one of the most important issues – **fault tolerance in Grid environment**, so that QoS can be maintained despite resources failing during execution of jobs on them.

### 1.2 Motivation

Grid systems differ from other distributed systems (cluster, peer to peer) in terms of their large scale, heterogeneity and dynamism. Scale of grid infrastructure, consisting of thousands of computational nodes, storage devices, affects its reliability. Since the reliability of a system is a product of the reliabilities of its components, as the complexity (scale) of the system increases its

reliability and its Mean Time to Failure (MTTF) decreases. Computing resources in grid are highly heterogeneous with varying hardware and software architectures. Heterogeneity increases chances of interaction faults occurring between disparate Grid nodes. Dynamicity – resources may enter and leave at any time, dynamically varying resource load - causes loss and delay of executing jobs. Grid resources are managed in different administrative domains each with its own access, security policies. So following properties hold for Grid resources [1].

1. There may be no guarantee that a resource is non-malicious.
2. There may be no guarantee that a resource is reliable (reliability of resources' hardware and software).
3. There may be no guarantee of processing power (we may have no control over resource's scheduling policy and its load).

Another factor which further increases chances of job failure in grid environment is that majority of the applications running on grid are compute-intensive requiring several days of computation and large number of resources. Consider a Grid application requiring 50 computational resources each with MTTF of 100 days and requires a week for computation. If the failure mode of resources is exponentially distributed then the composed application has MTTF of 2 days. So in the absence of any fault tolerance the application would rarely finish.

All the above factors necessitate the application of fault tolerant techniques for grid computing.

### **1.3 Problem Statement**

Problem statement: *“Design and performance study of adaptive checkpointing based fault tolerance techniques in Grid environment”*

The work done in this dissertation can be subdivided into two major tasks:

- Extending the metaheuristic algorithms such as Genetic Algorithm and Ant Colony Optimization which are commonly used for job scheduling in Grid environment with support for fault tolerance techniques such as checkpointing
- Inventing adaptive approaches for fault tolerance in computational grids by suitable modifying traditional approaches such as checkpointing to take into account various

characteristics of the grid environment. These adaptive approaches would be responsive to the present conditions such as frequency of resource failures, remaining time of completion of jobs.

## **1.4 Thesis Organization**

This thesis is organized into six chapters including this one which gives introduction, motivation and problem statement.

Chapter two gives background of available fault tolerance techniques in Grid environment.

Chapter three describes the proposed work.

Chapter four gives description of simulation environment and implementation details.

Chapter five gives performance comparison of proposed technique with available techniques and also experimental results are presented.

Chapter six concludes the dissertation and gives directions for future work.

## Chapter 2

### Background and Literature Review

---

#### 2.1 Grid Computing

In [2] Grid computing is defined as “A computational grid is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities.” Grid thrives to achieve pervasiveness of electric Grid where desired resources like electric power in electric grids would be available on-demand and in the desired amount. Another definition of Grid computing [3] defines it as “coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organization.” Resources shared comprise not only the computing power but memory, secondary storage, data, special equipments, etc. This sharing of resources occurs in a highly controlled and coordinated environment where consumers (resource users) and resource providers clearly define the sharing rules of what is shared, who is allowed to share and conditions under which this sharing takes place. These individuals/organizations/institutions bounded by sharing rules together constitute a virtual organization (VO). Example of a VO can be a set of physicists working on high energy physics (particle accelerators) experiment or some astronomical experiments such as study of galaxy formation. These individuals collaborate together to form a coordinated resource sharing environment. Ian Foster gives a three point checklist [4] for deciding whether a particular infrastructure is a Grid. To be classified as a Grid an infrastructure should have three important features:

##### 1) Coordination, collaboration of heterogeneous resources under Distributed control

A Grid is a decentralized environment where different organizations collaborate to provide a coordinated access to their resources. Organizations participating in a Grid are autonomous and have exclusive control over management of their resources. Resources owned by organizations can be highly heterogeneous ranging desktops, workstations, clusters, supercomputers etc .Grid deals with various issues arising in this “distributed control” environment such as security, membership management, pricing policy, fault tolerance, scheduling, load balancing, SLA management and synchronization.

##### 2) Standard, open and general purpose protocols



Services provided by Grid needs to be standard and open for interoperability in a heterogeneous environment.

### 3) Non trivial qualities of service

From feasibility aspect a Grid should deliver non trivial qualities of service. The QoS requirement may be related to response time, turnaround time, system utilization, waiting time etc.

## 2.2 Fault Tolerance in Grid Environment

Some general basic definitions as given in [5] are:

**Error:** A deviation of the internal or external state of the system from correct state. For ex. Memory faults may cause the state of memory cells to change from 0 to 1 or vice versa.

**Fault:** The adjudged or hypothesized cause of error.

**Failure:** The event that occurs when delivered service deviates from correct service.

In [6] six classes of faults that may be present in Grid environment are discussed: Hardware faults (CPU, memory, storage faults, components used beyond specification), Application and operating system faults (memory leaks, resource unavailable), Network faults (packet loss, packet corruption, network congestion), Software faults (unhandled exception, unexpected input), Response faults (Byzantine error), Timeout faults.

Other fault types are [1]: life-cycle faults (versioning faults), interaction faults (protocol incompatibilities). According to [1] interaction, timing, and omission faults (the resource omits the response) are more prevalent in Grid environment. In [7] taxonomy of Grid faults is presented. One categorization based on duration is:

**a) Transient faults:** Transient faults cause applications, resources to malfunction for some period and then disappear. Example: failed job due to insufficient disk space, machine reboot.

**b) Permanent faults:** Permanent faults cause failure of a component/resource for an undefined period. Resource remains unavailable for the duration of job and thus cannot be used for execution of job.

**c) Intermittent faults:** These faults fluctuate between active (causing malfunction) and inactive (normal functioning). For example load of a resource can change dynamically and this may affect the execution of jobs on that resource.

Faults have a tendency to cause loss of information [8]. Recovery from faults is based on redundancy. Redundancy can be of three types – spatial, temporal and information redundancy. In spatial redundancy multiple copies of jobs are executed on different nodes. Temporal redundancy involves multiple executions of job on same resource with different executions skewed in time. Information redundancy involves storing additional information during execution of job and this information is used during recovery.

As given in [9] the function of fault tolerance is “...to preserve the delivery of expected services despite the presence of fault-caused errors within the system itself. Errors are detected and corrected, and permanent faults are located and removed while the system continues to deliver acceptable service.” Fault tolerance increases the dependability of the system. Dependability is defined as: “The ability to avoid service failures that are more severe than it is acceptable to the users” [5]. Dependability encompasses a set of attributes: Availability, Reliability, Robustness, Safety, Integrity and Maintainability. Availability is the probability that the system is in correct state at a given time. Reliability is the probability of failure in a given interval of time. Other terms related to reliability are mean time to failure (MTTF), Mean Time between Failures (MTBF) and Mean time to repair (MTTR).

In [10] it has been studied that time between failures (MTBF) is best modeled by Weibull distribution with decreasing hazard rate (Weibull shape parameter 0.7-0.8) i.e. frequency of failures decreases with time. Mean repair time (MTTR) is well modeled by lognormal distribution. Failures tend to cluster in time and are caused by relative small set of computational nodes i.e. failures are temporally and spatially correlated.

## **2.2.1 Fault detection and handling techniques**

### **2.2.1.1 Fault Detection**

Implementation of failure detectors is based on the concept of notifications and timeouts. There are two models for interaction between the failure detector and monitored components: pull

model, push model. In push model the monitored component periodically sends heartbeat messages to failure detector. If message is not received within a certain time interval  $T$  the component is suspected as failed. In pull model the monitor periodically sends liveness messages (“Are you alive?” messages) to monitored components and suspects crash if no response arrives within a certain time interval. In [11] are presented problems that should be considered while implementing failure detectors for grid system. Some of the problems are message explosion (overloading of network with failure detection related messages), scalability, message loss, flexibility (adapting to different types of applications).

### **2.2.1.2 Fault handling techniques**

Fault tolerance techniques in grid can be divided into pro-active and post-active approaches. In pro-active approach failure consideration is made before scheduling of the job and the job is dispatched with the hope that job does not fail. In [6] an agent oriented pro-active failure handling framework is presented where agents deal with individual faults proactively. Agents monitor various properties of jobs and resources such as memory consumption of jobs, MTBF of resources. Other examples of pro-active approaches are replication, fault tolerant scheduling. Post-active approaches handle failures after they have occurred. Checkpointing is one of the post-active approaches. In a complex dynamic environment such as Grid, where resource conditions changes dynamically and unpredictably (unpredictable fluctuations in load), post-active approaches have greater significance. Both pro-active and post-active approaches are complementary and can be used in conjunction with each other.

### **2.2.1.3 Checkpointing and recovery**

Checkpointing is a process of periodically saving the state of a running process to stable storage. Checkpointing allows a failed process to be restarted from its last checkpoint, bounding the amount of lost work to be recomputed. It is commonly used to ensure the progress of long running applications. In [12] are described three checkpointing strategies for concurrent, inter communicating processes: coordinated checkpointing, uncoordinated checkpointing, and communication-induced checkpointing.

In coordinated checkpointing, processes synchronize their checkpoints to form a consistent global state (consistent system state is one in which if a process’s state reflects a message receipt,

then the state of the corresponding sender reflects sending that message). Each process stores only one checkpoint on stable storage (the latest checkpoint) and the recovery is simplified as it only involves restart of each process from its last saved checkpoint. However, coordinated checkpointing has a disadvantage that each checkpointing operation incurs a large overhead. Since the number of checkpointing operations performed are much larger than the number of failures this overhead can be considerably high. Coordinated checkpointing can be either blocking or non-blocking.

In uncoordinated checkpointing each process takes checkpoints independent of other processes. Consistent global state is determined during recovery procedure. This strategy has several disadvantages. First, there is a possibility of domino effect which causes the system to rollback to the beginning of the computation. Second, each process needs to maintain multiple checkpoints and thus this strategy imposes greater storage requirements. Third, a process may take useless checkpoint that will never be part of a consistent global state.

In communication-induced checkpointing in addition to independent checkpoints each process takes some additional forced checkpoints to ensure the progress of global recovery line and thus avoiding domino effect. Information regarding forced checkpoints is piggybacked on application messages.

Uncoordinated checkpointing may be combined with message logging to avoid domino effect. Message logging protocols [13] are based on the Piece Wise Deterministic (PWD) assumption which means that a process's execution can be modeled as a sequence of deterministic state intervals, each interval initiated by a non-deterministic event. The non-deterministic event can be receipt of a message. In logging, information about received messages (determinant) is stored on stable media and later replayed to recover a lost state. Message logging protocols need to ensure that once a crashed process recovers its state is consistent with the state of other processes. This consistency requirement can be expressed as avoiding orphan processes, which are surviving processes whose state is inconsistent with the state of recovered process [13]. Message logging protocols can be categorized as pessimistic, optimistic, or causal.

In pessimistic protocol no process ever sends a message until all the messages delivered before sending it have been logged. Pessimistic protocols never create orphans and reconstructing the

state of the crashed process just involves replaying the logged messages. However pessimistic protocols block the process for each received message, even when no process ever crashes.

In optimistic protocols processes logs determinants asynchronously into stable storage. They are based on the assumption that logging will complete before failure occurs. Optimistic protocol doesn't require the process to block waiting for logging to complete but may create orphan processes, which complicates the recovery.

Causal logging protocols ensures that the determinant of each nondeterministic event that causally precedes the state of a process is either stored in stable storage or is available locally to that process. This ensures that no orphans are created. As in optimistic logging each process takes asynchronous checkpoints. The causality information is piggybacked with each message.

Grid environment with its scalability, heterogeneity and dynamism presents certain issues regarding application of checkpointing. As applications typically running on Grid consist of a large number of interacting tasks, the overhead of synchronization required in checkpointing protocols will be large. Storage overhead is also magnified. Techniques such as incremental checkpointing, which only saves the memory pages modified since last checkpoint, and probabilistic checkpointing [14], in which unit of checkpointing is a memory block which is smaller than a memory page, poses less storage and bandwidth overhead. Application level checkpointing [15] with portable checkpoint files is more suitable in a heterogeneous environment such as Grid compared to system level checkpointing.

#### **2.2.1.3.1 Adaptive Checkpointing**

Effectiveness of checkpointing techniques strongly depends on the length of checkpointing interval. Inappropriate checkpoint intervals can have serious performance implications [16]. If checkpointing interval is very high a large amount of work is lost on each failure. On the other hand a very low checkpoint interval incurs a high overhead as each checkpointing operation takes some time (checkpoint overhead). Also with larger overall checkpoint overhead the effective running time of application increases which increases chances of failure of the application [16]. Determining the optimum checkpoint interval is a major issue involved in deciding the usability of checkpointing technique. Plank [17] studied the applicability of theoretical equations for optimum checkpoint interval using data sets of three workstation

monitoring projects for simulation. The theoretical equations assume failures to be exponentially distributed whereas the actual failures in these data sets did not follow exponential distribution. There are several factors affecting the optimum checkpoint interval such as application characteristics, failure conditions of resources. The problem worsens in a highly dynamic Grid environment where failure conditions of resources may change dynamically. Recent studies on checkpointing have considered dynamic adaption of checkpoint interval. These studies have leveraged the dynamic information about resource conditions and application execution times in designing heuristics for online modification of checkpoint interval. One such technique is cooperative checkpointing [18, 19, and 20]. In cooperative checkpointing technique the application programmer, compiler and runtime system cooperatively decide when checkpoint operations are to be performed. This technique dynamically skips checkpoints if the expected cost of taking a checkpoint is greater than the expected cost if checkpoint is not taken. FT-Pro [21] includes an adaption manager which after periodic interval (adaption points) chose one of the three actions – skip checkpoint, take checkpoint, proactively migrate. The action chosen depends on the failure prediction results and the failure impact during next interval. In [22] are presented adaptive checkpointing strategies that dynamically adapt the checkpointing frequency depending on changing system properties (resource failure frequency, remaining execution time of jobs). It presents two adaptive checkpointing techniques – last failure dependent checkpointing and mean failure dependent checkpointing. The last failure dependent checkpointing technique leverage information about last failure time of resources and temporal correlation of faults (if a resource has not failed for a long time then there is less probability that it will fail in the near future) in skipping checkpoints. The mean failure dependent checkpointing technique adapts the checkpoint interval online depending on remaining execution time of job and mean failure time of resources. Mean failure dependent checkpointing technique is shown to give performance comparable to the optimal checkpoint interval for the simulation environment for all checkpoint intervals. In [23] is presented adaptive checkpointing technique for economy based Grids. The presented technique maintains fault indexes of resources to adaptively decide the checkpoint interval during job submission. Fault index is an indicator of vulnerability of resource towards failures. When a job misses deadline (or job fails due to resource failure) fault index of resource is incremented by one and when job completes successfully fault index is decremented by one.

#### **2.2.1.4 Replication (Over provisioning)**

Replication [24] involves running different replicas of the same task on different Grid resources simultaneously with the hope that at least one of them will complete execution successfully. Replication technique can be classified into two categories: Active replication and Passive replication. In active replication replicas service requests in parallel and their states are closely synchronized. In passive replication a primary replica services requests while other replicas are passive and can take over when primary fails. Efficiency of replication depends on determining the optimal number of replicas and can pose high overhead in highly loaded systems if the number is not optimal. In [22] is presented an adaptive load dependent replication technique which postpones replication when system load is high. In [25] is presented overloading of backups of primaries of independent and dependent tasks for efficient scheduling of backups.

#### **2.2.1.5 Task-level, workflow-level and hybrid fault tolerance techniques [26]**

Workflow applications are structured in directed acyclic graph form where each node represents a task and edges represent inter-task dependencies. In [26] two-level fault tolerance techniques are incorporated into the workflow structure – task-level and workflow-level. Task-level fault tolerance techniques deal with task crash failures. Retry, replication and checkpointing are three task-level fault tolerance techniques. Workflow level failure handling techniques modify the execution flow to deal with faults. These techniques enable handling of task-specific failures. It includes three techniques: alternate task, workflow-level redundancy and user-defined exception handling. In alternate task technique if one implementation of a task fails it is replaced with an alternate implementation with different execution characteristics. For example on memory full exception a fast implementation, with high memory consumption, is replaced with a slow implementation, with low memory consumption. Redundancy requires different implementations of a task to run in parallel. User-defined exception handling technique allows defining of a special treatment for different task-specific failures.

The above mentioned techniques can be combined to create hybrid techniques. For ex. Replication with checkpointing, alternate task with checkpointing, etc. The fault handling framework proposed in [26] is flexible in that different failure handling techniques can be specified depending on application and resource characteristics such as execution time of job,

MTTF of resource, downtime of resource. This is particularly beneficial for Grid due to its heterogeneity. Conclusions derived from experimental assessment in [26] reveal that checkpointing is more beneficial compared to retrying and replication for resources with low MTTF. Also replication performs better compared to retrying and checkpointing for resources with large downtimes. For low values of MTTF checkpointing and replication with checkpointing have superior performance. If resources are reasonably reliable replication performs best. For large downtimes replication and replication with checkpointing performs better than other techniques. However MTTF has greater effect than downtime and for low MTTF checkpoint performs better than replication. In case information about downtimes and MTTF is not available replication with checkpointing is the best technique.

### **2.2.1.6 Other related works**

#### **2.2.1.6.1 Mobile agent based fault tolerance**

Mobile agents are suited for the development of grid infrastructure [27] due to their properties of cooperation (mobile agents can interact and cooperate with each other), autonomy (autonomous entities with little intervention from client), heterogeneity (several mobile agent platforms can be executed in heterogeneous environments), reactivity (can react to external events such as resource availability variation) and mobility. In [27] fault tolerance components are developed as mobile agents to handle node and application crashes.

#### **2.2.1.6.2 Fault tolerance by scheduling**

##### **2.2.1.6.2.1 Fault tolerance scheduling of scientific workflows**

[28] [29] presents combined fault tolerance and scheduling techniques for workflow applications. In [29] two scheduling algorithms for heterogeneous systems, HEFT and DSH, are combined with checkpointing and replication, with checkpointing being used during workflow execution and over-provisioning used during scheduling and planning phase. Lightweight Checkpointing is done after completion of each task. If task fails to finish due to resource unavailability, it is migrated to most reliable resource. If some resources on which parent tasks were running are also not available, then those parent tasks are restarted on some available resource.

##### **2.2.1.6.2.2 Volunteer Availability based fault tolerant scheduling**



Desktop grid computing environment consists of clients, volunteers and volunteer servers [30]. A client submits jobs for execution. A volunteer donates its computing resources. A volunteer server manages jobs and volunteers. Volunteers can freely join and leave in the middle of executions. Also, volunteers are not completely dedicated for grid use. So public executions i.e. grid jobs get temporarily suspended by a private execution at the volunteer. These situations are termed as volunteer autonomy failures. The proposed scheduling algorithm in [30] handles volunteer autonomy failures by selecting an appropriate volunteer for task scheduling based on volunteer availability and volunteering time information maintained at it.

#### **2.2.1.6.2.3 Fault tolerant Genetic algorithms**

Genetic Algorithm is a global search technique used for solving optimization problems. Genetic algorithm use a population of search nodes (chromosomes or solutions or individuals) in its search and uses probabilistic transition rules [31]. Genetic algorithm consists of a representation for nodes in the search space, genetic operators for generating new individuals, fitness function for evaluating each solution, a selection criteria, probability factors for application of genetic operators and a termination condition. It maintains a pool of potential solutions called chromosomes. For grid scheduling a chromosome is a mapping from job to resource. Genetic algorithms produces new solutions by randomly combining the good features present in existing solutions. A genetic algorithm consists of following steps [31]:

- a) Initialization: initial population of chromosomes is randomly generated.
- b) Evaluation of fitness function: fitness value of each chromosome is calculated.
- c) Genetic operations: new chromosomes are generated by applying genetic operators to the chromosomes.
- d) Steps b and c are repeated until termination criterion is reached.

The termination criterion may be all chromosomes converging to the same fitness value or predefined number of iterations. Crossover and mutation are two genetic operators. Crossover operator randomly selects two chromosomes and chooses a random point in the first one (crossover point) and exchanges the sections of both chromosomes from crossover point to the end of each chromosome. Mutation randomly selects a chromosome and randomly selects a task

within the chromosome and randomly assigns a new machine to it. Application of crossover (mutation) operation is controlled by a crossover (mutation) probability.

Several research efforts incorporate fault tolerance in meta-heuristics such as Genetic algorithm, Ant colony optimization etc. In [32] risk-resilient genetic algorithm scheduling strategies are proposed. It considers three scheduling modes - preemptive mode, replicated mode and delay-tolerant mode. In [33] Resource Fault Occurrence History (RFOH) information is maintained in Grid Information Service (GIS) and this information along with response time is used for designing the fitness function of Genetic Algorithm. So chromosomes having good fitness values are more fault tolerant compared to the ones with lower values. In [34] is presented a genetic algorithm for job scheduling. It uses checkpointing technique to tolerate faults but doesn't give experimental results. In [35] a reliability aware genetic algorithm based scheduling is presented. It designs a fitness function combining makespan, flowtime, average time to release and reliability (modeled using exponential distribution) of the chromosome (schedule). In [36] Genetic Algorithm based scheduling strategies supporting different fault tolerance techniques such as retry, migration, replication and checkpointing are proposed. Experiment results presented shows that checkpointing based strategy has best performance considering the performance metrics – makespan, average turnaround time and job failure rate. A portion of our work that uses checkpointing differs from work in [36] in using adaptive checkpointing techniques and also their work focuses on permanent failures of resources and job migration on failures (availability of spare nodes) whereas our work only considers transient failures [7] of resources and the restart of jobs on the same resource when the resource recovers from failure.

#### **2.2.1.6.2.3 Ant Colony Optimization (ACO)**

Ant Colony Optimization [37] is one of the popular nature-inspired algorithms for solving optimization problems. In ACO a groups of cooperating agents cooperate to find optimal solutions to the problem. These ants indirectly communicate with each other using pheromone which is a form of distributed memory. Figure 2.1 shows how real ants find optimal paths to food source. At decision point two paths exist. Each ant randomly chooses one of the paths. As ants move along a path it deposits pheromone on the path followed. Now assuming that half of the ants follow one path and the other half second path, in a given period of time more ants will visit the lower path compared to the upper as lower path is shorter. After a short period of time

the difference between amounts of pheromone on the two paths will be sufficiently larger that next ants will, in probability favor the lower path compared to upper which with a positive feedback further increases the number of ants following the lower path. This cooperative behavior of ants inspired the development of ant system, which further leads to the development of more efficient system called ACO.

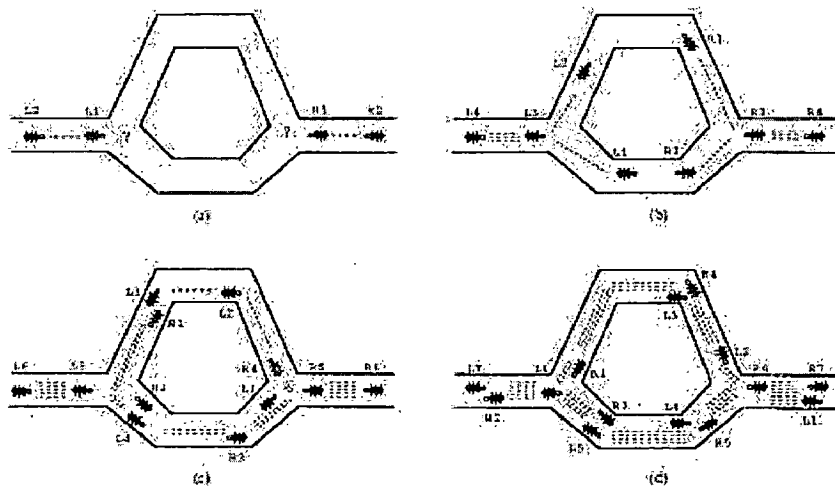


Figure 2.1 Illustration of how real ants find shortest path to food source [37]

Ant colony optimization algorithm is shown in figure 2.2.

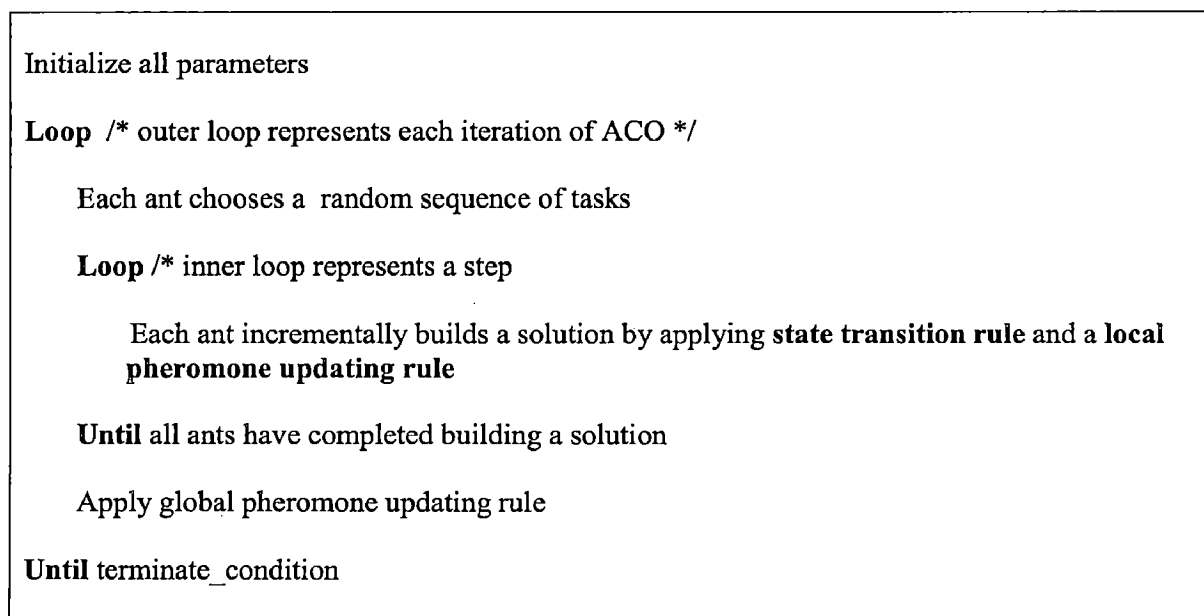


Figure 2.2 The ACO algorithm

In ACO each ant incrementally builds a solution in parallel with other ants. ACO consists of three main features – i) pseudo-random-proportional state transition rule, ii) global pheromone updating of only the solution of best ant and, iii) local pheromone updating rule as each ant builds a solution. The stochastic greedy state transition rule balances exploration of new solutions and exploitation of priori and accumulated knowledge about the problem. Global pheromone updating rule strengthens the task to resource mappings of best solution for selection by later iterations as mappings with higher pheromone values are more desirable. Local updating rule reduces the pheromone value for mapping found during a step so that this mapping is less favorable to be chosen by other ants during this iteration. This helps in moving out of local optimum. Other important information associated with ACO is heuristic information for each mapping. Heuristic information for each task to resource mapping can be execution time of that task on the resource. Both heuristic information and pheromone values are used for guiding selection.

In [38] presented ant algorithm based task scheduling in Grid environment. Scalability of ant algorithm is tested by adding more nodes to an existing network and testing the performance of new extended network. The routing information of solution for a network is used for finding solution for the extended network. Ant algorithm that uses previous information performs better than the one that does not use it. Pheromone value is associated with path between schedule center and resource. Initial pheromone value for each path depends on MIPS of resource, number of processing elements in that resource and transfer time (bandwidth) between that resource and schedule center. Pheromone value for a resource is updated when task is submitted, when task fails, and when task is completed successfully. In [39] is presented an improved ant algorithm for job scheduling. A load balancing factor is introduced to update pheromone. This helps in load balancing. In [42] ACO based dynamic job scheduling is used with the objective of minimizing total tardiness time of jobs. Heuristic information is based on completion times of jobs. Completion time of a job on a resource is sum of arrival time, release time and processing time. In [43] ACO is used for Grid task scheduling with multiple QoS dimensions with the objective of maximizing total utility. Five QoS dimensions are considered – time, reliability, version, security and priority. In [44] a balanced ACO for Grid job scheduling is proposed. Each ant represents a resource and pheromone value is associated between a job and a resource. Pheromone indicator of each resource for each job is based on CPU execution time and transfer time of job. For each

job pheromone indicator is used for selecting the resource for its execution. Local pheromone update is done after each job assignment. Global pheromone update is done after each job completion. Most influential work on ACO based job scheduling in Grid is [45] and it is the one our work on ACO is inspired from. This work will be discussed in chapter 3.

### **2.2.1.6.3 Application model specific fault tolerance**

#### **2.2.1.6.3.1 Malleability support for divide and conquer applications [46]**

Due to dynamic resource availability of grid the grid applications need to be fault tolerant and malleable (ability to cope with the increasing and decreasing number of processors). Divide and conquer applications work by dividing each problem recursively into sub problems until the sub problems become trivial. Then the solutions of sub problems are recursively combined until final solution is achieved. Divide and conquer applications can be run efficiently by running different sub problems on different machines. Each processor acquires work by work stealing: when a processor is idle it steals work from other processor's work queue. In [46] malleability mechanism is based on recomputing jobs lost by failed processors but the amount of lost work is minimized by restructuring the computation tree to reuse as many already computed partial results as possible. This mechanism salvages orphan jobs (jobs stolen from crashed processors) and partial results computed by crashed processors if they leave gracefully. Each processor maintains a list of jobs stolen from it and the identity of the thief. If a processor crashes, each of the live processors searches for jobs stolen by crashed processor and put the stolen jobs back into the work queue. Each job inserted into work queue is marked as restarted. Children of restarted jobs are marked as restarted when they are spawned. To reuse results of orphan jobs, for each finished orphan job a message containing jobID and processorID is broadcasted by the processor computing the orphan job. Each processor on receiving this information keeps it in a local orphan table. When recomputing jobs marked as restarted the orphan table is looked up. If the jobID matches any of the stored entry the corresponding processor is requested for result and job is not put in the work queue. For reusing partial results computed by leaving processors each leaving processor sends the already computed results of finished jobs to any other processor. These received jobs are treated as orphan jobs: processor receiving the finished job treats it as orphan and broadcasts jobID and its processorID and the mechanism followed is as described above.

## 2.3 Research Gaps

Following research gaps have been identified in present fault tolerant techniques in Grid environment:

- Checkpointing is one of the most commonly used approach for tolerating faults in grid environment. But in order to be effective support for checkpointing should be provided while scheduling jobs (checkpointing will not be beneficial on a resource with high probability of failure as even in the presence of checkpointing technique the job will miss its deadline in real time scenarios). Genetic Algorithms are an important class of algorithms for grid resource scheduling. Present researches provide support for fault tolerance in these algorithms using heuristics for checkpointing and replication. These heuristics have been used for building the fitness functions. However another issue with checkpointing is the size of the checkpointing interval. If the size of checkpointing interval is small the overhead of checkpointing (capturing the snapshot of the process i.e. its state, migrating and saving state on the checkpoint server) can be very high to make checkpointing ineffective. On the other hand if size of checkpointing interval is high large amount of work will be lost in case of failure. Approaches exist for online adaptive checkpointing which modify the checkpoint interval based on the frequency of resource failures and the remaining execution time of jobs. But no scheduling support for these adaptive checkpointing approaches exists in the present researches.
- Grid by its definition is a collection of autonomous resources. A Grid respects the autonomous nature of the administrative domains of which it comprises. This autonomous nature leads to inception of new kinds of failures called volunteer autonomy failures [30]. No heuristics exist for genetic algorithms which take into consideration this nature of Grid environment.
- Mean Time to Repair (MTTR) of a resource is an important factor while scheduling if checkpointing is used without support for migration. This may be the case when all resources are heavily loaded. In this case jobs that were running on the failed resource are restarted from the latest checkpoint on the same resource when that resource recovers from failure. So consideration of MTTR of a resource is important when building fitness

functions for Genetic algorithms supporting checkpointing techniques. No present research focuses on this issue.

- Ant Colony Optimization is another subclass of meta-heuristic algorithms which have been found effective for Grid resource scheduling. None of the available research works focuses on incorporating support for fault tolerance techniques such as checkpointing in these scheduling algorithms.
- A majority of study pertaining to fault tolerance are based on the assumption of failures of resources being independent and MTTF and MTTR following exponential distribution. In [10] it has been studied that time between failures is best modeled by Weibull distribution with decreasing hazard rate (Weibull shape parameter 0.7-0.8) and mean repair time is well modeled by lognormal distribution. Failures also tend to cluster in time and are caused by relative small set of computational nodes i.e. failures are temporally and spatially correlated.
- A majority of works in adaptive fault tolerance approaches exists in high performance computing environment. The applicability of these approaches in Grid environment is an area where avenues for study exist.

## Chapter 3

### Proposed Work

---

#### 3.1 Batch Mode Scheduling and Fault Model

In this work a batch-mode scheduler is assumed. This scheduler is a centralized scheduler accepting job requests from grid users. The scheduling time is divided into scheduling intervals and all job requests which have arrived in a scheduling interval form one batch of jobs to be scheduled. Figure 3.1 shows the batch scheduling model. In this figure PE is a resource and each job is assumed to require only one resource for its execution. A job/task executing in a resource on failure of that node/resource is re-submitted to the same resource.

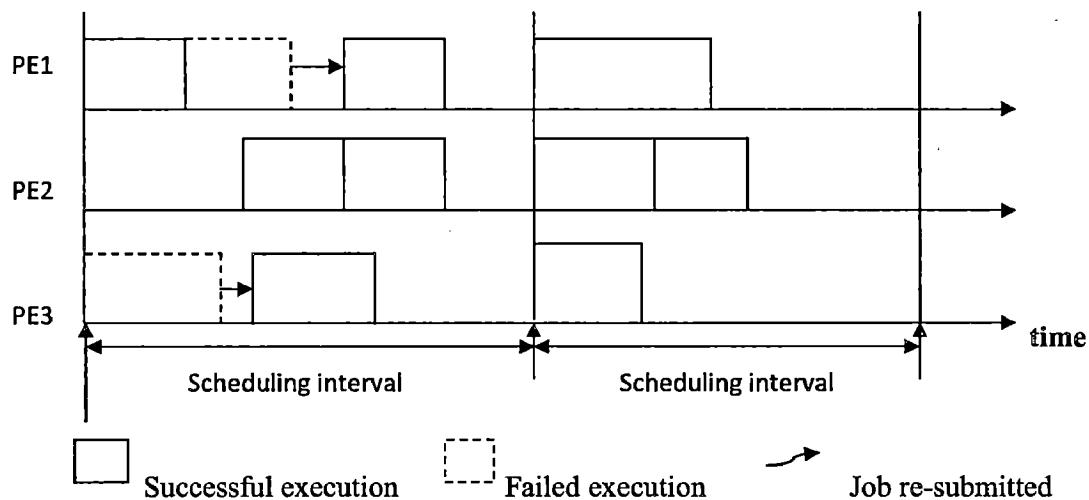


Figure 3.1 Batch mode scheduling

In this work only transient faults are considered. Job failure is any condition in hardware (node failure, network failure, memory failure) or software (exceptions, buffer overflow) [6] which causes the job to stop execution (fail-stop model). Failed nodes are assumed to eventually recover from failures and the failures of nodes are independent of each other ( in actual large scale systems failures are found to be temporally and spatially correlated [10] ). In addition, we assume that the batch scheduler is running on a highly stable (or highly replicated) machine and



does not fail. Failure of a resource is assumed to be immediately detected and recovery time in most cases is assumed to be negligible . Weibull distribution is used for modeling resource availability [10]. Other failure assumptions are mentioned in the sections where they apply.

### 3.2 Fault Tolerance Support in Metaheuristics

#### 3.2.1 Genetic Algorithm Based Fault Tolerance Techniques in Grid Environment

Genetic Algorithm used in this work is the one used in Global Optimization toolbox of MATLAB R2010a [53]. The algorithm is follows:

##### **Genetic Algorithm**

Generate initial population;

Evaluate initial population using **fitness function**;

Loop (**termination criteria**)

a) **Selection**

Select chromosomes from current population on which crossover and mutation operators will be applied to produce new population.

b) Apply genetic operators on current population

**Crossover and mutation** operators are applied on chromosomes selected using selection function.

c) Create new population

The new population has the same size as the current population consisting of:

i) **Elite Count**

ii)  $\text{Crossover\_kids} = \text{Crossover fraction} * (\text{size of current population} - \text{Elite Count})$

iii)  $\text{Mutation\_kids} = \text{size of current population} - \text{Elite Count} - \text{Crossover\_kids}$

d) Evaluate new population

End Loop

Figure 3.2 Genetic algorithm

### A) Chromosome Representation

A chromosome is represented as a one dimensional array indexed by job id. Each entry in the array is the id of the node to which the corresponding job is allocated. For example in Figure 3.3 job J1 is allocated to node N1 job J2 to node N8.

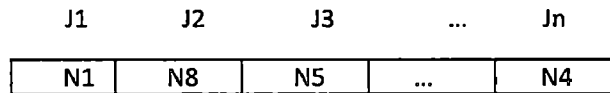


Figure 3.3 Representation of Chromosome

### B) Initial Population

Initial population is represented as a matrix with number of rows equal to the population size and number of columns equal to the number of jobs. Each row represents a chromosome.

### C) Selection Function

Selection function selects the chromosomes from the current population which are used for reproduction of new chromosomes using operations of crossover and mutation. Roulette wheel based selection and stochastic universal sampling are commonly employed techniques for selection.

#### 1) Roulette Wheel Selection

In this technique chromosomes are one-to-one mapped to contiguous intervals which has a range  $[0, \text{Total}]$ , where Total is the sum of the fitness values of the chromosomes in the current population. Each chromosome occupies a contiguous interval equal to its fitness value. Then a random number in the range  $[0, \text{Total}]$  is generated and the chromosome in whose interval that number lies is selected for reproduction. Clearly a chromosome with higher fitness value has greater probability of being selected.

#### 2) Stochastic Universal Sampling

This technique selects a random number between 0 and  $\text{Total}/n$  ( $n$  is the size of current population) as initial pointer. Then  $n$  equally spaced pointers starting at the initial pointer and with the interval size equal to  $\text{Total}/n$  are used for selecting the chromosomes. The chromosome in whose interval the pointer lies is selected for that pointer.

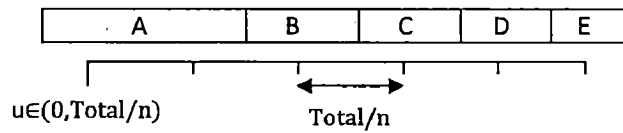
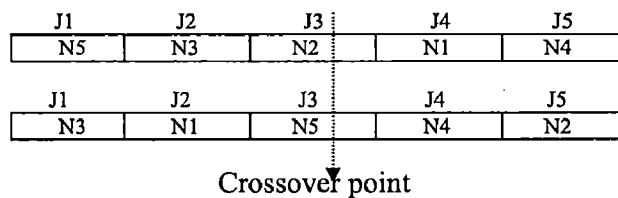


Figure 3.4 Stochastic Universal Sampling

#### D) Crossover Operator

Crossover operator is a global search technique that produces new chromosomes by combining features of the parent chromosomes. The simplest form of crossover operator is single-point crossover. It randomly selects a number in the range (1, num\_tasks) where num\_tasks is the number of tasks. This point is known as crossover point. The portions of the two parent chromosomes after the crossover point are swapped to create two new chromosomes. Figure 3.5 shows application of crossover operator.

##### Parent chromosomes



##### Offspring chromosomes

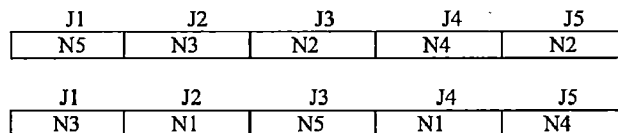


Figure 3.5 Single-point crossover operation

#### E) Mutation Operator

Mutation operator randomly selects a job and replaces the value in its entry in the parent chromosome with some randomly chosen valid resource id. The role of mutation is of guaranteeing that no possible solution in the solution space has zero probability of being searched. This helps in moving out of local optimum.

#### F) Fitness function

Fitness function is used to evaluate the quality of chromosomes and it is the measure that we want to optimize. Fitness function used in this section is a composition of two objective

functions – makespan and flowtime. This function is similar to the one used in [35]. Our objective is to minimize the values of makespan and flowtime of the schedule. Fitness function is given by the following equation:

Fitness = Makespan * Flowtime	(1)
Makespan = $\max \{C_r\}$	(2)
Flowtime = $\sum_{j=1}^m C_j$	(3)

Above notations for makespan and flowtime are similar to those used in [35].  $C_r$  is the completion time of jobs allocated to resource r. So, makespan is maximum of the completion times of Grid resources and flowtime is sum of the completion times of resources (where in equation (3) m is the total number of resources in Grid). In this paper completion time of resource r is taken as sum of execution times of jobs allocated to r i.e.

$C_r = \sum_j^n EX_r^j$	(4)
-------------------------	-----

$EX_r^j$  is execution time of job j on resource r to which it is allocated. In this work execution time of a job is taken as the CPU time for running that job on the resource to which it is allocated. The waiting time and transmission time of a job are ignored. Below are presented some heuristics for execution time of jobs that takes into account the failure conditions of resources.

### 1) MTBF and LF based Adaptive Checkpointing based fitness functions

Following are fitness functions that take into account the failure characteristics of resources such as Mean Failure Time (Mean time between failures), Last Failure Time to modify the checkpointing interval to reduce the cost of checkpointing. These adaptive checkpointing approaches derives the basic idea from [22] where online adaptive checkpointing is presented.

#### a) Mean Failure Time Based Checkpointing

This approach modifies the checkpointing interval based on Mean Failure Time of resources. Mean Failure Time of each resource is maintained in GIS (Grid Information Service) based on historical failure patterns of that resource. The basic idea is that the resource with larger Mean Failure Time (which is more reliable) will incur less checkpointing overhead and thus the execution time on it will be low compared to other resources with comparable resource speed but lower MTBF. If value of makespan is used as fitness function this leads to selection of former resource for job execution against latter resources, which also results in reliable execution (less work lost due to failures).

$$EX_n(i) = \frac{Task_{MI}}{resource_{MIPS}} \quad (5)$$

$$E_n(i) = EX_n(i) + \left\lfloor \frac{(EX_n(i) - MF_n)}{Checkpoint_{interval}} \right\rfloor * Checkpt_{cost} + \left\{ \left\lfloor \frac{MF_n}{2 * Checkpt_{interval}} \right\rfloor * Checkpt_{cost} \right\} \quad (6)$$

$EX_n(i)$  is CPU time required for execution of job  $i$  on node  $n$ .  $Task_{MI}$  is task size in Million Instructions,  $resource_{MIPS}$  is resource speed in Millions of Instructions per Second,  $E_n(i)$  is execution time of job  $i$  on node  $n$ ,  $MF_n$  is mean failure time of node  $n$ ,  $Checkpt_{cost}$  is the overhead of individual checkpoints and  $Checkpoint_{interval}$  is the size of checkpointing interval. Equations 5 and 6 are based on the assumption that on failure a resource is down only for the period required for its restart and this period is assumed to be negligible. Size of checkpointing interval is doubled if the remaining execution time of job on that resource is less than mean failure time of resource. Also the entire computing capability of a resource is assumed to be available for Grid job execution and there is no local load on a resource.

#### b) Last Failure Time Based Checkpointing

This approach is represented by following equation

$$\text{If } (C1 - LF_n + EX_n(i) < MF_n)$$

$$Checkpoint_{interval} = k * Checkpt_{interval} \quad (7)$$

Where  $C1$  is current system time,  $LF_n$  is last failure time of node  $n$  and  $k$  is an integer  $\geq 2$ . Equation for execution time can be derived as in previous approach. If multiple jobs in a batch are assigned to the same resource then for each successively scheduled job  $i$ ,  $LF_n$  is set using the formula:

$$LF_n = C_n(i - 2) + EX_n(i - 1)/2 \quad (8)$$

In equation 8 it is assumed that last job would have failed in the middle of its execution.

Figure 3.6 shows the operation of Last Failure Time Based Checkpointing approach. Since  $C1 - LF_n + EX_n(i) < MF_n$  checkpoint interval is increased by multiplying it by an integer  $k \geq 2$ . The logic behind incrementing checkpointing interval is that the node is less likely to fail during

the execution of this job on it. So overhead of checkpointing can be reduced by taking a higher checkpoint interval.

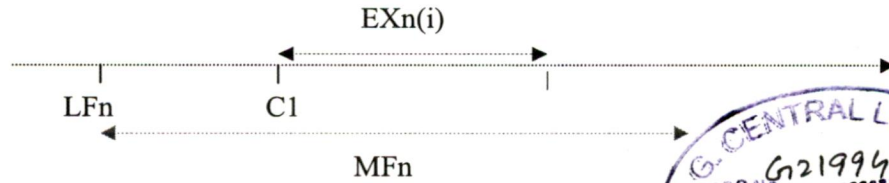
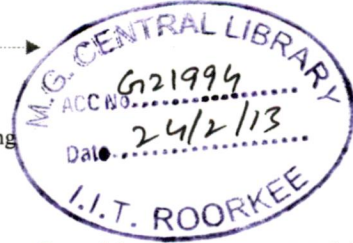


Figure 3.6 Last Failure Time based checkpointing



**c) Checkpointing with Downtimes**

This approach takes into account the downtime (Mean Time to Repair) of resources while scheduling jobs on them. Resources with high downtimes are less favorable during resource allocation.

Following is presented the derivation of the heuristic for representing the execution time of a job on a resource in presence of resource failures.

**i) CPU time** – The time spent in actual execution on the CPU on a resource is given by the formula –

$$CPU_n(i) = \frac{Task_{MI}}{resource_{MIPS}}$$

Task<sub>MI</sub> is task size in Million Instructions, resource<sub>MIPS</sub> is resource speed in Millions of Instructions per Second.

**ii) Total checkpoint overhead** – During its execution a job periodically performs checkpoint after *Checkpt<sub>interval</sub>* seconds. Each checkpointing operation takes some time and this time is equal to *Checkpt<sub>cost</sub>* seconds. So total time spend/wasted in checkpointing operation is the product of total number of checkpoints performed during execution ( $\lfloor CPU_n(i) / Checkpt_{interval} \rfloor$ ) and time taken in performing each operation.

**iii) Failure free execution time (EX<sub>n</sub>(i))** – Failure free execution time is the sum of CPU time and total checkpoint overhead i.e.

$$EX_n(i) = CPU_n(i) + \left\lfloor \frac{CPU_n(i)}{Checkpt_{interval}} \right\rfloor * Checkpt_{cost} \tag{9}$$

iv) **Total time with faults (FTime)** – Expected number of failures during the execution of job is obtained by dividing total time for which a job executes on a resource by the Mean Time Between Failures (MTBF) of the resource.

Now for each failure event the resource goes down for MTTR seconds. So for each failure event an extra MTTR seconds is added to the failure free execution time ( $EX_n(i)$ ). So total time required for execution in addition to the failure free execution time ( $EX_n(i)$ ) is the product of expected number of failures during execution of job i.e.  $\left[\frac{EX_n(i)}{MTBF}\right]$  and MTTR of the resource and hence the total time with faults is

$$FTime = EX_n(i) + \left[\frac{EX_n(i)}{MTBF}\right] * MTTR \quad (10)$$

Above formula assumes that no work is lost during failure as the fault arrives just after the successful completion of checkpointing operation. If the work lost due to failures is taken into account then the equation for FTime becomes

$$FTime = EX_n(i) + \left[\frac{EX_n(i)}{MTBF}\right] * (MTTR + Checkpt_{interval} + Checkpt_{cost}) \quad (11)$$

Above equation is pessimistic in that the failure is assumed to occur just before the completion of a checkpoint.

v) **Total execution time (En(i))** – Total time spend in execution for a job i on resource n is the total time with faults (FTime) i.e.

$$En(i) = FTime_n(i) \quad (12)$$

#### d) Resource Provider Autonomy Based Scheduling

In [30] is presented volunteer autonomy failures in desktop Grids. These failures results from resource providers freely entering and leaving the grid at any time and also due to preference of private execution (local load of a resource) over public execution (Grid jobs). Our approach maintains the time of resource (a volunteer) registration in Grid Information Service. Also the mean time for which a resource remains in the grid (stay time) is maintained. If sum of current system time and execution time on a resource is greater than sum of present registration start time (LRn) and mean stay time (MSn) then a penalty is added to execution time on that resource.

This is due to the possibility of resource provider disconnecting from the Grid before job completion.

## 2) Fault index and Fault ratio based Adaptive Checkpointing based fitness function

The work discussed in this subsection maintains history of fault indexes of resources in a Fault Occurrence History Table (FOHT) as is done in [33]. FOHT maintains two entries for each resource. First entry contains the number of fault occurrences and second entry contains number of jobs submitted to that resource. The FOHT is shown below

	Number of faults	Number of jobs submitted
R1	3	6
R2	5	8
R3	3	11
R4	7	3
R5	4	2
R6	6	12
R7	5	15
R8	3	4

Figure 3.7 Fault occurrence history table

Initially all entries are set to 0. Entries for each resource is updated according to following rules

- If a job is submitted to a resource second column is incremented by 1.
- If a resource fails to complete job within deadline (due to resource failure) first column for that resource is incremented by 1.

### A) Fault ratio based adaptive Checkpointing

The heuristic for representing execution times of jobs is represented by following equations

$$Checkpt_{interval}[i] = C * \left( \min \left( \alpha, \left( \frac{FOHT[i][1]}{FOHT[i][0]} - \beta \right) \right) + 1 \right)$$

$$EX_n(i) = \frac{Task_{MI}}{resource_{MIPS}}$$

(13)

$$E_n(i) = EX_n(i) + \left\lfloor \frac{(EX_n(i))}{Checkpt_{interval}[i]} \right\rfloor * Checkpt_{cost} + (FOHT[i][0] / FOHT[i][1]) * (Checkpt_{interval}[i] + Checkpt_{cost})$$



In equation 13  $C$  represents the initial checkpointing interval for the experiment and  $Checkpt_{interval}[i]$  represents the adaptive checkpointing interval for jobs submitted to resource  $i$  set depending on FOHT values.  $\alpha$  limits the increase in checkpointing interval and  $\beta$  limits the decrease in checkpointing interval.

### B) Fault index based adaptive Checkpointing

$$\begin{aligned}
 Checkpt_{interval}[i] &= C * (\min(\max((FOHT[i][1] - FOHT[i][0]), \beta), \alpha)) \\
 EX_n(i) &= \frac{Task_{MI}}{resource_{MIPS}} \\
 E_n(i) &= EX_n(i) + \left[ \frac{(EX_n(i))}{Checkpt_{interval}[i]} \right] * Checkpt_{cost} + \\
 & (FOHT[i][0] / FOHT[i][1]) * (Checkpt_{interval}[i] + Checkpt_{cost})
 \end{aligned} \tag{14}$$

In equation 14  $\alpha$  limits the increase in checkpointing interval.

### 3.2.2 Ant Colony Optimization Based Fault Tolerant Scheduling in Grid

Various phases of ACO are explained below. This work on ACO based job scheduling is similar to that in [45].

#### 1) Initialization:

Parameters of ACO are set in this phase.  $\rho$  which controls local pheromone update and global pheromone update is set to .1.  $\beta$  which determines weight age of heuristic value is set to 1.2.  $q_0$  which controls application of exploration vs exploitation is set to .9.

#### 2) Pseudorandom state transition rule:

An ant chooses a resource  $r$  for a job  $j$  using the following rule

$$r = \left\{ \begin{array}{l} \arg \max_{u \in R} \{ [\text{pheromone}(u, j)] \cdot [\eta(u, j)]^\beta \}, \\ \text{if } q < q_0 \text{ (exploitation)} \\ S \text{ (biased exploration), otherwise} \end{array} \right\} \tag{15}$$

R is the set of available resources. Pheromone(u,j) is pheromone value of resource u for job j.  $\eta$  is the heuristic value of resource u for job j.  $\beta$  is a parameter greater than 1 which determines relative importance of heuristic vs pheromone.  $q_0$  is a parameter and q is a random variable uniformly distributed between 0 and 1. S is the resource selected using stochastic universal sampling.

### 3) Local pheromone update rule:

Local pheromone update is performed by each ant after each step. Phermone value of resource r selected for a job j is updated using the following rule:

$$\text{Pheromone}(r,j) = (1-\rho).\text{pheromone}(r,j) + \rho.\text{initial\_pheromon} \quad (16)$$

Local pheromone updating rule is pheromone value of resource r for job j. So this selection will be less for other ants. This helps is exploring the entire solution space and moving out of local optimum.

### 4) Global pheromone update rule:

Global pheromone update rule is performed after each iteration for the best ant found in that

iteration. Pheromone value of job to task mappings in the best ant is updated using the formula:

$$\text{Pheromone}(r,j) = (1-\rho).\text{pheromone}(r,j) + \rho.\text{score} \quad (17)$$

score for the best ant is calculated as

$$\text{score} = 1 + \text{minimum\_makespan}/\text{makespan} \quad (18)$$

minimum\_makespan is the minimum makespan found in all preceding iterations and current iteration, whereas makespan is makespan of the best ant of this iteration for which score is being calculated.

## 3.3 Adaptive Checkpointing based Fault Tolerance Techniques

### 3.3.1 Fault Index Based Adaptive Skip

Periodic skip [19] based checkpointing technique periodically skips checkpoints for a job. For example if a “skip parameter” d is equal to 1 then alternate checkpoints are skipped. Higher value of d higher will be the number of checkpoints skipped. Fault index based periodic skip

uses fault indexes [23] of resources to determine the intensity of skipping checkpoints (skipping parameter). For each resource fault index is maintained. Fault index is a measure of vulnerability of the resource towards failures. It is incremented on each job failure and decremented on each successful job completion. Pseudo code for fault index based periodic skip is given below

```

FI(i): Fault Index of resource i
FI1, FI2, FI3.....FIN fault index values such that FI1 < FI2 < FI3 < ..... <FIN
D1, D2, D3,.....,DN skip parameter to determine intensity of skip such that D1 < D2 <D3
...DN
If(FI(i) > Fin) then
    Perform all checkpoints
If( FIN >FI(i)>FIN-1
    Use D1 has skip parameter
If(FI1 >FI(i))
    Use DN has skip parameter
Exit

```

Resources with higher fault indexes are more prone to failures. So fewer checkpoints are skipped (or no checkpoints are skipped). On the other hand, since resources with less fault index value are less vulnerable to failures more checkpoints are skipped for these resources. This helps in adapting the checkpoint interval depending on failure conditions of resources. As a consequence lesser checkpoint overhead is incurred and lesser work is lost due to failures. Finally this would result in less execution time for resources.

In addition to adaptive fault index based periodic skip fault index based exponential backoff skip can also be done. This technique a cooperative checkpointing technique [19] increases the amount of saved work for each completed checkpoint. For example 1<sup>st</sup>, 2<sup>nd</sup>, 4<sup>th</sup>, 8<sup>th</sup>, 16<sup>th</sup>, etc checkpoints are performed in each availability interval for “exponential parameter” equal to 2. Modifying this exponential parameter depending on fault indexes of resources can be highly useful in reducing the amount of work lost due to failures.

Other than using fault index for determining “skip parameter” or “exponential parameter” fault ratio as discussed in 3.2.1 can be used for the same purpose.

### 3.3.2 MTBF and Last Failure Based Adaptive Checkpointing for Temporally Correlated Failures

As found in [10] failures in production HPC systems are temporally correlated. This observation can be harnessed in online adaption of checkpoint interval for each resource. Heuristics based on last failure and MTBF for adapting the checkpoint interval have been presented in [22]. These heuristics are dependent on execution times of resources which can be highly unpredictable in a highly dynamic environment such as Grid where load on resources changes dynamically and unpredictably. To eliminate this shortcoming following heuristic is proposed

<p>If <math>((C1-Lf) &gt; \alpha * MTBF)</math>  <math>AI = AI + I</math> for each checkpoint request in interval <math>(\alpha * MTBF, \beta * MTBF)</math> where <math>\beta &gt; \alpha</math> and <math>\alpha &gt; 1</math> <span style="float: right;">(19)</span></p> <p>If <math>((C1-Lf) &lt; \gamma * MTBF)</math>  <math>AI = AI - I</math> for each checkpoint request in interval <math>(\gamma * MTBF, \eta * MTBF)</math> where <math>\eta &gt; \gamma</math> and <math>\gamma &lt; 1</math></p>
---

where I is an initial periodic checkpoint interval and AI is adapted checkpoint interval. C1 is current time and Lf is last failure time of resource. The logic behind above equations is based on two properties of temporal correlation between failures.  $\alpha$ ,  $\beta$ ,  $\gamma$ , and  $\eta$  are parameters of the technique.

- i) If a node has not failed for a long time then there is less probability that it will fail in the near future.
- ii) If a node has failed recently then there is high probability that it will fail in the near future.

Concentrating on first if condition in equation 19 and point i. If difference between current time C1 and last failure time Lf is greater than some multiple of MTBF ( $\alpha * MTBF$  where  $\alpha > 1$ ) then increase checkpoint interval by I for each request during interval  $(\alpha * MTBF, \beta * MTBF)$ .

Similarly concentrating on second if condition in equation 19 and point ii. If difference between current time  $C1$  and last failure time  $Lf$  is less than some fraction of MTBF ( $\gamma * MTBF$  where  $\gamma < 1$ ) then decrease checkpoint interval by  $I$  for each request during interval ( $\gamma * MTBF, \eta * MTBF$ ).

## Chapter 4

### Simulation Environment and Implementation Details

---

Grid environment is a very complex system with resource and network conditions changing rapidly and unpredictably. Also Grid is a decentralized environment with resources managed in multiple autonomous administrative domains. Performance testing of developed techniques for various aspects such as job scheduling, fault tolerance etc. in a repeatable and controllable manner becomes very difficult in such environment. Simulation is the best tool for performance testing in repeatable and controllable manner and we stick to this choice for evaluating our techniques.

GridSim toolkit [47][48] is one of the most popular event driven simulator which is used for simulating Grid environment. GridSim implements core entities that simulate resource, information service, statistics, and shutdown services. These services are used to simulate a user with application, a broker for scheduling, etc. Interaction between these entities takes place through events. Events are used for service request and service delivery. Events can be internal (generated by the entity which receives it) or external (generated by some other entity).

Figure 4.1 shows the components of the Grid environment simulated using GridSim. These components are described below.

#### a) Grid user

Submits the jobs to the broker for execution using a Grid portal and receives the results back on successful execution. Grid user may also specify certain constraints such as budget, deadline etc. Each job has a job id, user id, length (MI) associated with it.

#### b) Resource broker

Resource broker is responsible for receiving jobs from the users. Its functionality is broken down into following modules

##### i) Metascheduler

Meta scheduler is a batch scheduler as discussed in Section III. It is used for scheduling of jobs.

## ii) Job Dispatcher

It is responsible for submitting jobs to resources either from the beginning or from the most recent checkpoint.

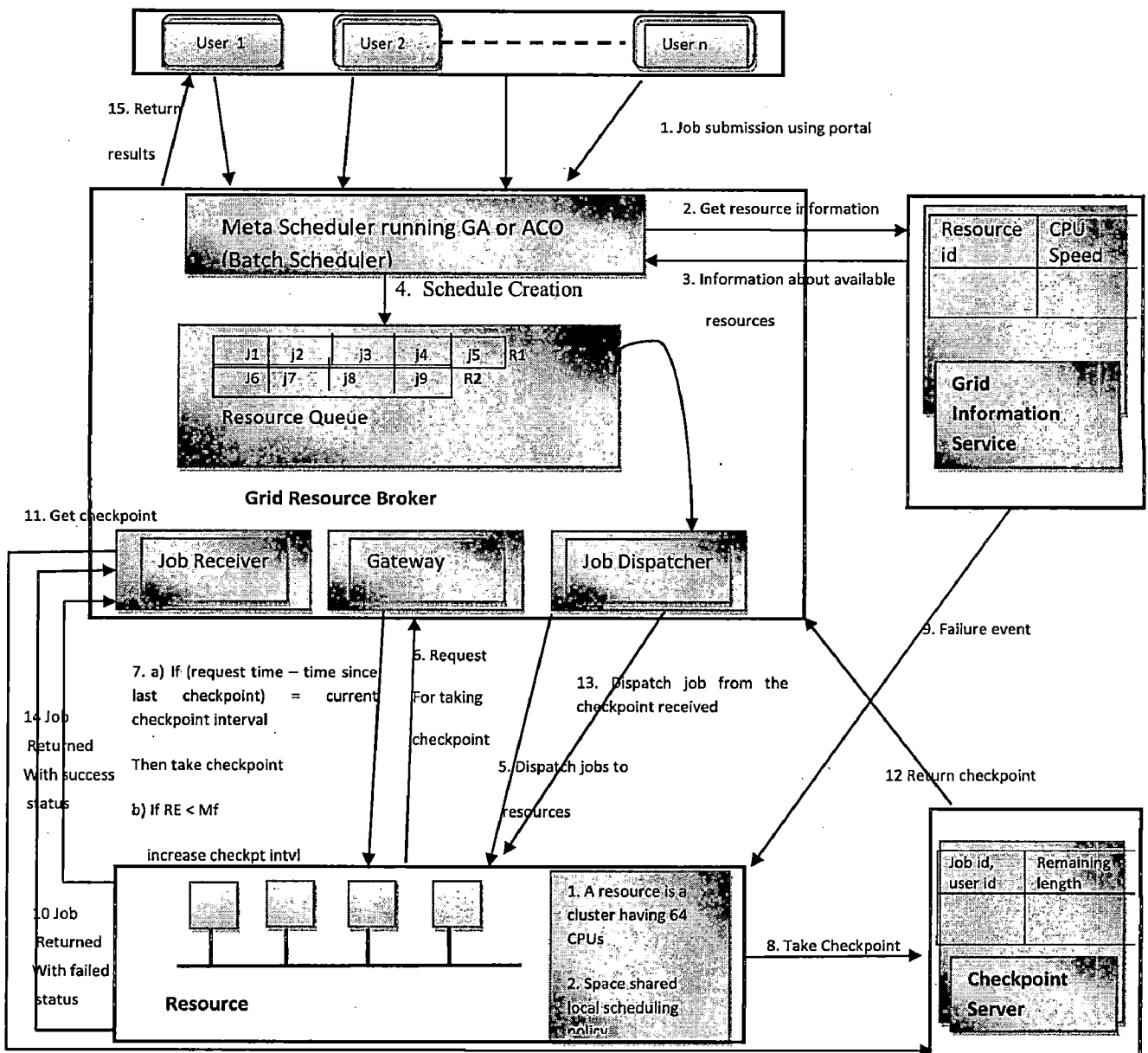


Figure 4.1 Components of Grid environment

### **iii) Job Receiver**

It is responsible for receiving jobs with status either marked as success, in which case result is returned to user, or marked as failed. In case of failed status a `get_checkpoint` request is send to checkpoint server. This request contains the job id and user id.

### **iv) Gateway**

The gateway [12] is responsible for deciding whether to take or skip a checkpoint and also to modify the checkpoint interval.

### **c) Grid Resource**

It is a cluster of computing nodes and is used for executing Grid jobs. Each resource in our simulation implements a space shared policy [47].

### **d) Grid Information Service (GIS)**

This module maintains resource ids and other characteristics of resources such as number of CPUs, computing power, available memory, MTBF, etc. The broker module requests information about resources from GIS.

### **e) Checkpoint Server**

It is responsible for maintain a table containing job id, corresponding user id and remaining length (MI) of jobs. Only the most recent checkpoint is maintained for each job.

Figure 4.2 shows interaction between various entities and various events generated during execution of a gridlet/job. These sequences of steps are described below.

a) At the start of simulation all Grid resource entities register themselves in GIS.

b) Grid user submits job to Grid resource broker. In GridSim a job is represented by gridlet class. This class encapsulates all properties of a job such as its length (MI), size (PEs required), user id (user to which this gridlet belongs), gridlet id, i/p file size, and o/p file size. This class can also be used to encapsulate other properties such as deadline and budget constraints.



c) On receipt of job broker requests to GIS for list of available resources. Checkpoint server and fault index manager also request GIS (not shown in figure) for list of available resources. Broker also sends **GET\_FAULT\_INDEX** event to Fault index manager to get fault indexes of all resources.

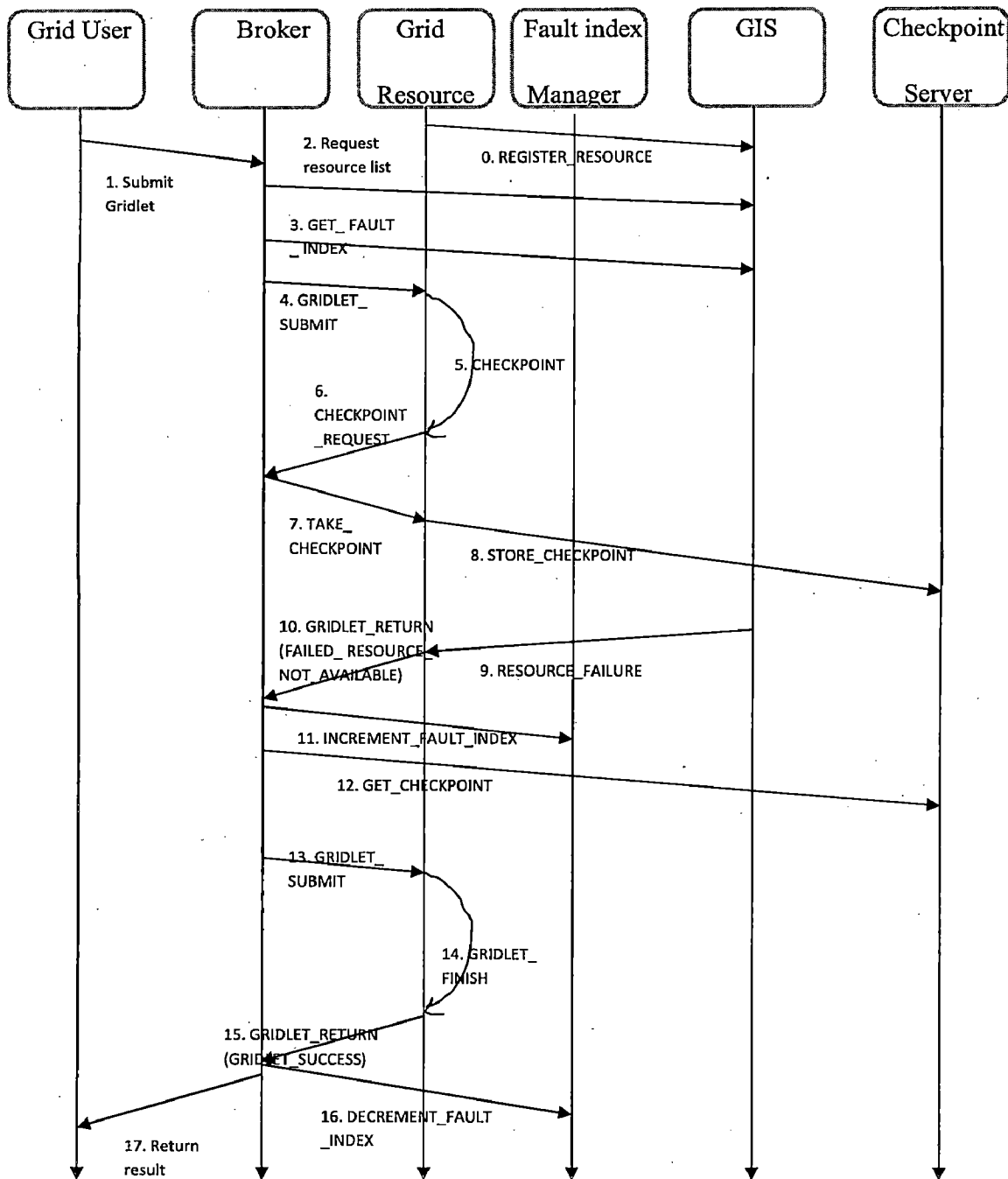


Figure 4.2 Event diagram to represent interaction between different GridSim entities

d) Resource broker submits the job to appropriate resource i.e. resource which meets all requirements such as number of free PEs, size of available memory etc. In case multiple jobs have arrived at broker, it uses appropriate scheduling algorithm (GA, ACO, etc.) to find a schedule for jobs. Submission of a job in GridSim is modeled as a send event with **GRIDLET\_SUBMIT** tag (Note: each event in GridSim has an integer tag associated with it that uniquely identifies that event).

e) On receiving a job Grid resource entity allocates resources (PEs) to it according to its local scheduling policy (time shared or space shared). Also if execution time of job is greater than checkpoint interval an internal event with tag set as **CHECKPOINT** is sent to arrive after checkpoint interval seconds.

f) On receiving an event with **CHECKPOINT** tag the remaining lengths of all jobs in execution list of this resource is updated and an event (with tag **CHECKPOINT\_REQUEST**) is sent to gateway module of broker. The gateway module of broker decides whether to take or skip checkpoint. If the request is granted a **TAKE\_CHECKPOINT** event is send back to the resource. On receiving a **TAKE\_CHECKPOINT** event the resource sends a **STORE\_CHECKPOINT** to checkpoint server. This event contains Gridlet id, user id, and remaining Gridlet id as its data field. Also the resource checks whether the remaining execution time of Gridlet with which the checkpoint event was associated is less than or greater than checkpoint interval seconds. If it is less than checkpoint interval seconds then a **GRIDLET\_FINISH** event is send after remaining time plus checkpoint overhead seconds (to account for overhead of checkpoint) otherwise a **CHECKPOINT** event is sent after the same interval.

g) During the course of simulation GIS may send **RESOURCE\_FAILURE** events to resources. Actually these events are sent to each resource after its MTBF interval. On receiving this event the resource sends all executing and waiting jobs on it to their respective users. A **GRIDLET\_RETURN** event with Gridlet status set as **FAILED\_RESOURCE\_NOT\_AVAILABLE** is sent for each Gridlet.

h) On receiving a **GRIDLET\_RETURN** event with Gridlet status marked as **FAILED**

RESOURCE\_NOT\_AVAILABLE broker sends INCREMENT\_FAULT\_INDEX event to fault index manager to update failure information about the failed resource. A GET\_CHECKPOINT event is sent to the checkpoint server to get any saved checkpoint for the failed Gridlet. After receiving the checkpoint Gridlet is resubmitted to the same resource from its checkpoint rather than from the start.

i) On receiving the Gridlet submission event the resource again allocates resource to Gridlet and checks whether its execution time is less than checkpoint interval seconds. If it is less a GRIDLET\_FINISH internal event is send with a delay equal to the execution time of Gridlet.

j) On receiving a GRIDLET\_FINISH event the Gridlet is returned to the broker with status marked as GRIDLET\_SUCESS. On receiving the successfully executed Gridlet broker returns the results/output to the user.

To simulate our techniques extensive modifications were made in the classes available in GridSim Toolkit and also a few new classes were developed. All these classes with modifications made are described below:

### 1) GridScheduler.java

This class represents centralized broker and a batch scheduler. Following functionalities were added to this class:

- a) GA and Ant colony based centralized batch scheduler for job scheduling.
- b) Interaction with FTManager for incrementing and decrementing failure information of resources.
- c) Interaction with CheckpointServer for getting the latest successively saved checkpoint on gridlet failure.
- d) Gateway module for skipping and granting checkpoint requests.
- e) Gridlet creation, submission, reception.

### 2) FTGridResource.java

This class models a Grid resource (cluster). It has following functionalities:

- a) Allocation of PEs to gridlets. Allocation of multiple PE to a single gridlet is supported.

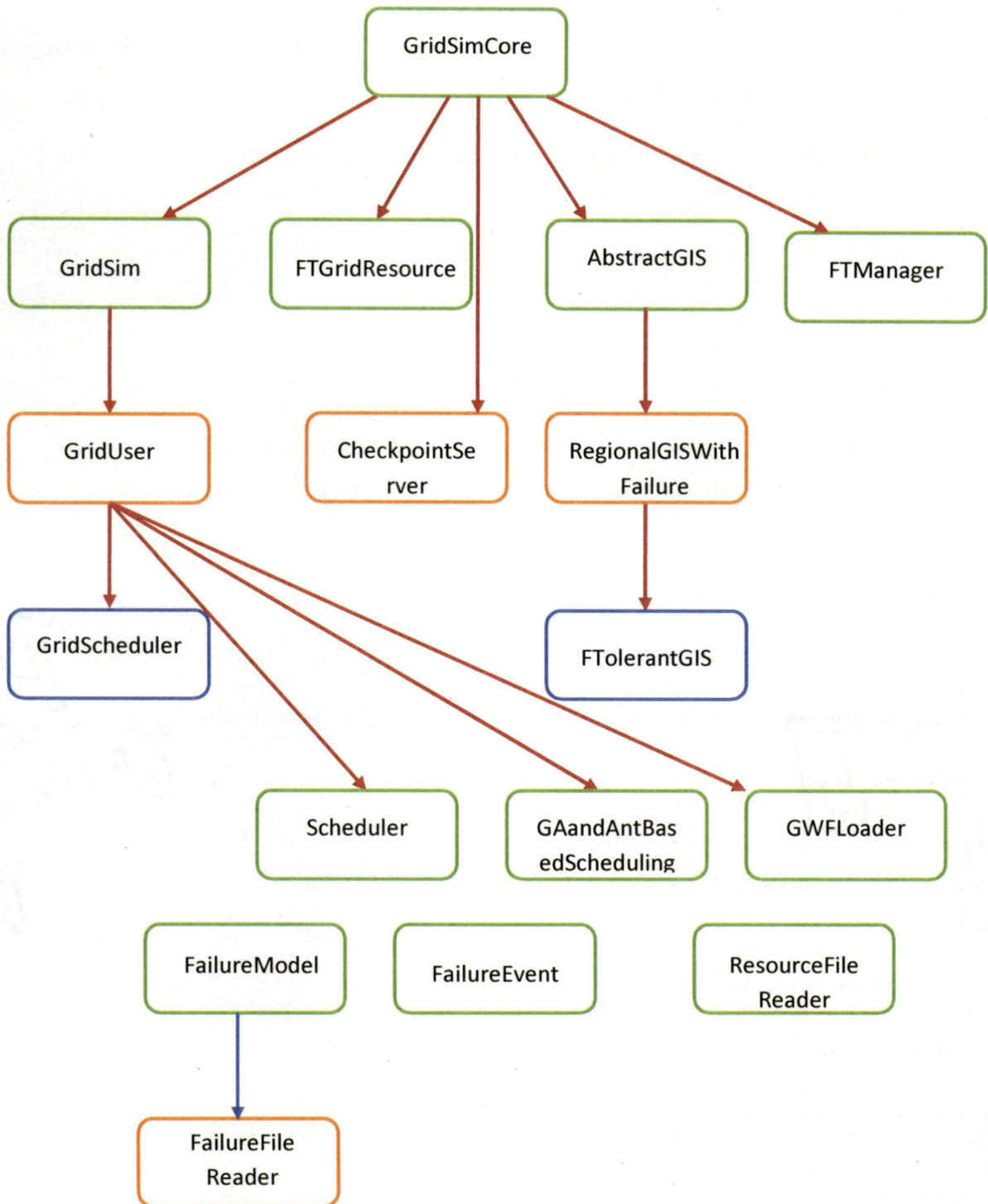


Figure 4.3 Modules hierarchy used for experimentation of proposed techniques

b) Periodic checkpointing of gridlets with checkpointing overhead factored into the execution time of gridlet. Each checkpoint request involves interaction with gateway.

### **3) FTolerantGIS.java**

This class represents Grid Information Service (GIS). It maintains information about available resources in Grid. Other added functionalities include

a) Sending failure events to resources with successive events separated by Weibull distribution generated interval.

b) Maintaining information about last failure and MTBF of resources.

### **4) CheckpointServer.java**

This module represents a centralized checkpoint server. It has following functionalities:

a) Storing checkpoints for gridlets. A table is maintained with each entry having three fields – gridlet id, user (owner) id, and remaining gridlet length. As coordinated checkpointing is assumed only latest checkpoint is stored for each gridlet.

b) Returning latest checkpoint for gridlet on request from scheduler.

### **5) FTManager.java**

This module maintains Fault Occurrence History Table (FOHT) as discussed in chapter 3.

### **6) FailureFileReader.java, ResourceFileReader.java**

These files are used for reading real failure traces, node information available in FTA Archive []. Failed trace reading has been modified to ignore downtimes of resources.

### **7) Scheduler.java**

This class is modification of a class (with the same name) available in Alea Grid simulator [41].

### **8) GAandAntBasedScheduling.java**

This class represents a scheduling policy for job scheduling. It has following functionalities:

a) Implementation of Genetic Algorithm for scheduling of gridlets from real workload traces (with gridlets having different arrival times and different PE requirements) on multiple PE resources.

b) Implementation of Ant Colony Optimization for scheduling of gridlets from real workload traces (with gridlets having different arrival times and different PE requirements) on multiple PE resources.

### **9) GWFLoader.java**

This class is used for reading gridlets from .gwf files.

## Chapter 5

### Performance Evaluation and Experimental Results

---

#### 5.1 Performance Metrics

Following metrics are used for comparing various checkpointing techniques.

**a) Makespan:** It is the maximum completion time for any resource and is basically the time when all jobs finish execution. Completion time for a resource is the point of time when all jobs allocated to that resource completes execution.

**b) Flowtime:** It is the sum of the completion time for all the resources.

**c) Average bounded slowdown:** It is the average slowdown of a job. It is the difference between time taken to execute a job and the CPU time  $\left(\frac{\text{Task}_{MI}}{\text{resource}_{MIPS}}\right)$  averaged over all jobs. Sizes of jobs are taken to be comparable to each other.

**d) Work lost due to failures:** It is the unsaved work which is lost due to failure of jobs.

**e) Utilization:** Utilization is the fraction of time of the resources which is used in executing jobs i.e. in doing useful work. This work does not include the time spent in carrying out work which is lost due to failures.

**f) Number of Checkpoints:** It is the total number of checkpoints performed during the entire simulation run for a batch of job.

**g) Average turnaround time:** It is the average of completion times of jobs. Completion time of job is the finish time of a job minus the submission time.

Makespan and flowtime are parameters which are affected by scheduling decisions. Makespan and flowtime are also affected by arrival time of jobs. In this case these may not be useful parameter for performance comparison. Flowtime may be better performance parameter when job results are returned as soon as it completes execution. Average bounded slowdown is affected by two parameters – number of checkpoints taken (checkpoint overhead) and work lost due to failures. It is an important parameter as it shows the balance between work lost due to failures and time wasted in checkpoints. Utilization is dependent on makespan and is inversely related to it. Turnaround time decides average response times of jobs and is an important parameter for performance comparison. Work lost due to failures depends on two factors – checkpoint interval and scheduling decision. Scheduling decisions that allocate more jobs

to stable resources compared to unstable ones have less work lost due to failures. Work lost due to failures is directly proportional to checkpoint interval. Higher checkpoint interval causes more work loss due to failures. Much lesser checkpoint interval may also lead to more work lost due to failures. This is due to more time required for execution of jobs (due to greater overall checkpoint overhead) which can increase the probability of failure of job. Other than these waiting time, response time and tardiness time of a job are also considered in 5.13 and 5.14. Waiting time is submission time minus arrival time of a job. Response time is the turnaround time as discussed above. Tardiness time is completion time minus due date.

## 5.2 Simulation Parameters

Simulation parameters for GA-based and ACO-based fault tolerance techniques comparison are given in table 1.

Parameter	Value
1. Number of Resources (Clusters)	5
2. Number of Processors per Cluster	64
3. Number of jobs	200
4. Computation time per job	48 hours
5. Checkpoint Overhead	360 seconds
6. Size (Number of processors) of job	64
7. Checkpoint Interval	1000 to 10000 seconds
8. MTBF of Resources	3 hours to 18 hours
9. Failure Distribution	Weibull (shape parameter .7, 1, 1.5)
10. Elite Count	2
11. Crossover fraction	.9
12. Initial Population (GA)	Number of jobs (200)
13. Number of Ants	Number of resources (5)

Table 1 Simulation Parameters for GA-based and ACO-based fault tolerance techniques comparison



### 5.3 GA-based Adaptive Fault Tolerance Using MTBF of Resources

The simulated Grid environment consists of 5 resources (clusters). Each resource has 64 processing elements. Each job requires around 2 days of computation on 64 processing elements. A total of 200 jobs are submitted in a batch. After every checkpointing interval all processes of a job coordinates to take a global checkpoint (coordinated checkpointing [14]). Each checkpoint incurs an overhead of 720 seconds. Processes of a job are assumed to be heavily depended on each other. So even if one of them fails all have to be restarted from latest saved checkpoint. It is assumed that an entire resource is allocated for execution of a job. MTBF of a resource follow Weibull distribution with resources having shape parameters .7, .7, 1, 1, 1.5. Mean Time between failures of resources ranges from 5 hours to 18 hours. Scale parameter of Weibull distribution is calculated using the formulae

$$\frac{1}{\eta} = \lambda \Gamma(1 + \frac{1}{\beta})$$

Where  $\eta$  is scale parameter,  $\lambda$  is failure rate and  $\beta$  is shape parameter. Failure rate ( $\lambda$ ) is inverse of MTBF. Only transient faults are considered with recovery requiring only a restart of the failed machine (this restart time is assumed to be negligible). On failure user jobs are returned with status marked as FAILED\_RES\_UNAVAIL. Resource failures are assumed to be immediately detected by polling mechanism employed by both broker and GIS. Failed job is restarted on the same resource from last saved checkpoint available at the checkpoint server.

Figures 5.1 to 5.7 shows results of experiments performed. The checkpointing interval is varied from 1000

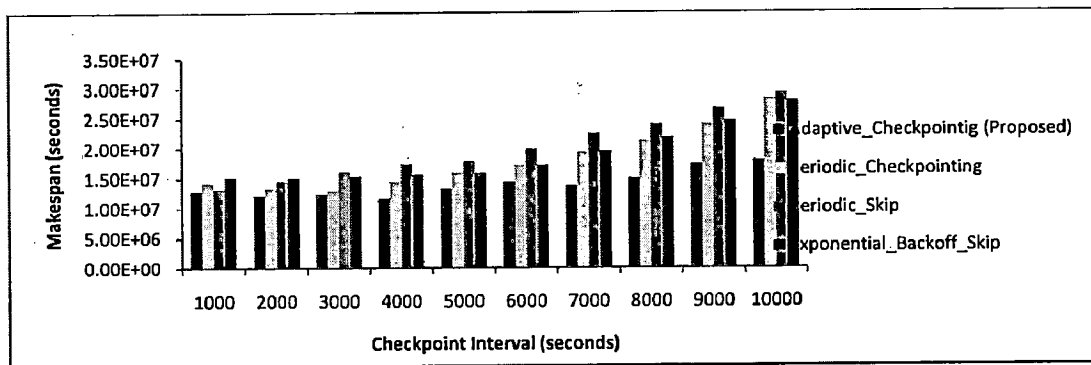


Figure 5.1 Makespan comparison between checkpointing techniques

seconds to 10000 seconds. 5.1 shows comparison against makespan parameter. Adaptive checkpointing performs better than periodic checkpointing for every checkpointing interval. Periodic skip has lesser makespan for initial intervals due to its checkpointing interval (period after which checkpoint is actually taken) being closer to the optimal checkpoint interval for the experiment. However its performance deteriorates at higher checkpointing intervals due to the reason which is clear from figure 5.2 i.e. work lost due to failures becomes very high at higher checkpointing intervals.

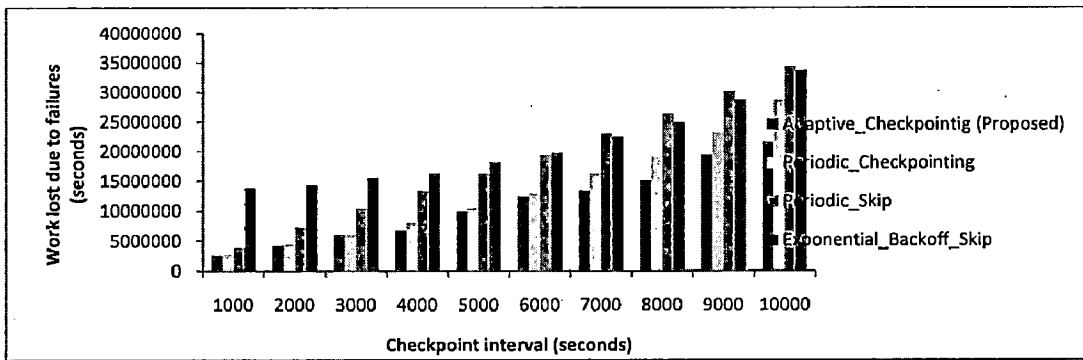


Figure 5.2 Comparison for work lost due to failures

Similar reasons explain the results for exponential backoff skip. As seen from Figure 5.2 work lost due to failures of adaptive checkpointing is higher than that of periodic checkpointing. This is due to adaptive increase in checkpoint interval which also increases amount of work lost due to failures. Periodic skip and exponential backoff skip have very high values for work lost due to higher interval for taking checkpoints. Figure 5.3 show results for flowtime of execution.

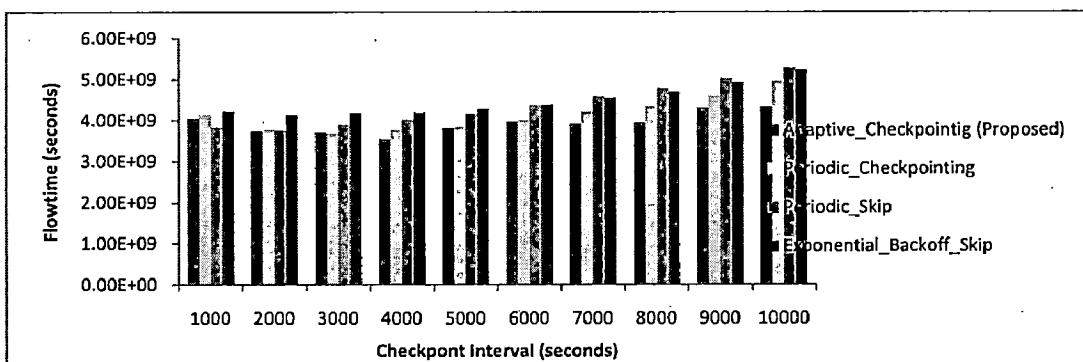


Figure 5.3 Comparison for flowtime parameter

Results for flowtime are similar to those for makespan. Adaptive checkpointing takes lesser number of checkpoints compared to periodic checkpointing due to adaptive increase of checkpointing interval.

Periodic skip and exponential skip take very few checkpoints which is also the reason for high values for lost work.

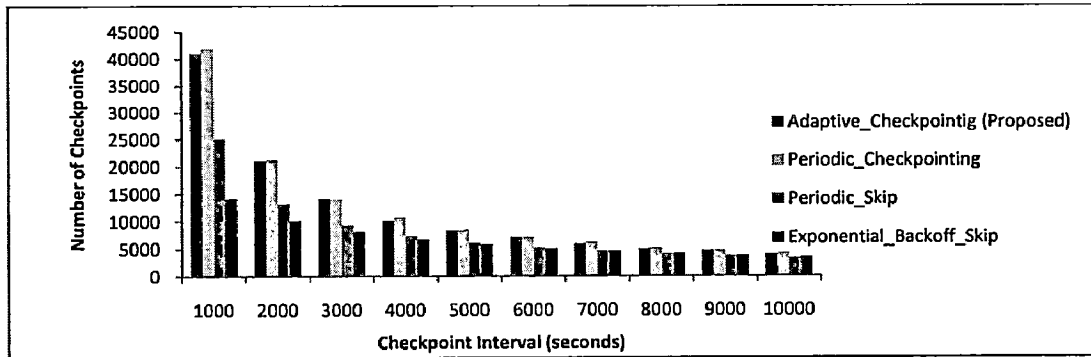


Figure 5.4 Comparison for number of checkpoints taken parameter

Adaptive checkpointing technique shows significant improvements in average bounded slowdown which can be seen in figure 5.5. Results for average bounded slowdown are similar to those for makespan and flowtime.

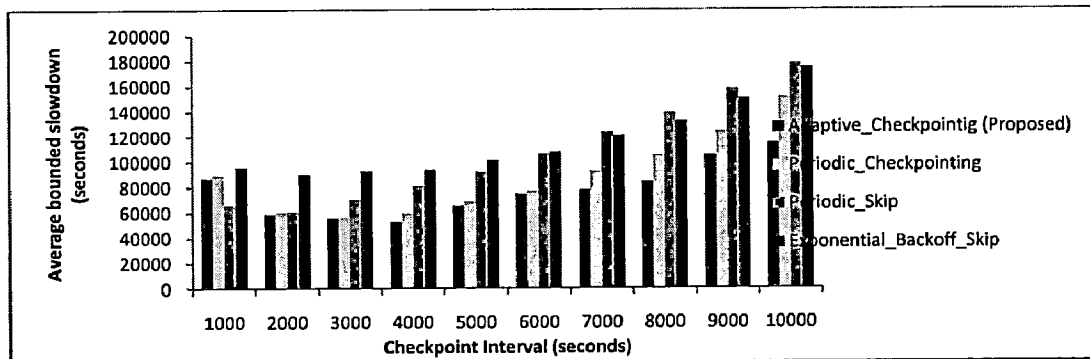


Figure 5.5 Comparison for average bounded slowdown parameter

Figure 5.6 gives results for utilization. Again adaptive checkpointing technique performs better than other approaches and has higher utilization. Results for utilization are just inverse of results for average bounded slowdown (makespan, and flowtime). For initial checkpointing intervals adaptive checkpointing has higher average bounded slowdown (makespan, flowtime) compared to periodic skip and exponential backoff skip. Whereas as checkpointing interval increases adaptive checkpointing performs better. On the other hand utilization is low for adaptive checkpointing at lower checkpointing intervals but utilization becomes higher for adaptive checkpointing with increase in checkpointing intervals.

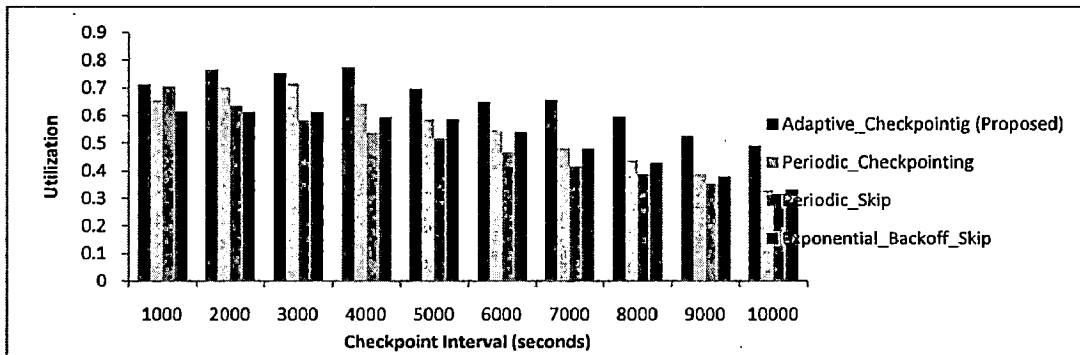


Figure 5.6 Comparison for utilization parameter

Figure 5.7 shows comparison between four techniques for makespan parameter for total number of jobs varying from 200 to 600. Clearly adaptive checkpointing technique has very low values for makespan compared to other techniques.



Figure 5.7 Comparison for makespan parameter for varying number of jobs

Figure 5.8 shows comparison between four techniques for work lost due to failures parameter for total number of jobs varying from 200 to 600. Clearly adaptive checkpointing technique has very low values for total work lost due to failures compared to other techniques. This is due to scheduling support for fault tolerance. Figure 5.9 shows comparison between four techniques for flowtime parameter for total number of jobs varying from 200 to 600. Clearly adaptive checkpointing technique has very low values for flowtime compared to other techniques. Figure 5.10 compares four techniques for total number of checkpoints taken parameter.

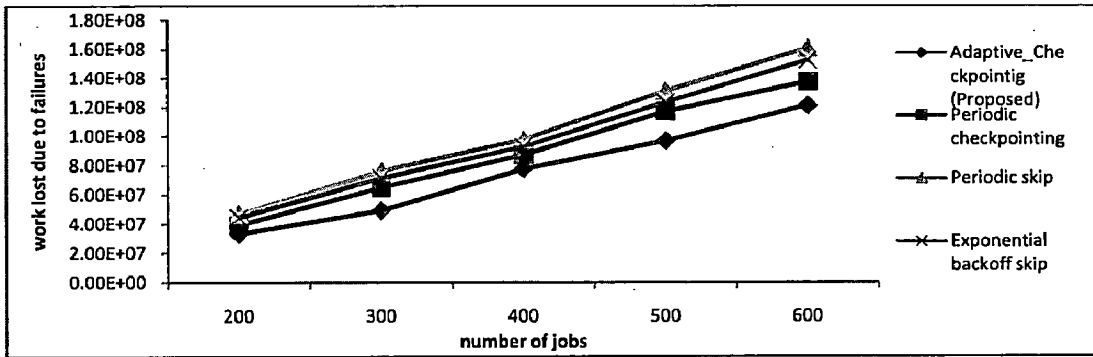


Figure 5.8 Comparison for work lost due to failures for varying number of jobs

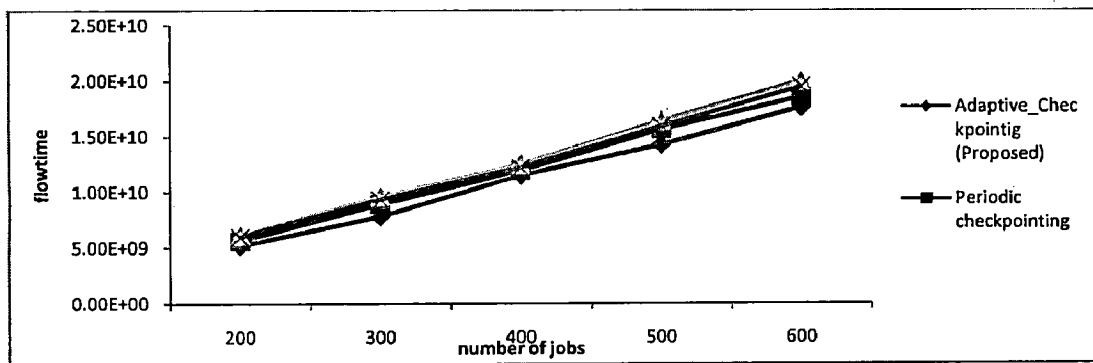


Figure 5.9 Comparison for flowtime for varying number of jobs

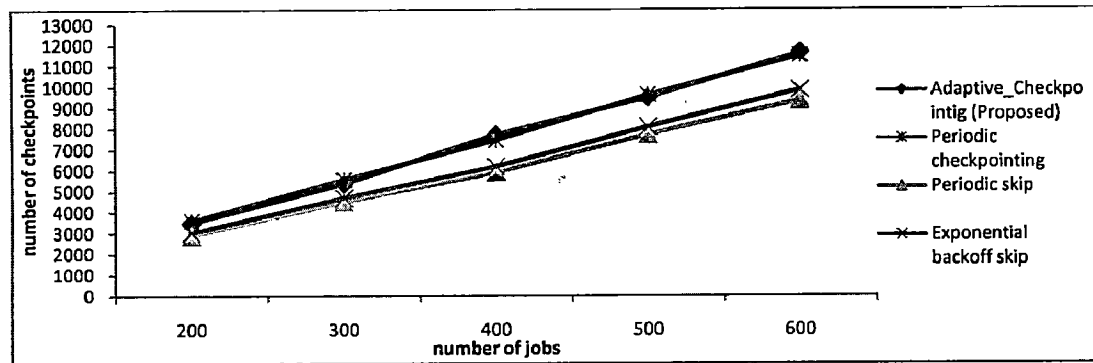


Figure 5.10 Comparison for number of checkpoints taken for varying number of jobs

Figure 5.11 shows comparison between four techniques for average bounded slowdown parameter for total number of jobs varying from 200 to 600. Clearly adaptive checkpointing technique has very low values for total average bounded slowdown compared to other techniques.

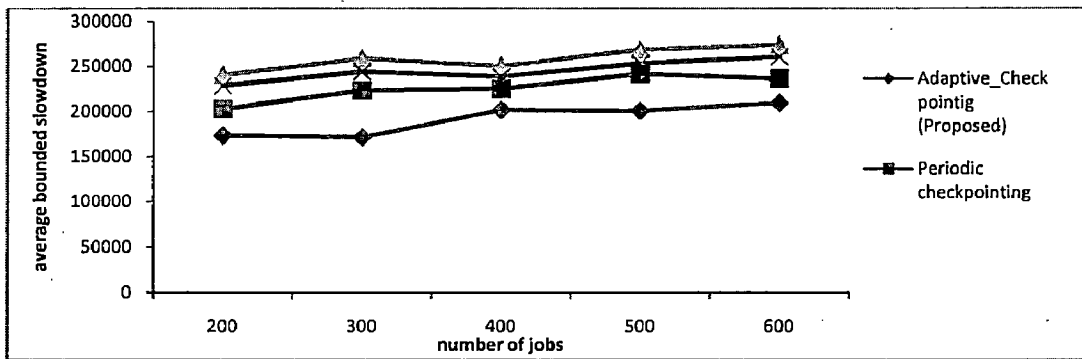


Figure 5.11 Comparison for average bounded slowdown parameter for varying number of jobs

Figure 5.12 shows comparison between four techniques for utilization parameter for total number of jobs varying from 200 to 600. Clearly adaptive checkpointing technique has very high values for utilization compared to other techniques.

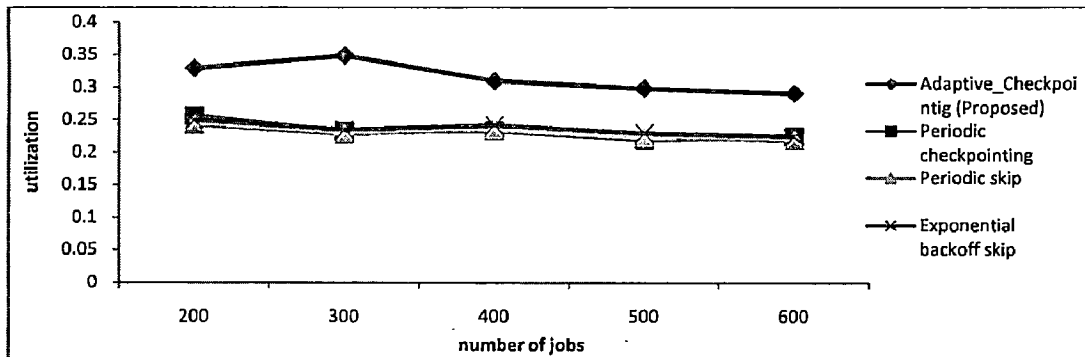


Figure 5.12 Comparison for utilization parameter for varying number of jobs

Figure 5.13 gives a radar plot for mean values for both adaptive checkpointing and periodic checkpointing. Values for adaptive checkpointing relative to periodic checkpointing are -22% for makespan, -4.84 for flowtime, -12.19% for average bounded slowdown, +22% for utilization, -15% work lost due to failures, -2.4% for number of checkpoints taken. Negative percentages represent the percent by which the value is smaller and positive percentages represent the percent by which the value is greater compared to the corresponding value for periodic checkpointing.

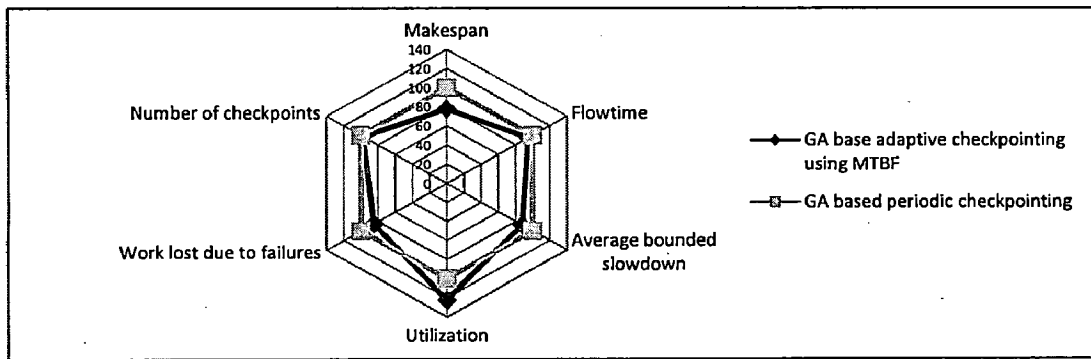


Figure 5.13 Overall comparison between periodic checkpointing and MTBF based adaptive checkpointing technique

### 5.4 GA-based Adaptive Fault Tolerance Using Fault Ratios of Resources

In this subsection 100 jobs are submitted to 10 clusters each with 64 processing elements. Shape parameters are .7, .7, .7, .7, 1, 1, 1, 1.5, 1.5, 1.5, respectively for 10 clusters. The checkpoint overhead is taken as 360 seconds. Others assumptions and parameters are same as subsection 5.3.

Figure 5.14 gives makespan comparison between adaptive checkpointing, periodic checkpointing, periodic skip and exponential backoff skip.

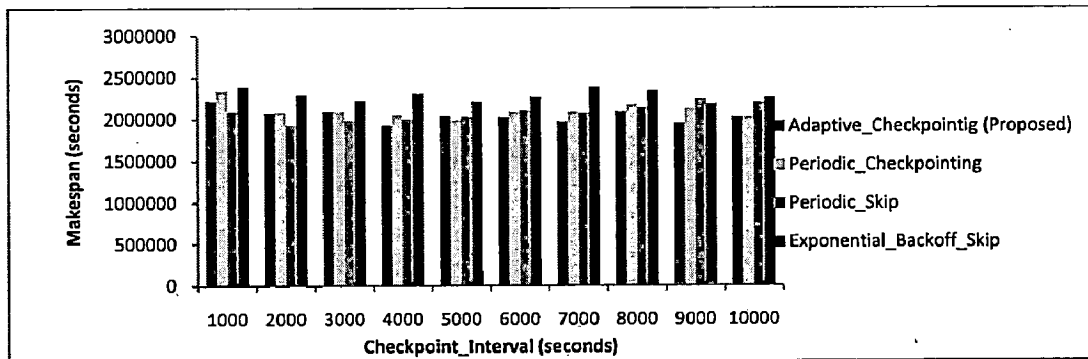


Figure 5.14 Comparison for makespan parameter

Adaptive checkpointing technique performs better than periodic checkpointing for almost all checkpointing intervals. Only for interval equal to 5000sec performance is slightly worse. This is due to 5000sec being the optimal checkpoint interval for the parameters used for simulation. Behavior of periodic skip and exponential backoff skip is similar to as in subsection 4.1.

Figure 5.15 gives the work lost due to failures for the four techniques. Adaptive checkpointing technique loses less work due to failures compared to periodic checkpointing technique due to adaptive change in

checkpoint interval depending on the failure conditions of resources. Periodic skip and exponential backoff skip techniques have very high values for work lost which is the prime reason for their poor performance.

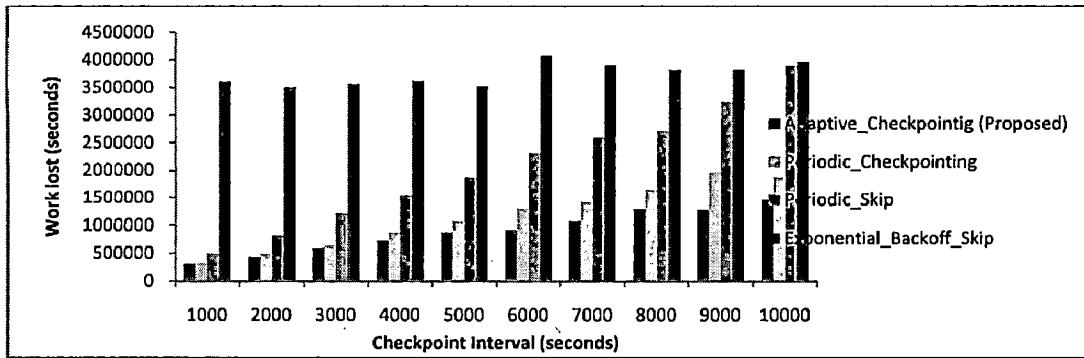


Figure 5.15 Comparison for flowtime parameter

Figures 5.16, 5.17, 5.18, 5.19 give results respectively for flowtime, number of checkpoints taken, average bounded slowdown and utilization. All these graphs shows superior performance of adaptive checkpointing technique compared to other three techniques.

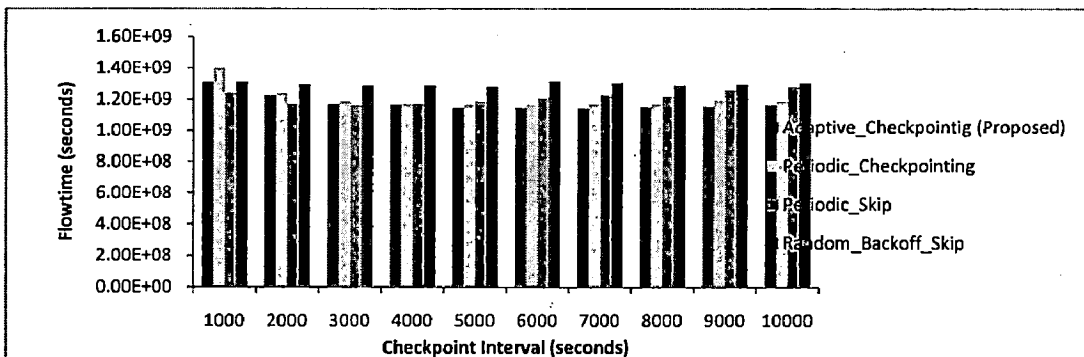


Figure 5.16 Comparison for flowtime parameter



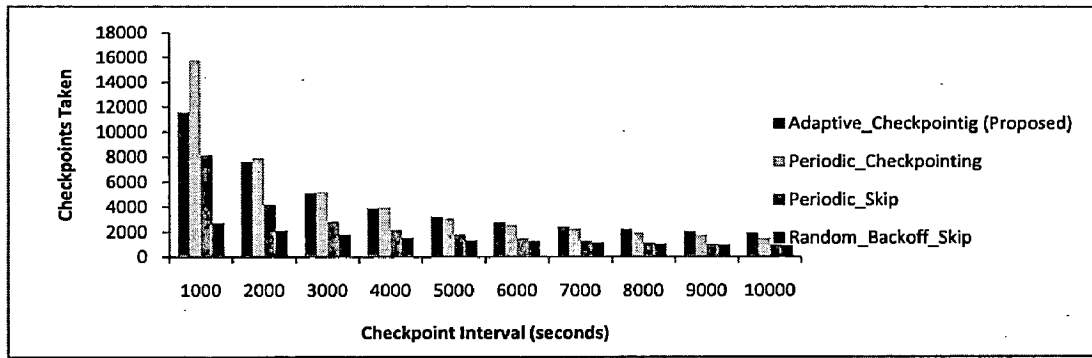


Figure 5.17 Comparison for number of checkpoints taken parameter

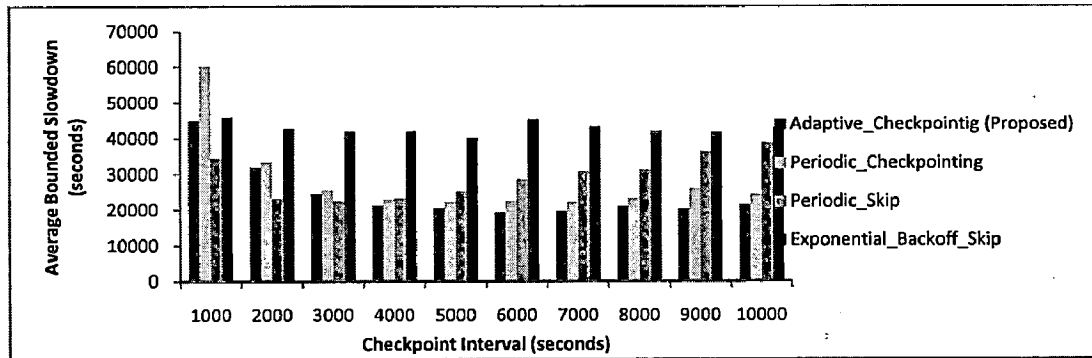


Figure 5.18 Comparison for average bounded slowdown parameter

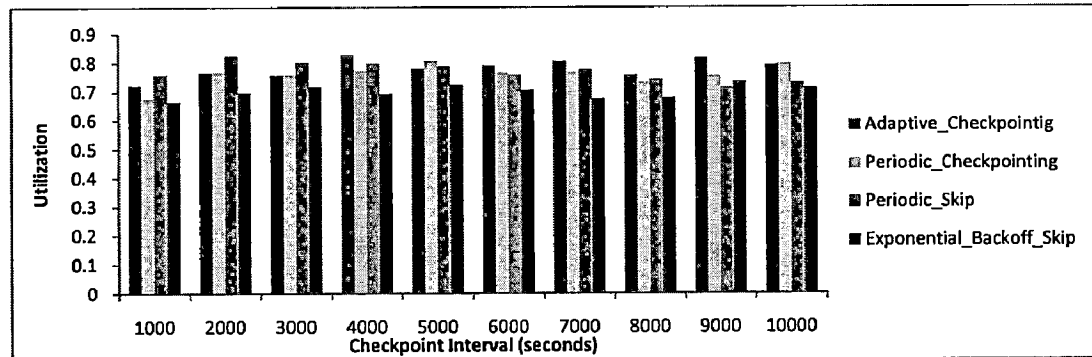


Figure 5.19 Comparison for utilization parameter

Figure 5.20 gives a radar plot for mean values for both adaptive checkpointing and periodic checkpointing. Values for adaptive checkpointing relative to periodic checkpointing are -3.13% for makespan, -13.43 for flowtime, -13.41% for average bounded slowdown, +2.65% for utilization, -22.51% for work lost due to failures, -7.21% for number of checkpoints taken.

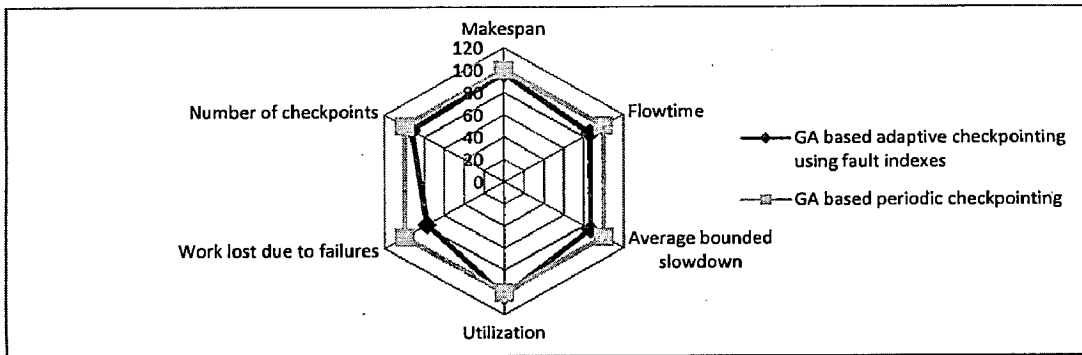


Figure 5.20 Overall comparison between fault ratio based adaptive checkpointing and periodic checkpointing

### 5.5 Ant Colony Based Adaptive Checkpointing Using MTBF of Resources

This section compares ant colony based adaptive checkpointing with ant colony based periodic checkpointing technique. Figure 5.21 shows values for makespan parameter. MTBF based checkpointing has lesser makespan compared to periodic checkpointing. Figure 5.16 gives results for work lost due to failures.

As shown in figures 5.23, 5.24, 5.25, 5.26 MTBF based ant colony techniques have lesser flowtime, average bounded slowdown, number of checkpoints taken and higher utilization compared to periodic checkpointing technique.

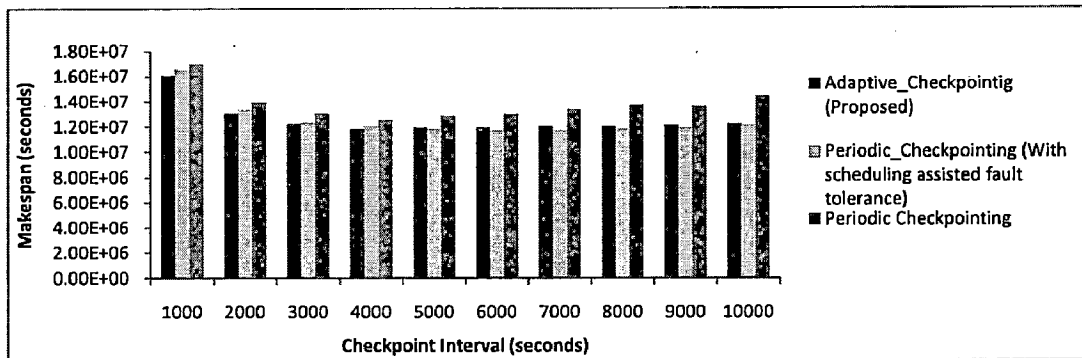


Figure 5.21 Comparison for makespan parameter for adaptive checkpointing, scheduling assisted fault tolerant periodic and periodic checkpointing

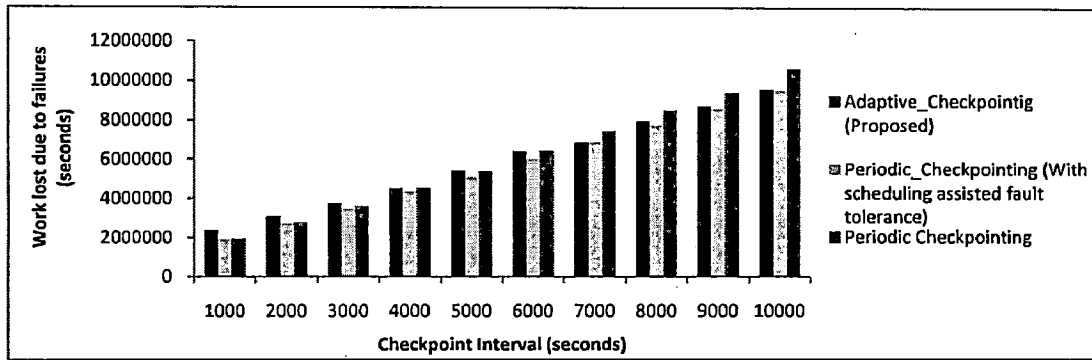


Figure 5.22 Comparison for work lost due to failures parameter

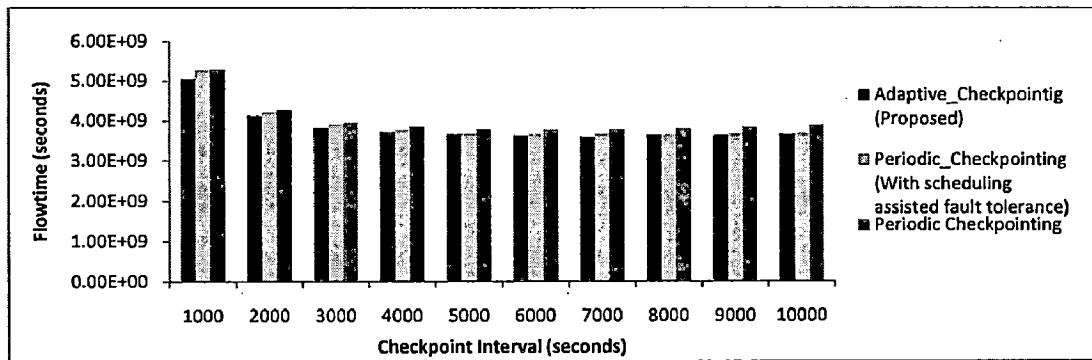


Figure 5.23 Comparison for flowtime parameter

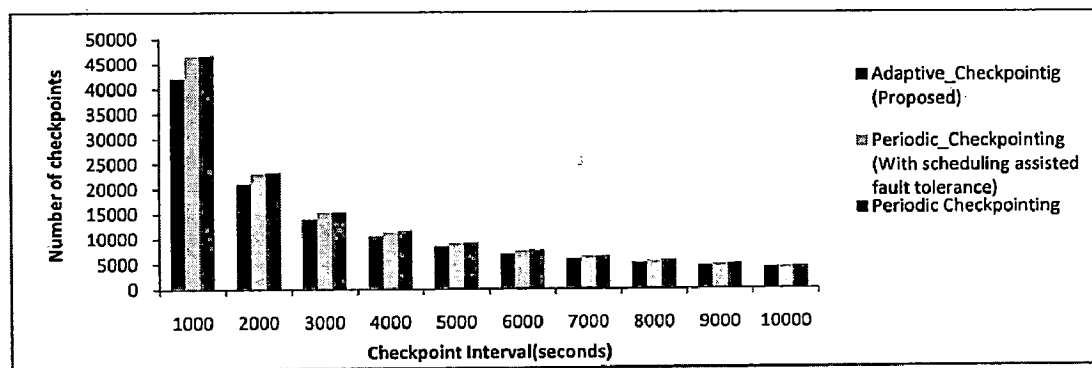


Figure 5.24 Comparison for number of checkpoints taken parameter

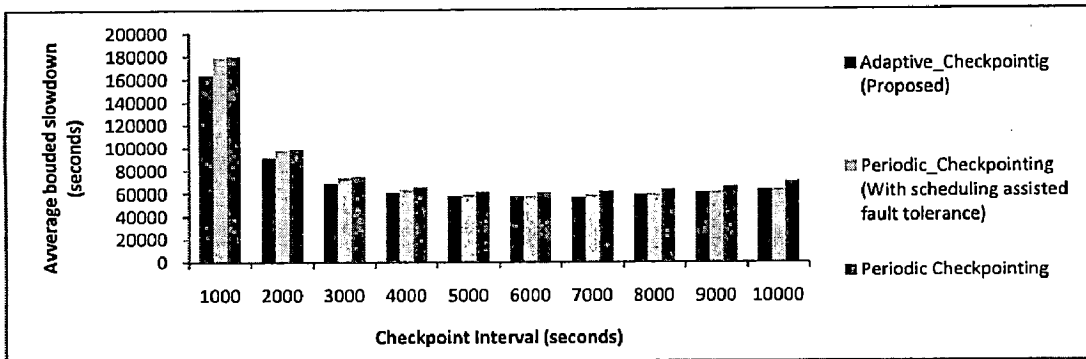


Figure 5.25 Comparison for average bounded slowdown parameter

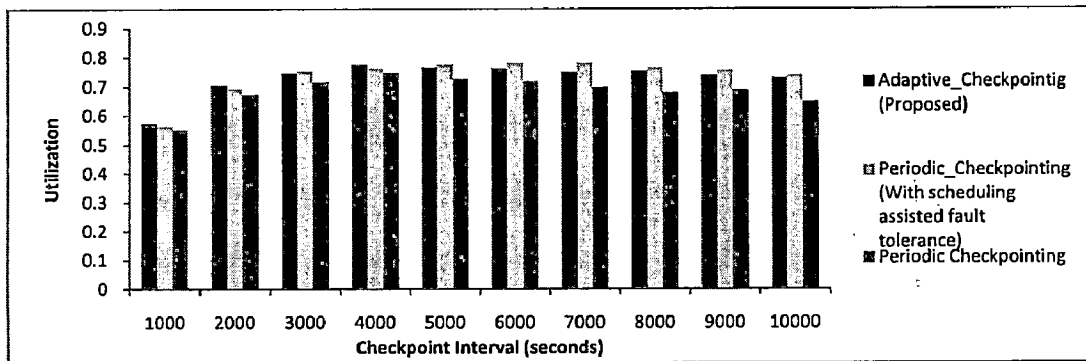


Figure 5.26 Comparison for utilization parameter

Figure 5.27 gives a holistic view of comparison for all parameters using a radar plot. Values for adaptive checkpointing relative to periodic checkpointing are -8.7% for makespan, -4.22 for flowtime, -7.8% for average bounded slowdown, +6.569% for utilization, -3.24% for work lost due to failures, -10% for number of checkpoints taken.

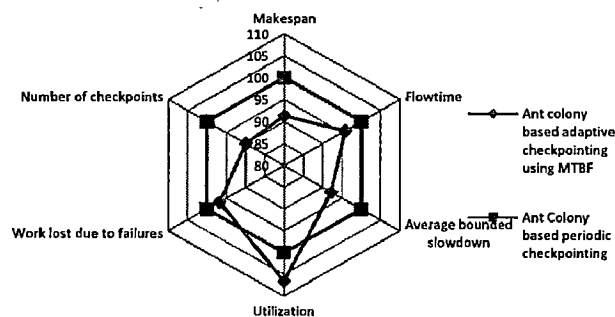


Figure 5.27 Overall comparison between MTBF based adaptive checkpointing and periodic checkpointing

### 5.6 Ant Colony Based Adaptive Checkpointing Using Fault Ratios of Resources

The simulated Grid environment consists of 5 clusters with 64 PEs in each cluster. MTBF of resources is varied from 3 to 28 hours. The overhead associated with individual checkpoints is 360 seconds. Other parameters and assumptions are same as in subsection 5.3. As shows in figures 5.28, 5.29, 5.30, 5.31, 5.32 the adaptive checkpointing technique has poor performance for makespan, flowtime, average bounded slowdown for initial four checkpointing intervals. This is due to excessive checkpointing performed which is clear from figure 5.31. For rest of the checkpointing intervals adaptive checkpointing has superior performance compared to periodic checkpointing technique.

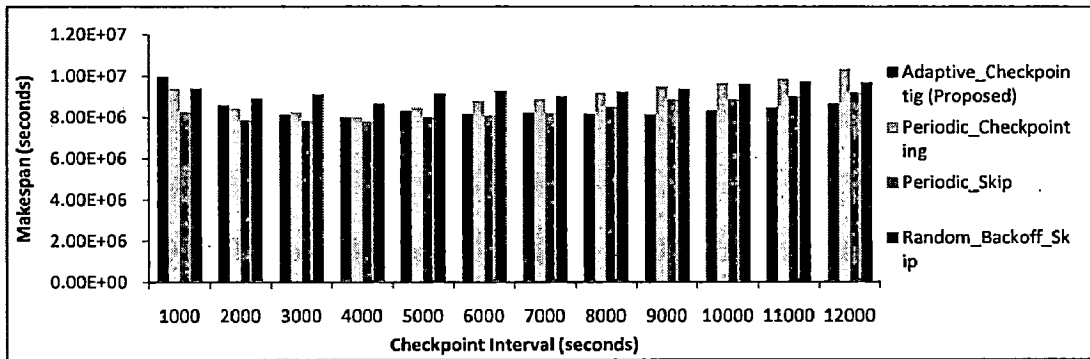


Figure 5.28 Comparison for makespan parameter between adaptive checkpointing, periodic checkpointing and skipping checkpointing techniques

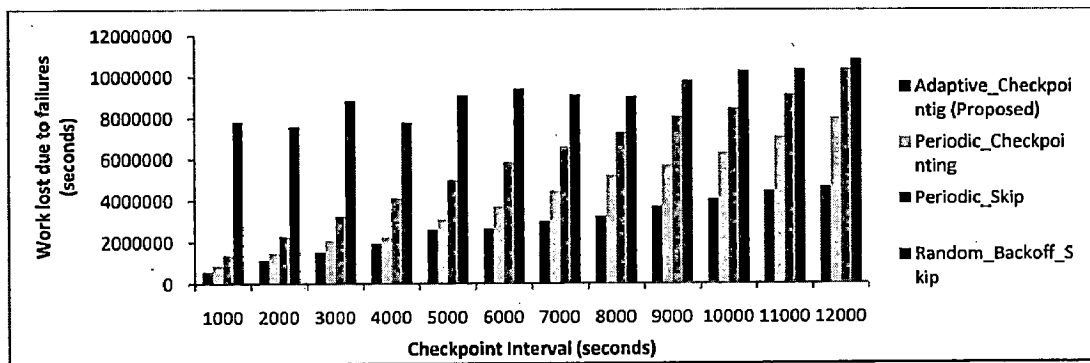


Figure 5.29 Comparison for work lost due to failures parameter

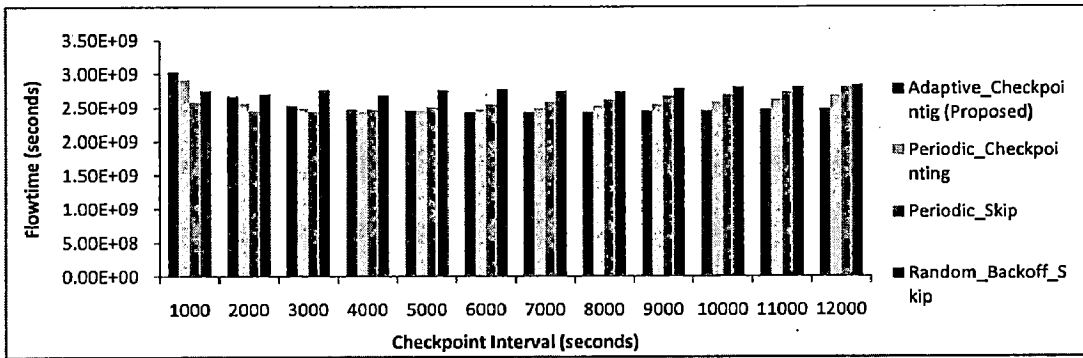


Figure 5.30 Comparison for flowtime parameter

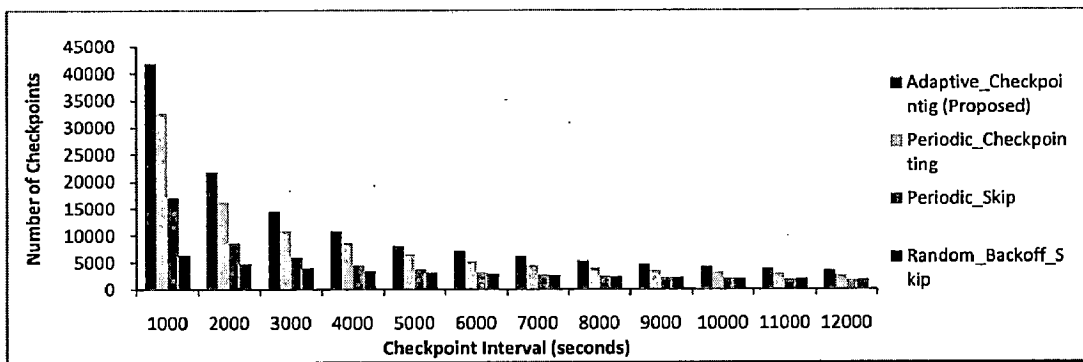


Figure 5.31 Comparison for number of checkpoints taken parameter

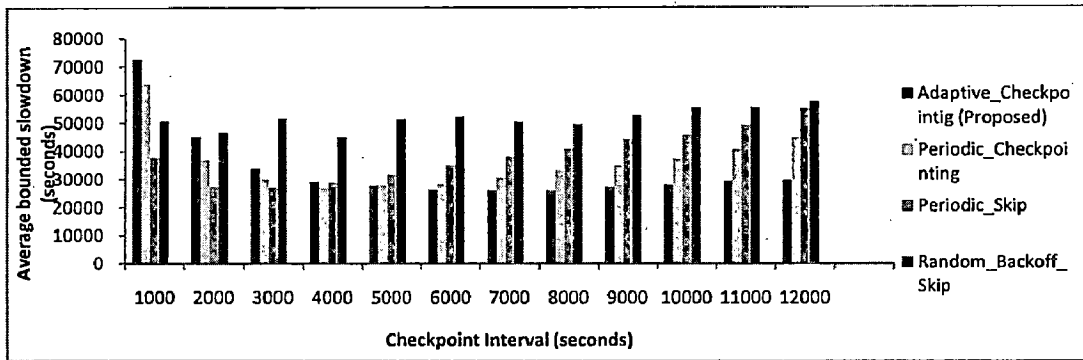


Figure 5.32 Comparison for average bounded slowdown parameter

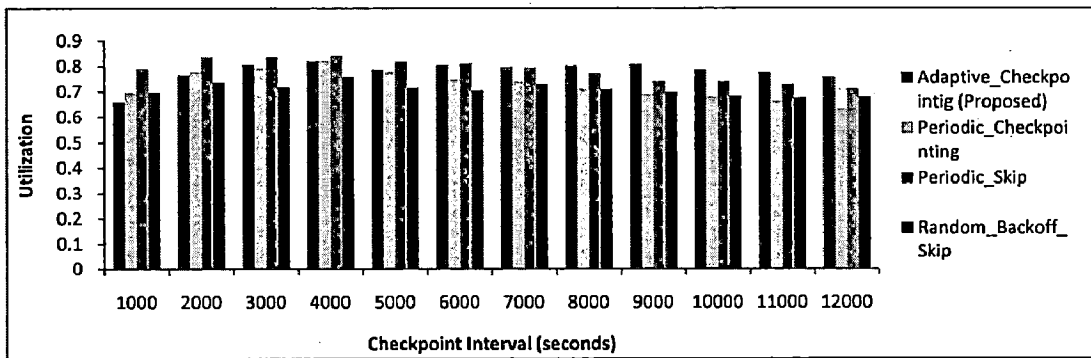


Figure 5.33 Comparison for utilization parameter

Figure 5.34 gives a holistic view of the comparison between adaptive checkpointing and periodic checkpointing for all the parameters considered using a radar plot. Values for adaptive checkpointing relative to periodic checkpointing are -6.95% for makespan, -1.55 for flowtime, -7.7% for average bounded slowdown, +7.17% for utilization, -28.2% for work lost due to failures, +30% for number of checkpoints taken.

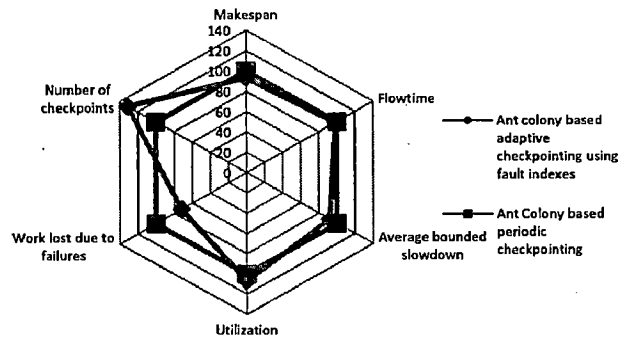


Figure 5.28 Overall comparison between MTBF based adaptive checkpointing and periodic checkpointing

### 5.7 GA-based Adaptive Fault Tolerance Using Fault Ratios of Resources for spatially and temporally Correlated Failures

Spatial correlation means that failures are localized in space i.e. in a given time frame nodes in one cluster may be more prone to failures compared to nodes in other clusters. Temporal correlation can be of following two types

- i) If a node has not failed for a long time then there is less probability that it will fail in the near future.
- ii) If a node has failed recently then there is high probability that it will fail in the near future.

To simulate above characteristics of failures a unique experiment has been developed. The simulated Grid environment consists of two clusters each with 64 processing elements. First failures are generated from Weibull distribution. Then for one resource first half of the failures generated is sorted in ascending order and the second half in descending order. For the other resource first half is sorted in descending order and second half in ascending order. The sorting of failures for two resources in different order means that in a time frame nodes in one cluster are more prone to failures compared to others (spatial correlation). Also sorting two halves of failures for a resource in alternate order helps achieve the above properties of temporal correlation. This technique is similar to [40].

A total of 32 batches of 100 jobs each requiring 6 hours of computation are submitted one after the completion of previous. Each job is composed of 64 processes and an entire cluster is assumed to be allocated to a job for its execution. Coordinated checkpointing is used for performing checkpoints and it is assumed that even if one of the processes of a job fails, all processes of that job are restarted from the latest successively saved checkpoint. Size of checkpoint interval is 3000 seconds and each checkpoint has an overhead of 180 seconds.

Figures 5.35 to 5.40 compares adaptive checkpointing with periodic checkpointing respectively for makespan, work lost due to failures, flowtime, number of checkpoints taken, average bounded slowdown, and utilization. Peaks in curves for makespan, flowtime, average bounded slowdown, and dip for utilization are where one resource has higher rate of failures compared to others (spatial correlation) as is clear from figure 5.41.. Adaptive checkpointing is able to take advantage of this correlation and gives superior performance compared to periodic checkpointing. Rising above the peak is where temporal correlation is achieved. Adaptive checkpointing gives better performance for this situation. Falling down a peak is where anti-temporal correlation is simulated and adaptive checkpointing gives a little poorer performance compared to periodic checkpointing for this case.

Figure 5.42 gives resource allocations for each batch of jobs for both techniques compared. Figure 5.43 gives completion time of resources for each batch of jobs for both the techniques.

Figure 5.44 gives a holistic view of comparison between the two techniques for all parameters using a radar plot. Values for adaptive checkpointing relative to periodic checkpointing are -3.5% for makespan, -3.1 for flowtime, -17.84% for average bounded slowdown, +1.395% for utilization, -24.84% for work lost due to failures, -5.76% for number of checkpoints taken.



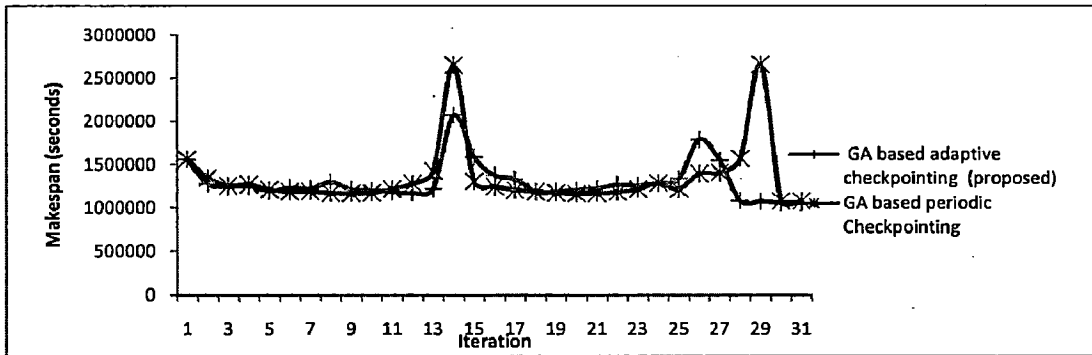


Figure 5.35 Makespan comparison for adaptive checkpointing and periodic checkpointing

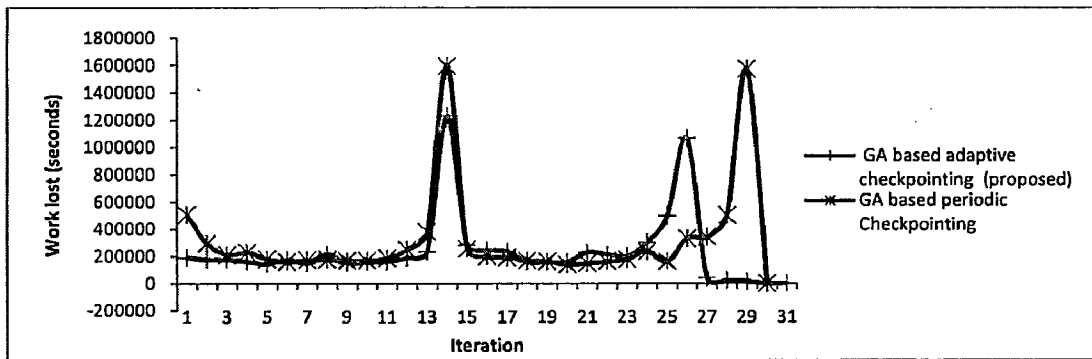


Figure 5.36 Comparison for work lost due to failures

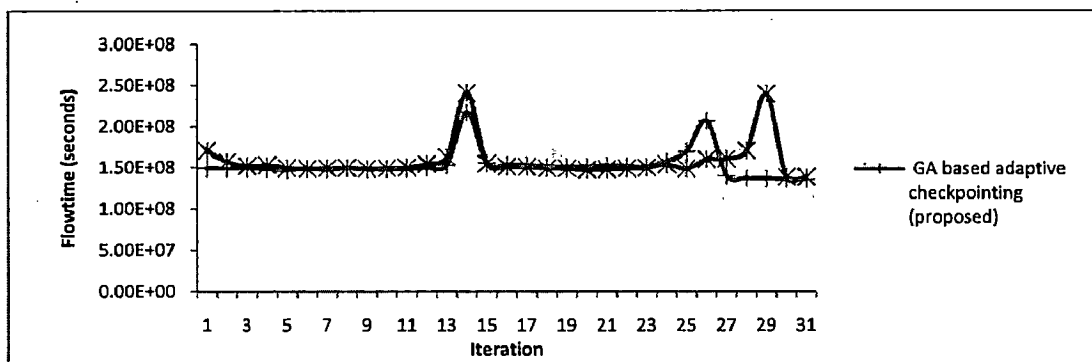


Figure 5.37 Comparison for flowtime parameter

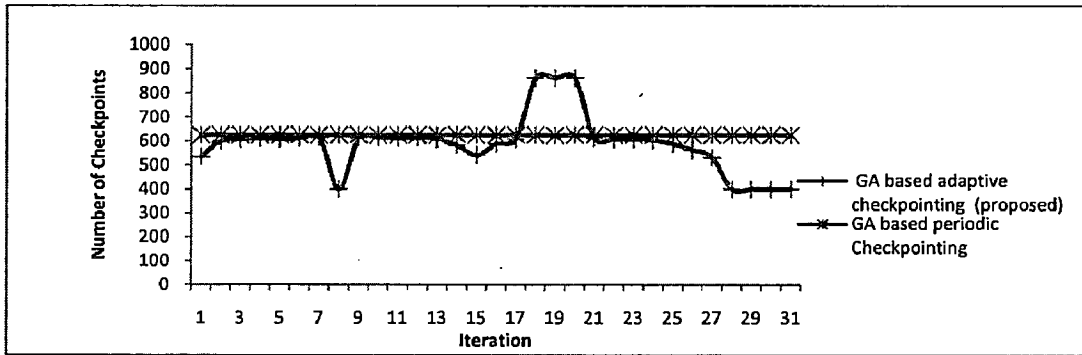


Figure 5.38 Comparison for number of checkpoints parameter

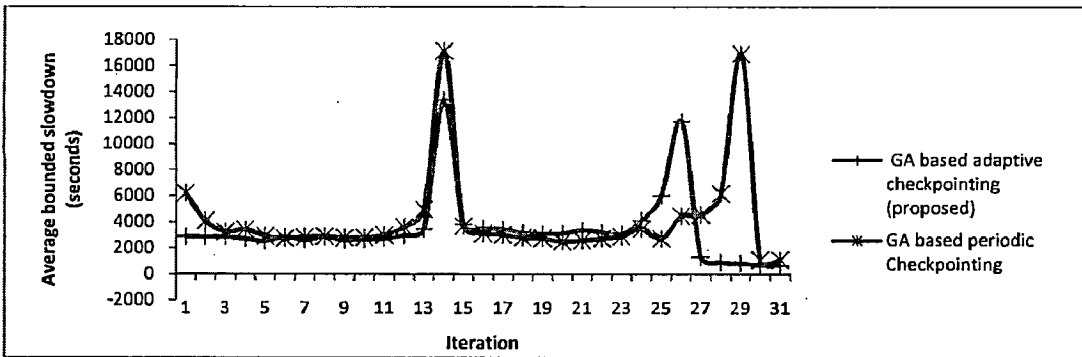


Figure 5.39 Comparison for average bounded slowdown parameter

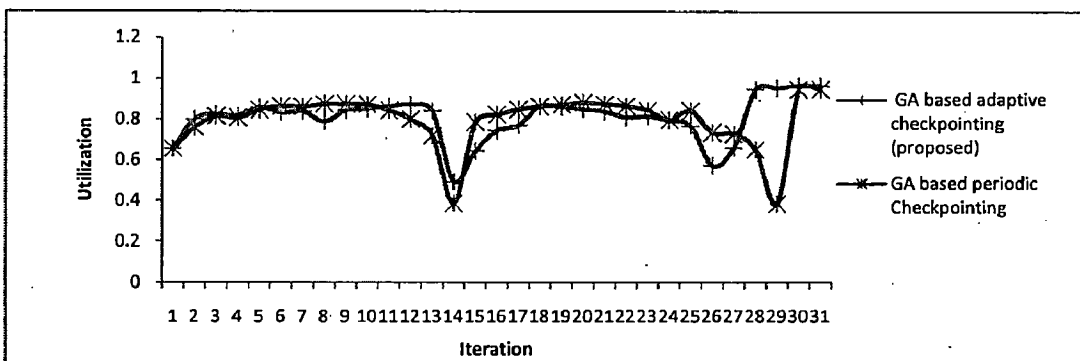


Figure 5.40 Comparison for utilization parameter

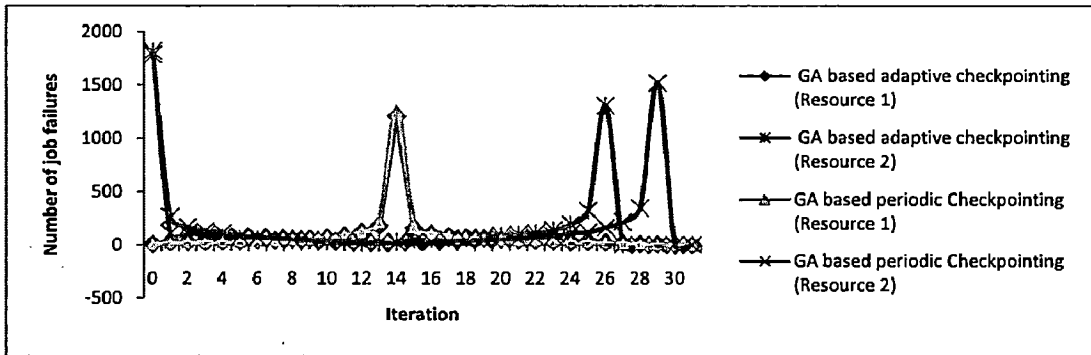


Figure 5.41 Number of fault occurrences for each resource

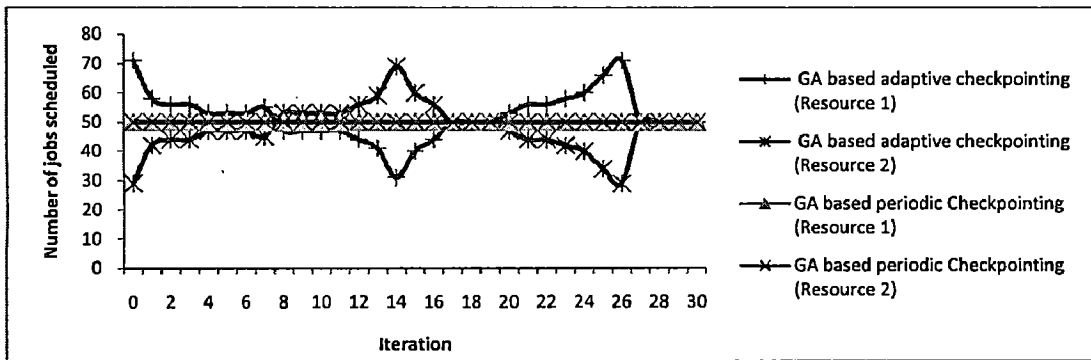


Figure 5.42 Number of jobs scheduled on each resource

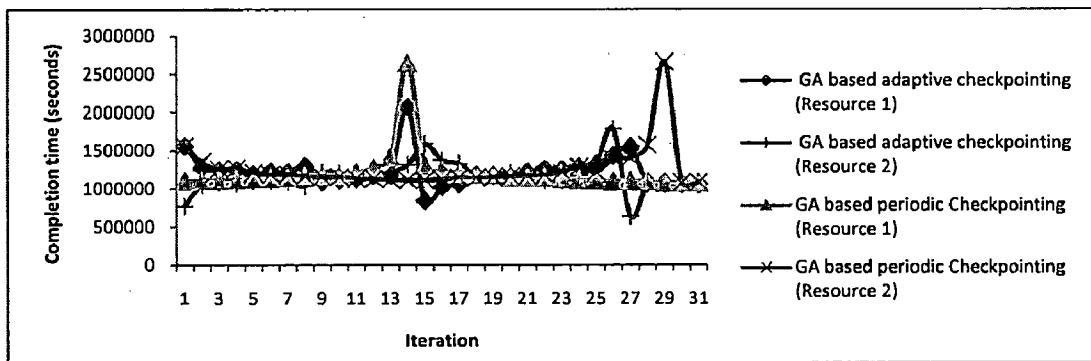


Figure 5.43 Completion times of each resource

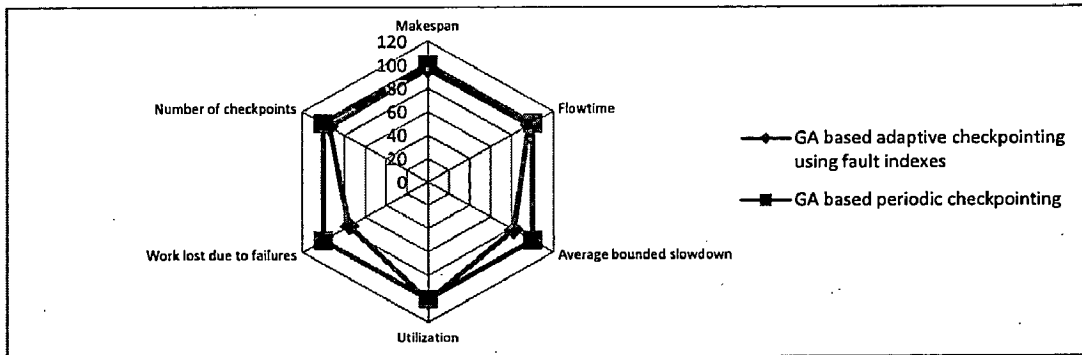


Figure 5.44 Overall comparison of periodic and adaptive checkpointing

### 5.8 Ant Colony-based Adaptive Fault Tolerance Using Fault Ratios of Resources for spatially and temporally Correlated Failures

This subsection compares ant colony based adaptive checkpointing with ant colony based periodic checkpointing for spatially and temporally correlated failures. Parameters and underlying assumptions are same as in subsection 5.7.

Figures 5.45 to 5.50 compares the two techniques respectively for makespan, work lost due to failures, flowtime, number of checkpoints taken, average bounded slowdown, and utilization for each batch of jobs. Peaks for makespan, work lost, flowtime, average bounded slowdown, and dip for utilization is where spatial correlation is simulated. Rising up a peak simulates temporal correlation and falling down a peak simulates anti-temporal correlation. Other interpretations are same as in subsection 4.5.

Figures 5.51, 5.52, 5.53 respectively shows number of fault occurrences of each resource, job allocations of each resource, and completion time of each resource for both techniques for each batch of jobs.

Figure 5.54 gives a holistic view of comparison between the two techniques for all parameters using a radar plot. Values for adaptive checkpointing relative to periodic checkpointing are -2.3% for makespan, -3.8 for flowtime, -20.5% for average bounded slowdown, +9.7% for utilization, -25.3% for work lost due to failures, -7.52% for number of checkpoints taken.

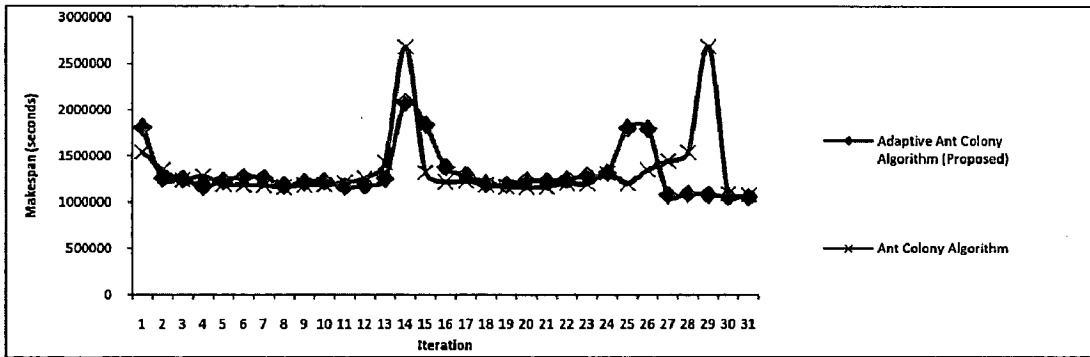


Figure 5.45 Makespan comparison for periodic and adaptive checkpointing

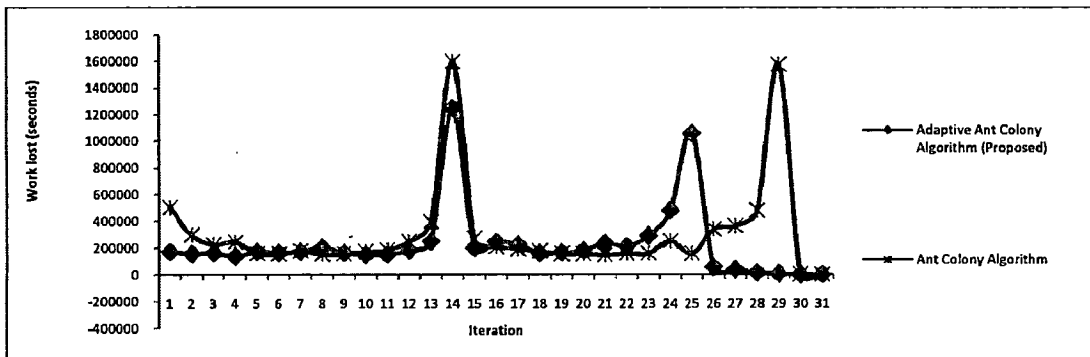


Figure 5.46 Comparison for work lost due to failures parameter

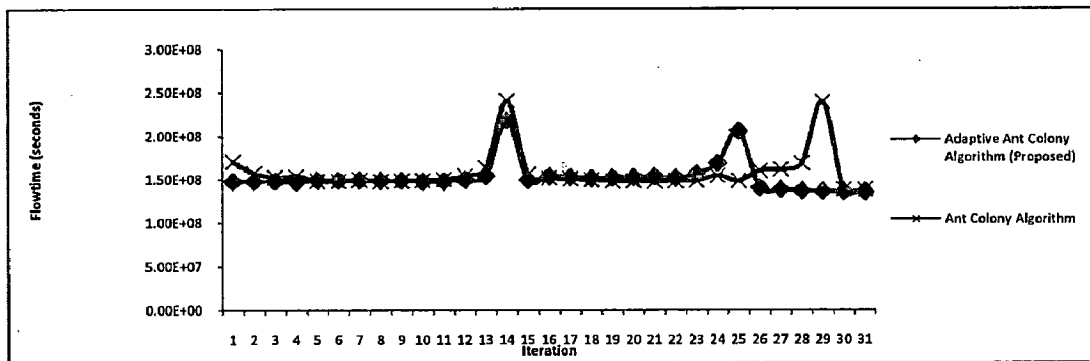


Figure 5.47 Comparison for flowtime parameter

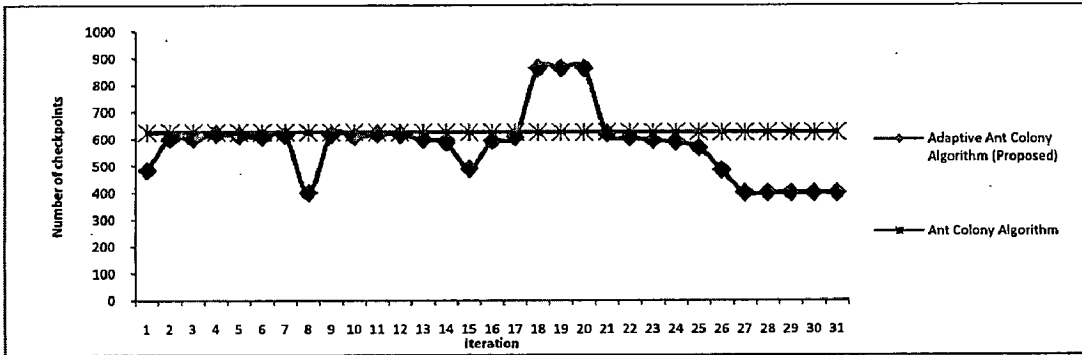


Figure 5.48 Comparison for number of checkpoints parameter

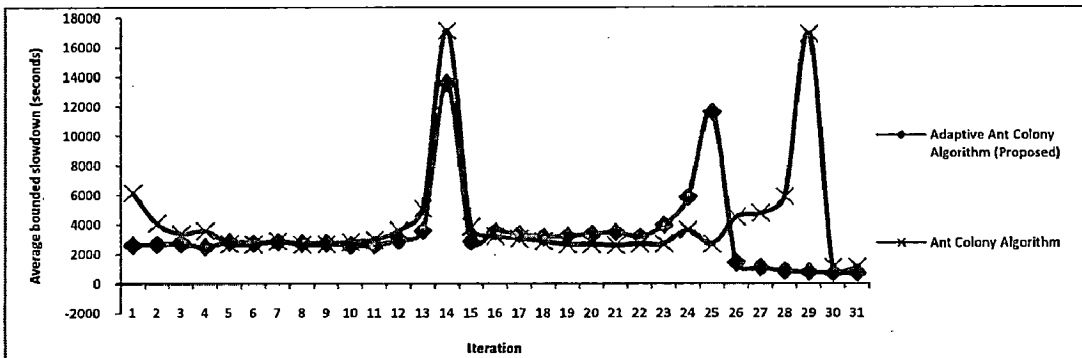


Figure 5.49 Comparison for number of average bounded slowdown parameter

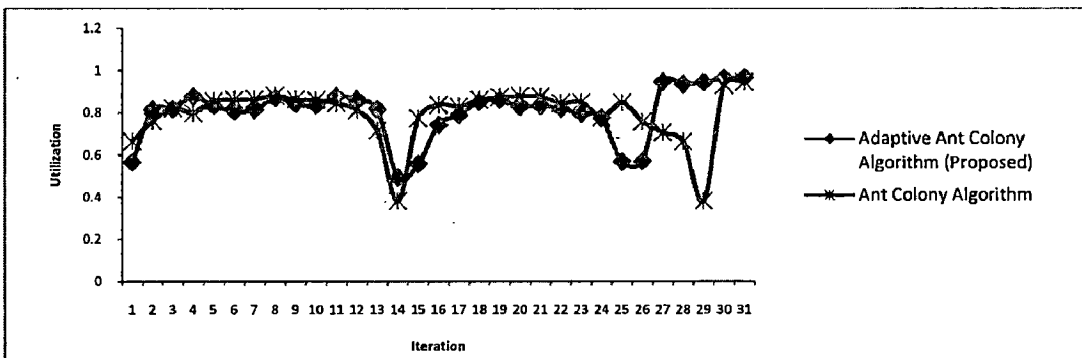


Figure 5.50 Comparison for utilization parameter

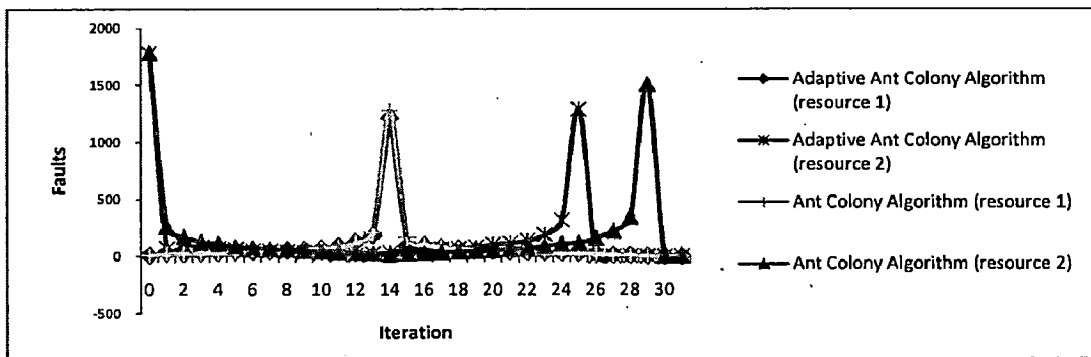


Figure 5.51 Number of fault occurrences for each resource

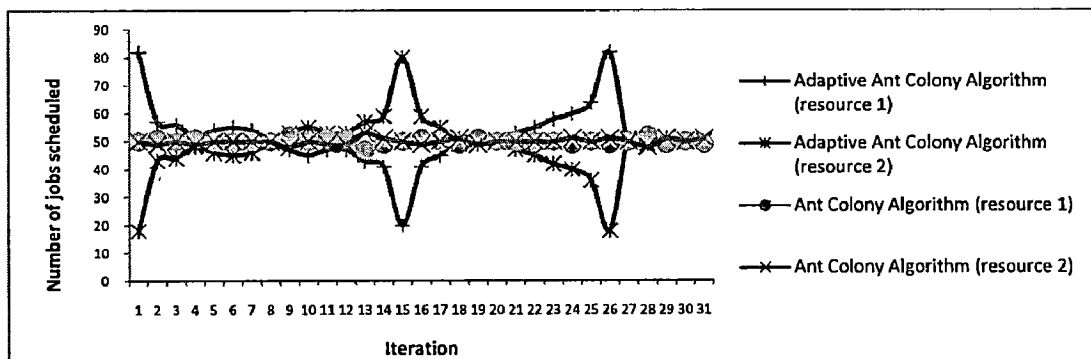


Figure 5.52 Number of jobs scheduled on each resource

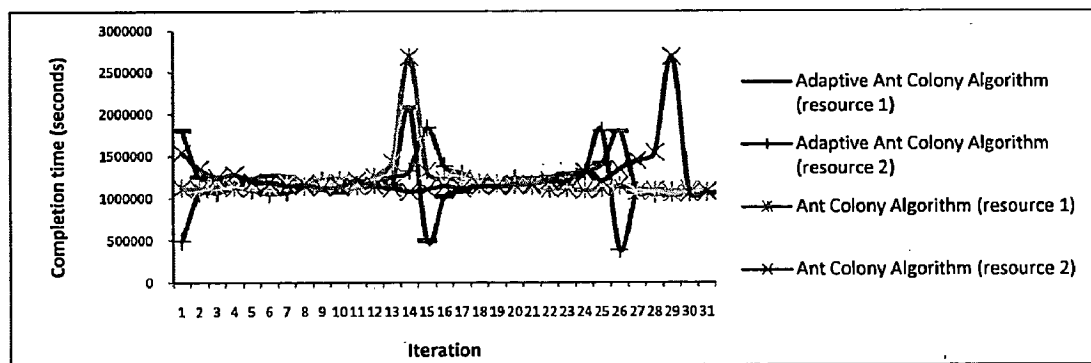


Figure 5.53 Completion times of each resource

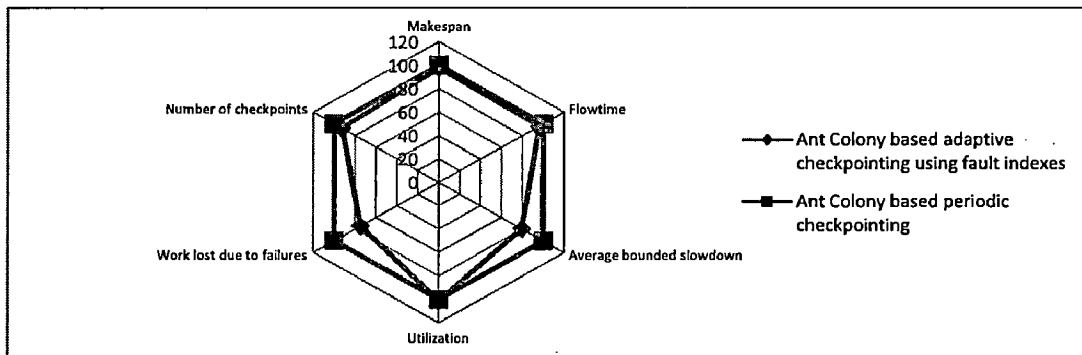


Figure 5.54 Overall comparison of periodic and adaptive checkpointing

## 5.9 Fault Index based periodic Skip

### 5.9.1 Comparison between periodic skip and fault index based periodic skip

This subsection compares periodic skip with fault index based periodic skip. A total of 200 jobs each requiring around 1 hour of computation on a single processor were submitted. Checkpointing interval was varied from 150 seconds to 400 seconds with each checkpointing operation having a cost of 30 seconds. A total of 30 resources each with a single processor are available for execution of jobs. MTBF of resources is varied from 15 minutes to 5 hour. Figures 5.55 to 5.59 compares the two techniques respectively for makespan, work lost due to failures, flowtime, number of checkpoints, and average bounded slowdown. Adaptive checkpointing performs better as it is able to adapt the skipping behavior depending on the failure conditions of resources i.e. don't skip checkpoints if failure rate is high and skip more checkpoints if failure rate is very low.

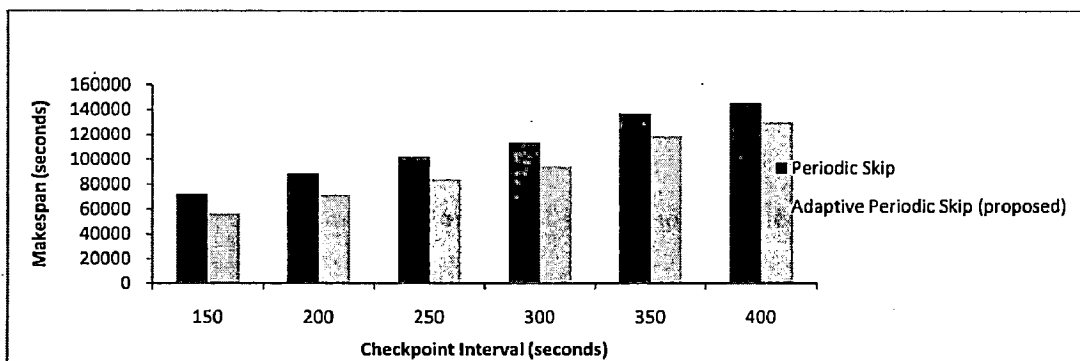


Figure 5.55 Makespan comparison for periodic skip and adaptive periodic skip



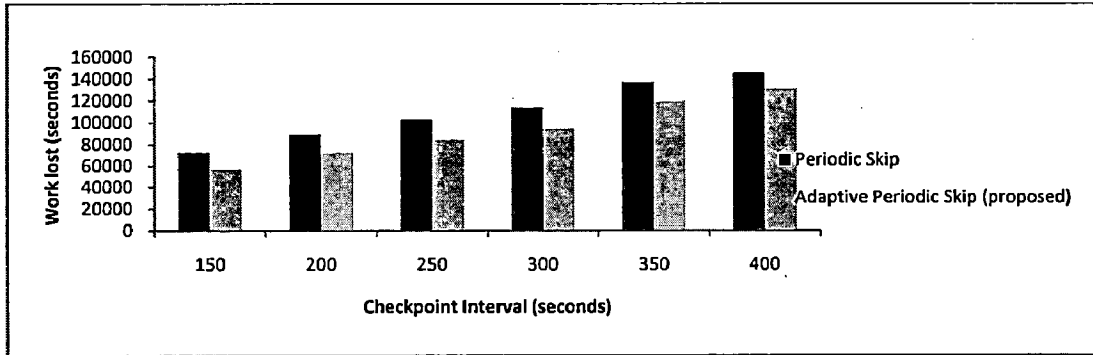


Figure 5.56 Comparison for work lost due to failures parameter

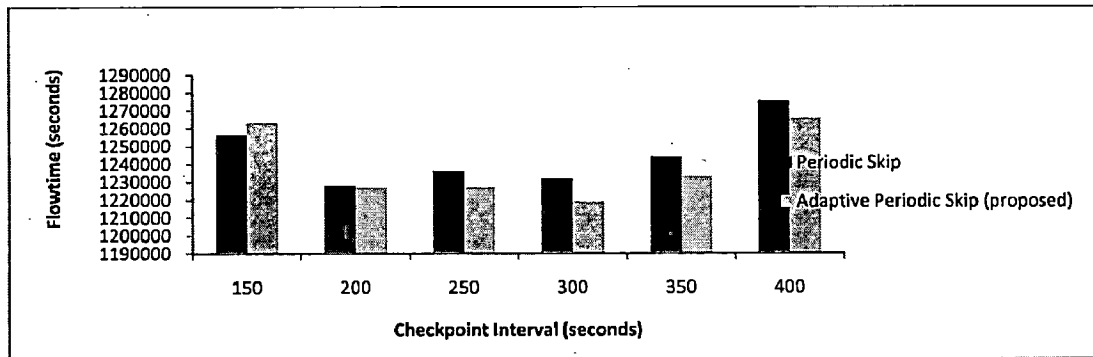


Figure 5.57 Comparison for flowtime parameter

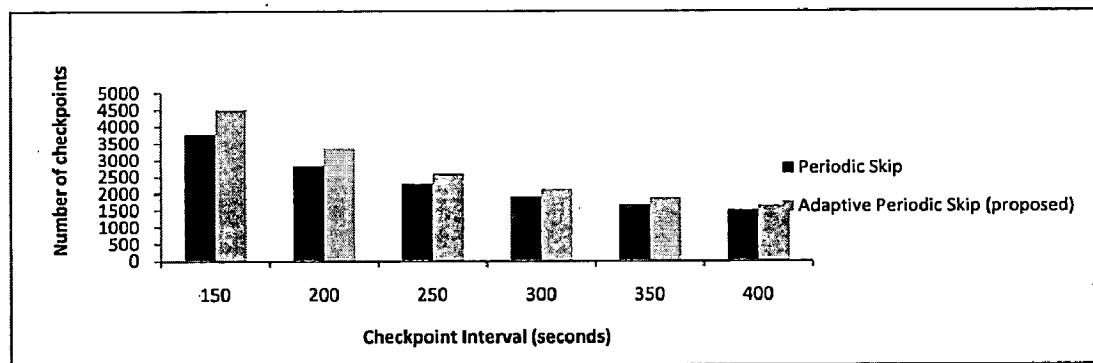


Figure 5.58 Comparison for number of checkpoints taken parameter

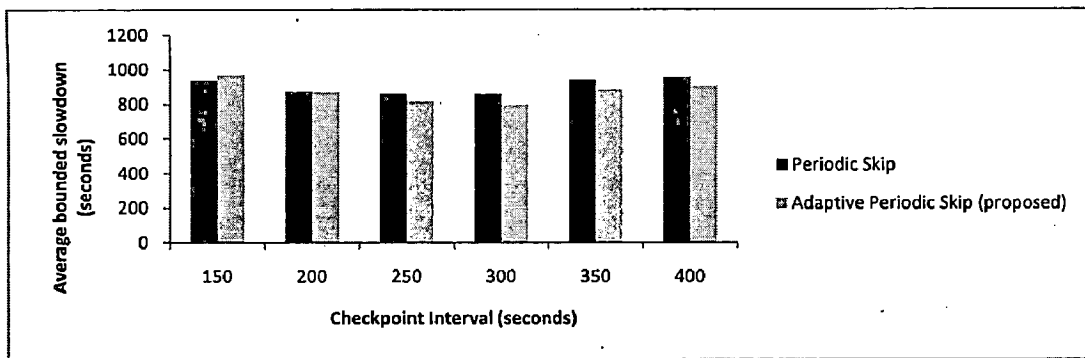


Figure 5.59 Comparison average bounded slowdown parameter

### 5.9.2 Comparison for temporally and spatially correlated failures

Parameters and underlying assumptions are same as in subsection 5.7. Figures 5.60 to 5.65 respectively compares adaptive periodic skip and periodic skip for makespan, work lost due to failures, flowtime, number of checkpoints taken, average bounded slowdown, and utilization for each batch of jobs.

Figures 5.66 and 5.67 respectively show plots for number of faults on each resource, completion time for each resource for both techniques after execution of each batch of jobs.

Figure 5.68 gives a holistic view of comparison between the two techniques for all parameters using a radar plot. Values for adaptive checkpointing relative to periodic checkpointing are -6.42% for makespan, -2.25 for flowtime, -10.1% for average bounded slowdown, +5.36% for utilization, -11.425% for work lost due to failures, -79% for number of checkpoints taken.

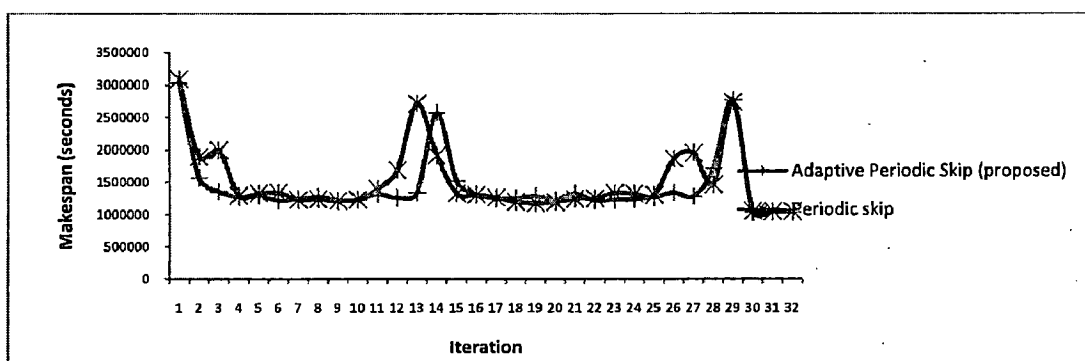


Figure 5.60 Makespan comparison for periodic and adaptive checkpointing

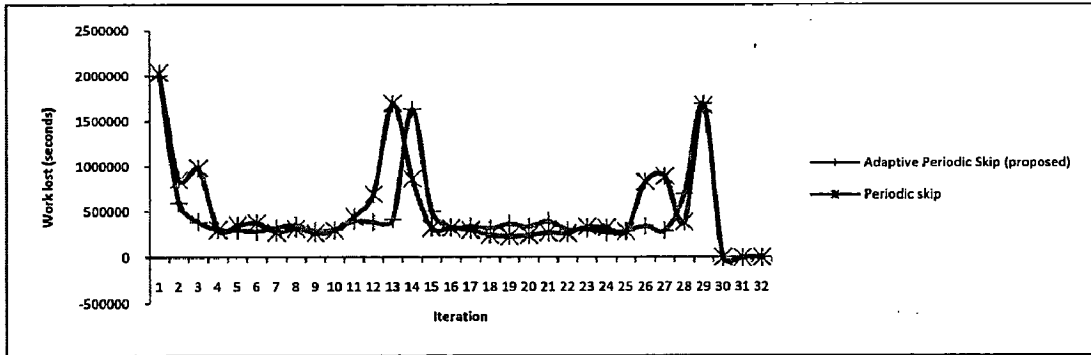


Figure 5.61 Comparison for work lost due to failures parameter

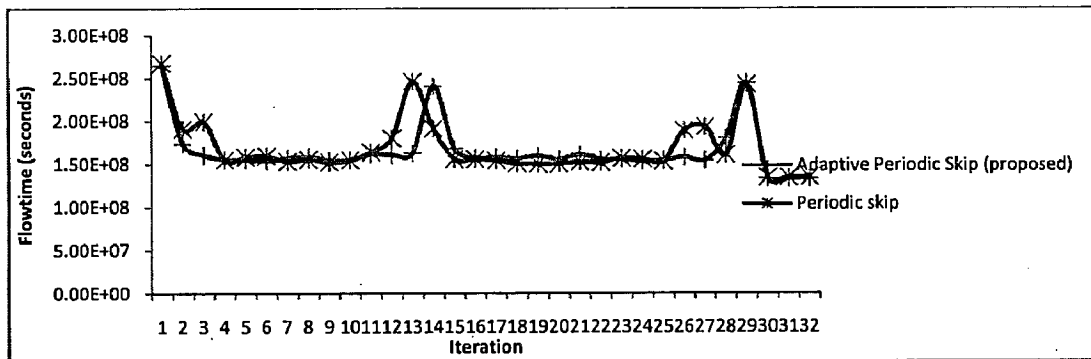


Figure 5.62 Comparison for flowtime parameter

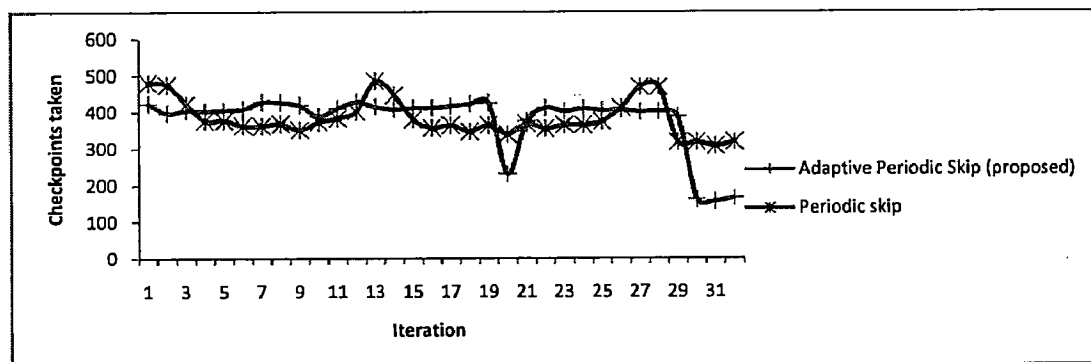


Figure 5.63 Comparison for number of checkpoints taken parameter

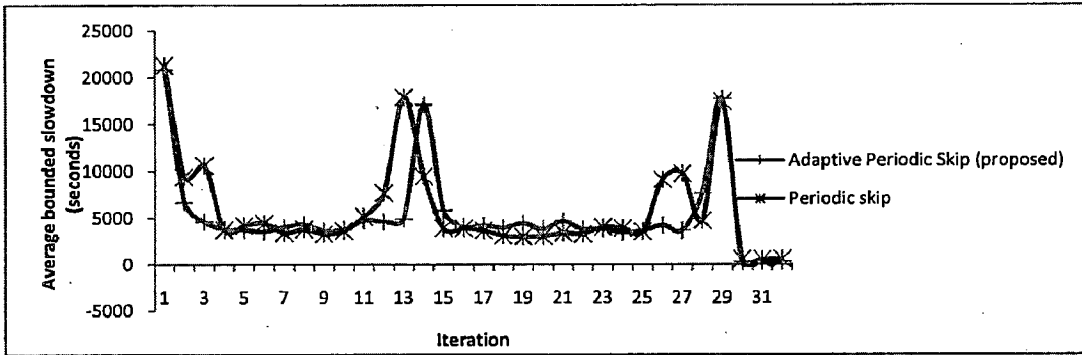


Figure 5.64 Comparison for average bounded slowdown parameter

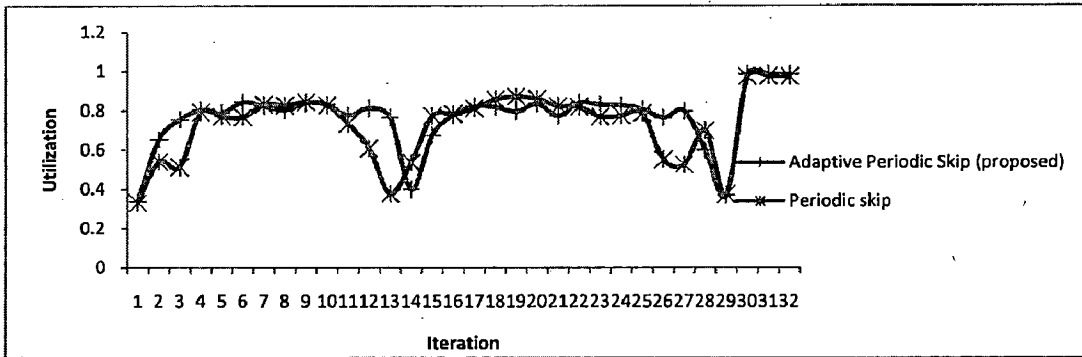


Figure 5.65 Comparison for utilization parameter

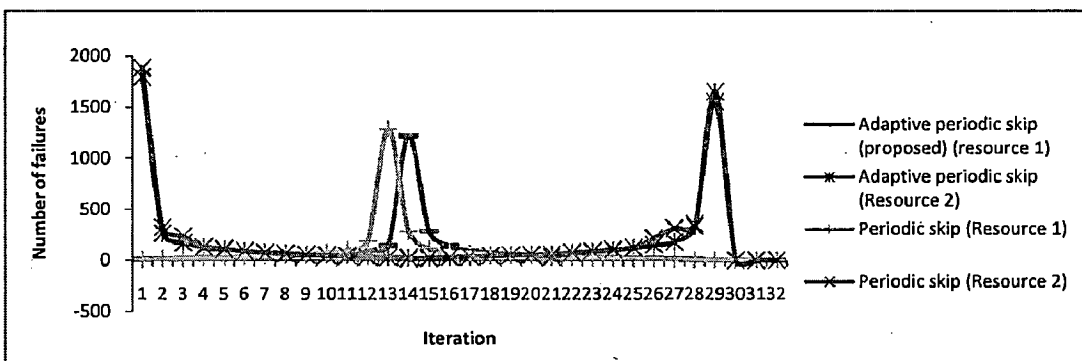


Figure 5.66 Number of fault occurrences for each resource

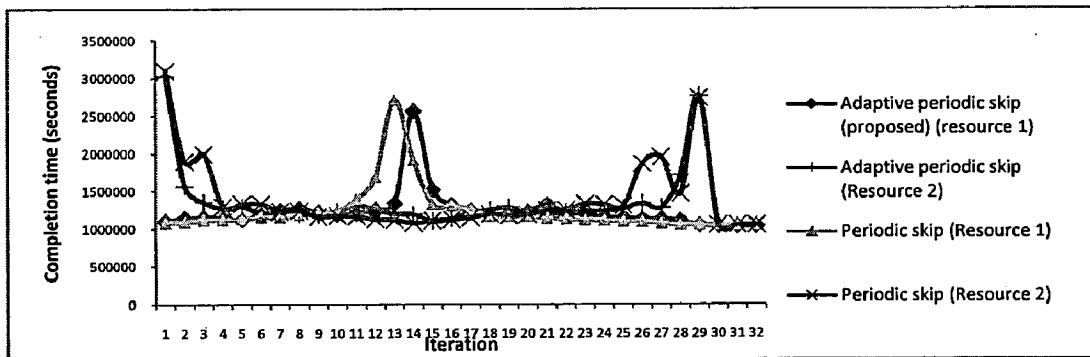


Figure 5.67 Completion times of resources

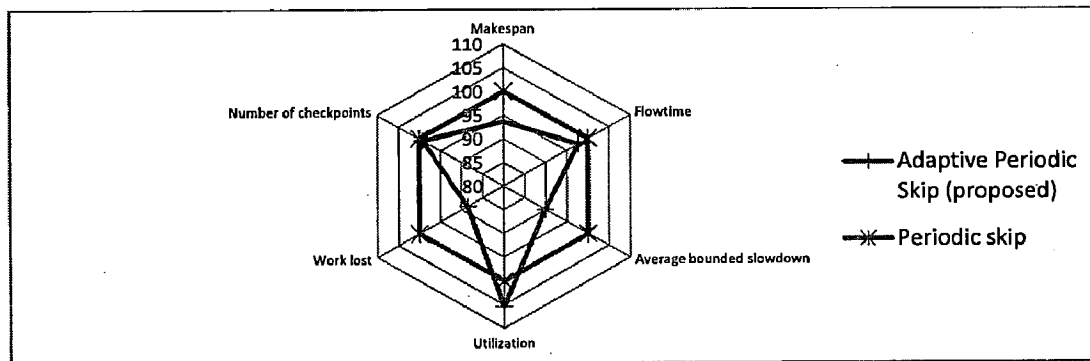


Figure 5.68 Overall comparison between periodic skip and adaptive periodic skip

### 5.10 Adaptive Checkpointing using MTBF and Last failure times of resources

This subsection compares adaptive checkpointing using MTBF of resources with periodic checkpointing for spatially and temporally correlated failures. For generating temporally and spatially correlating failures in addition to the previously mentioned method (subsection 5.7) of sorting two halves of generated failures into ascending and descending order (referred to as  $w = 2$  in this subsection), another method which divides the generated failures into four parts and sorts parts alternatively in ascending and descending order (referred to as  $w = 4$ ) is used.

Figures 5.69 to 5.74 respectively compares the two techniques for makespan, work lost due to failures, flowtime, number of checkpoints taken, average bounded slowdown, and utilization for 4 batches of jobs. It is to be observed that adaptive checkpointing gives slightly poorer value for makespan. This is acceptable as average completion time of a resource (flowtime) is significantly better for adaptive checkpointing technique. Also the adaptive checkpointing technique has significantly better values for average bounded slowdown as shown in figure 5.73.

Figures 5.75 and 5.76 respectively give completion times of resources and number of faults on each resource for both techniques for each batch of jobs.

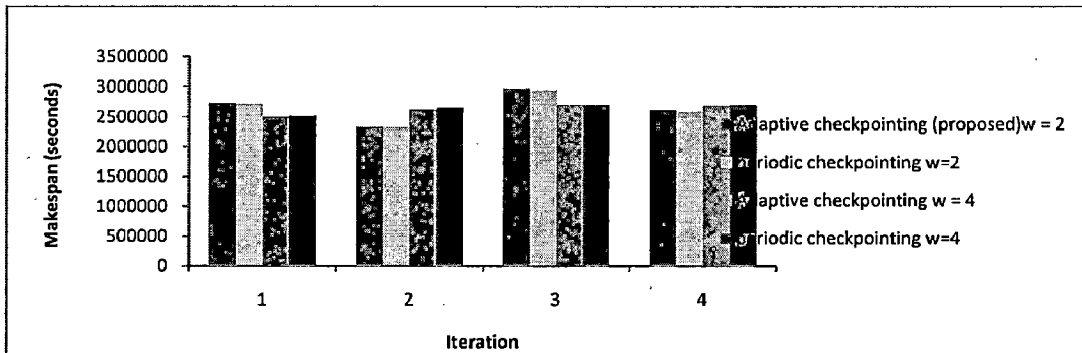


Figure 5.69 Makespan comparison between periodic and adaptive checkpointing

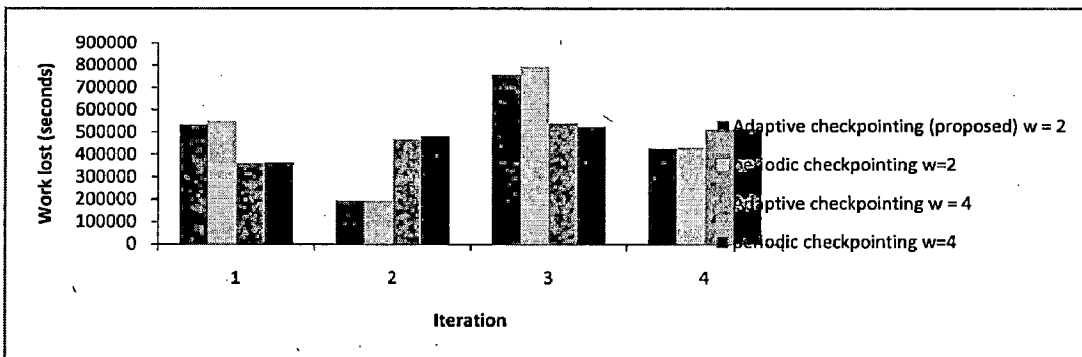


Figure 5.70 Comparison for work lost due to failures parameter

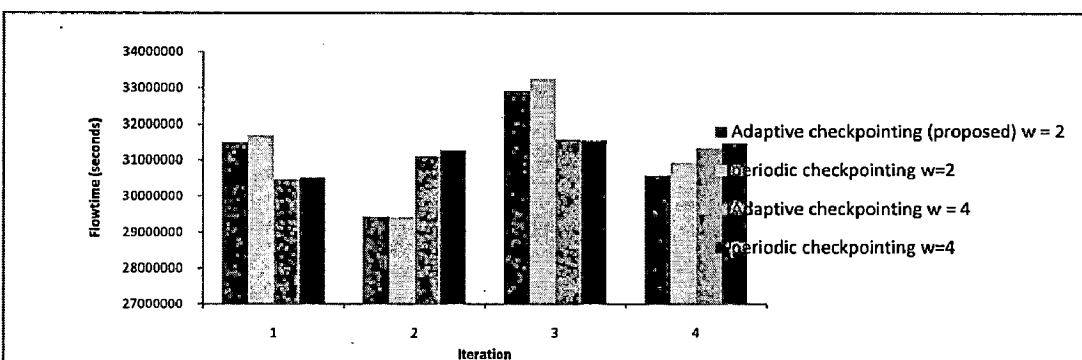


Figure 5.71 Comparison for flowtime parameter

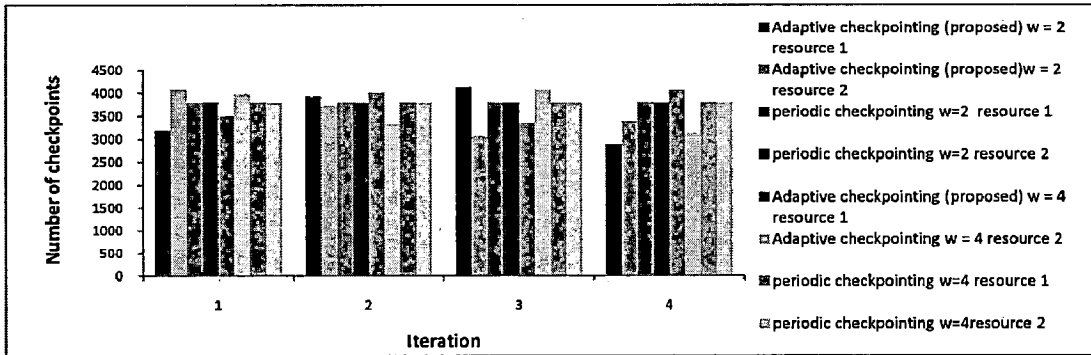


Figure 5.72 Comparison for number of checkpoints taken parameter

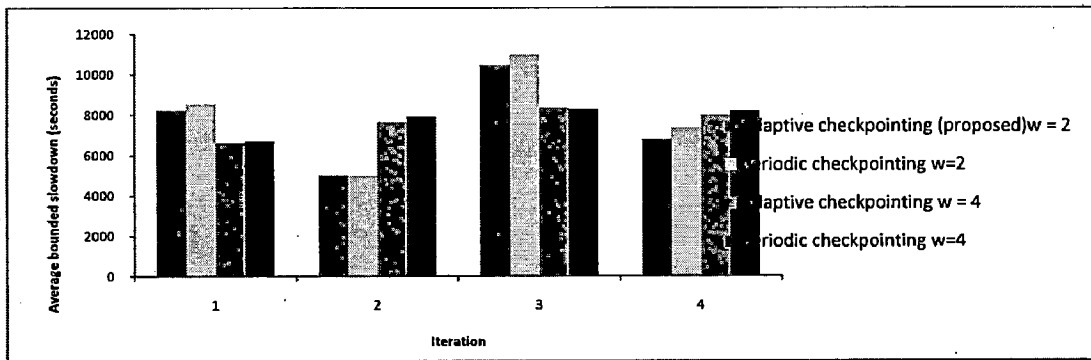


Figure 5.73 Comparison for average bounded slowdown parameter

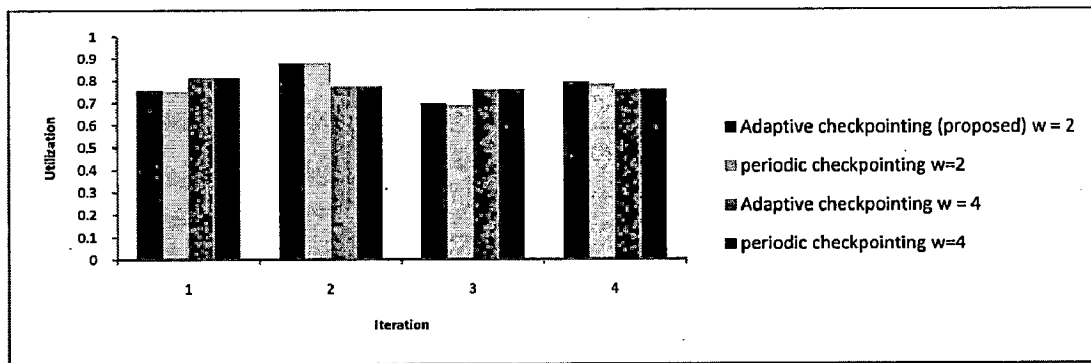


Figure 5.74 Comparison for utilization parameter

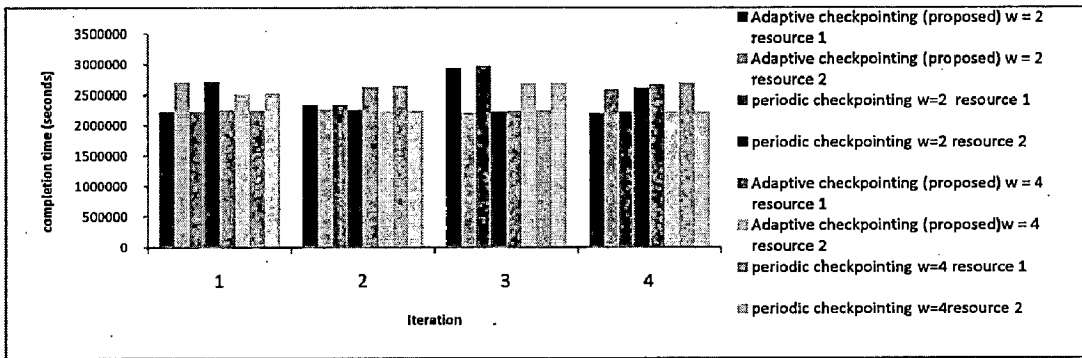


Figure 5.75 Completion times of resources

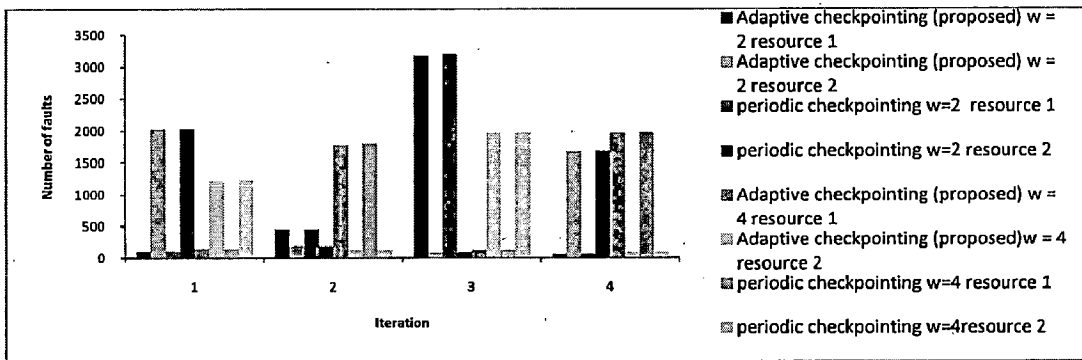


Figure 5.76 Fault occurrences on resources



### 5.11 Performance Comparison of GA based checkpointing techniques using failure traces

This section compares GA based adaptive checkpointing based on fault ratios of resources with GA based periodic checkpointing. Five failure traces overnet, skype, ucb, Notre and glow from failure trace archive [49, 50] were used for comparing the two techniques. Following configuration is used for each trace.

- Overnet trace consisted of 100 nodes with each node having 1 processing element. 12 batches of 150 jobs were submitted with each job requiring 3 hours of computation on 1 processing element. MTTR of each resource was ignored and taken to be 0.
- Skype trace consisted of 100 nodes with each node having single PE. 12 batches of 150 jobs were submitted with each job requiring around 12 hours of computation.
- Ucb trace consisted of 80 nodes with single PE on each node. 12 batches of 120 jobs were submitted with each job requiring around 7 to 8 minutes of computing time.
- Notre trace consisted of 25 nodes with single PE on each resource. 12 batches of 300 jobs were submitted with each job requiring around 6 hours of computation.
- Glow trace consisted of 87 nodes with single PE on each resource. 12 batches of 150 jobs were submitted with each requiring around 48 hours of computation.

Figure 5.77 gives comparison between adaptive checkpointing and periodic checkpointing for makespan parameter for all failure traces. 12 iterations refer to 12 batches of jobs submitted for each trace. As is clear from the figure adaptive checkpointing has better performance compared to periodic checkpointing for all failure traces.

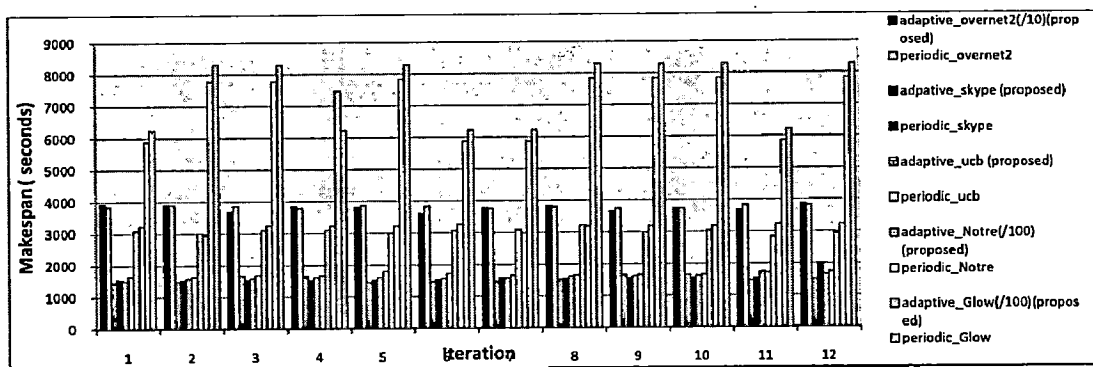


Figure 5.77 Comparison between adaptive checkpointing and periodic checkpointing for makespan parameter

Figure 5.78 compares adaptive checkpointing and periodic checkpointing for work lost due to failures. Adaptive checkpointing technique tends to lose more work due to adaptive increase in checkpoint interval which also results in more work lost for each failure.

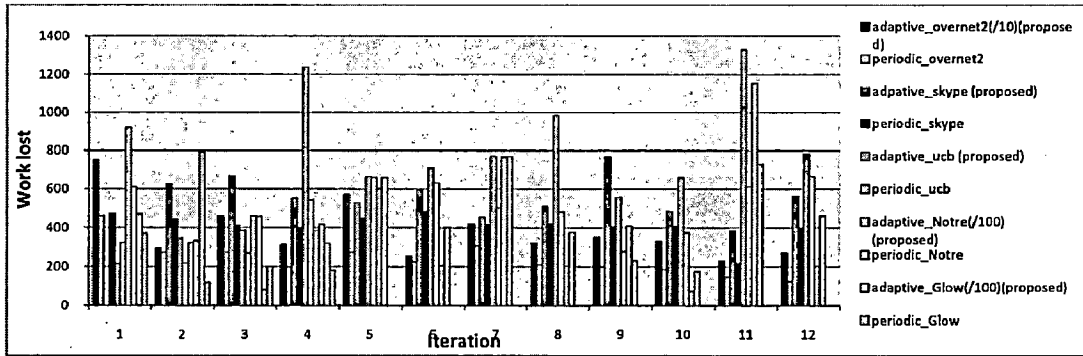


Figure 5.78 Comparison for work lost due to failures parameter

Figure 5.79 compares GA based adaptive checkpointing with GA based periodic checkpointing for flowtime parameter for all failure traces. Adaptive checkpointing technique has superior performance for this parameter for all failure traces.

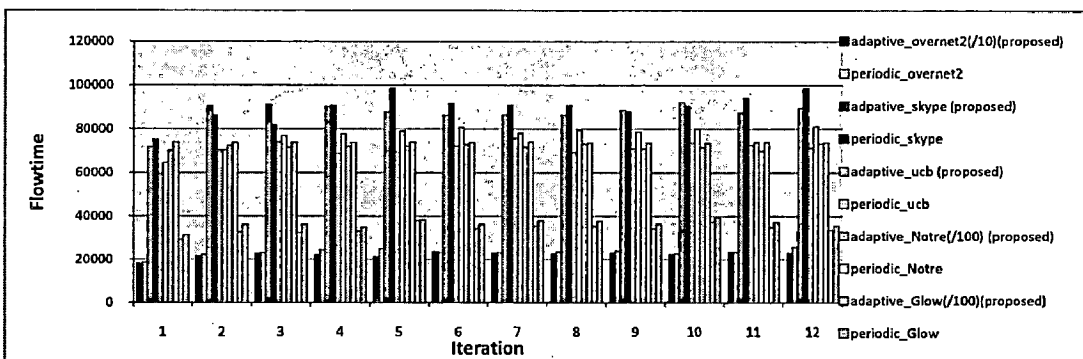


Figure 5.79 Comparison for flowtime parameter

Figure 5.80 compares the two techniques for number of checkpoints taken. Adaptive checkpointing technique takes fewer checkpoints compared to periodic checkpointing for all failure traces. The reason for fewer checkpoints is adaptive increase in checkpoint interval due to less failure rate of resources.

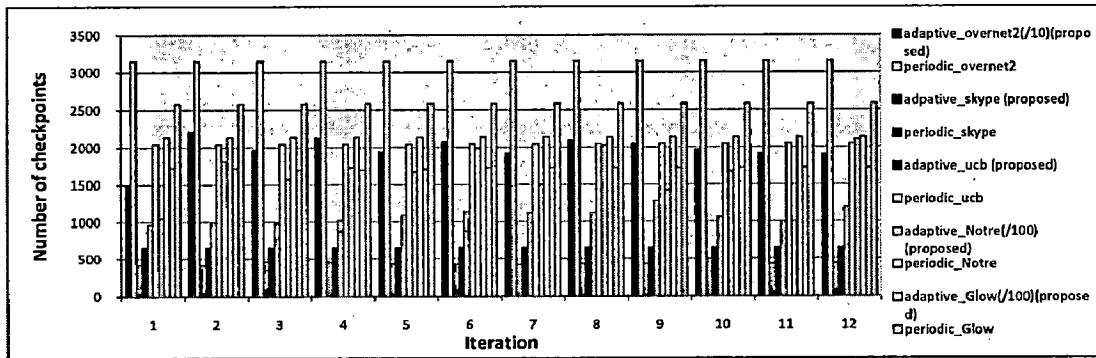


Figure 5.80 Comparison for number of checkpoints taken parameter

Figure 5.81 compares the two techniques for average bounded slowdown parameter. Adaptive checkpointing performs better for this parameter with lesser average bounded slowdown for each job.

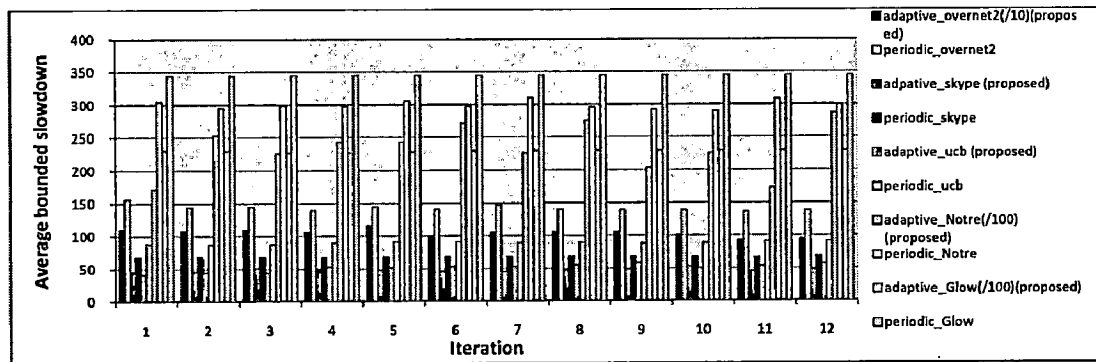


Figure 5.81 Comparison for average bounded slowdown parameter

Figure 5.82 compares the two techniques for utilization parameter. Adaptive checkpointing has slightly higher utilization of resources compared to periodic checkpointing.

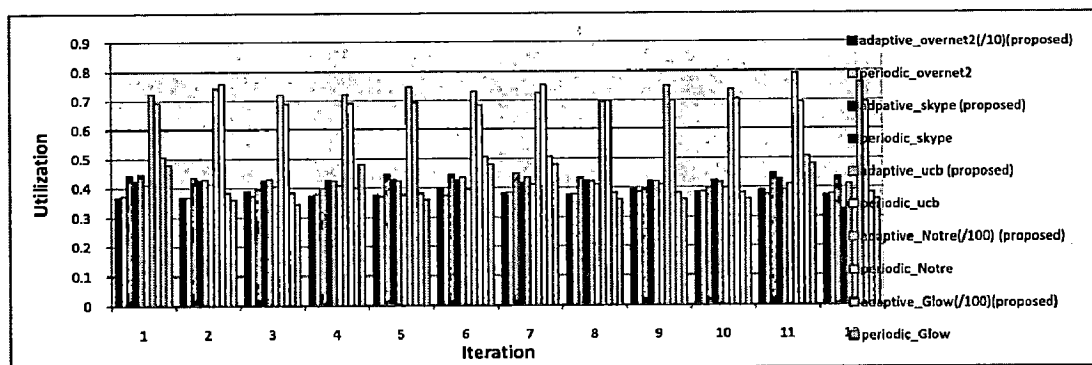


Figure 5.82 Comparison for utilization parameter

Figure 5.83 compares the two techniques for turnaround time parameter. Adaptive checkpointing has lesser turnaround time compared to periodic checkpointing.

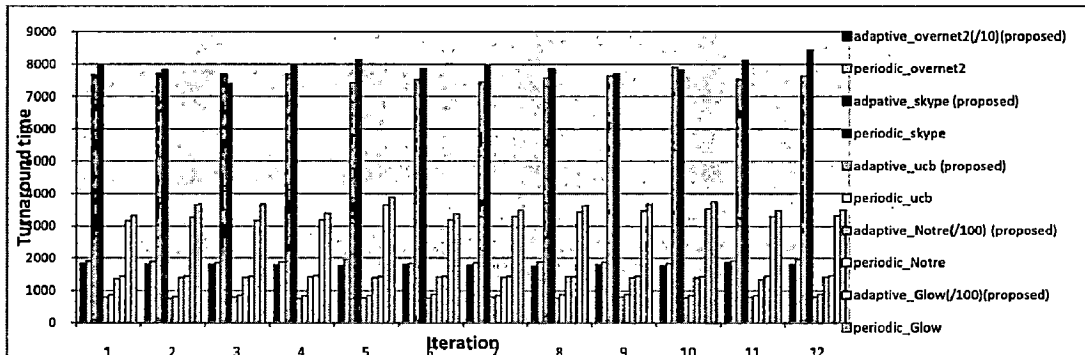


Figure 5.83 Comparison for turnaround time parameter

Figure 5.84 and 5.85 respectively gives overall comparison between GA based adaptive checkpointing and

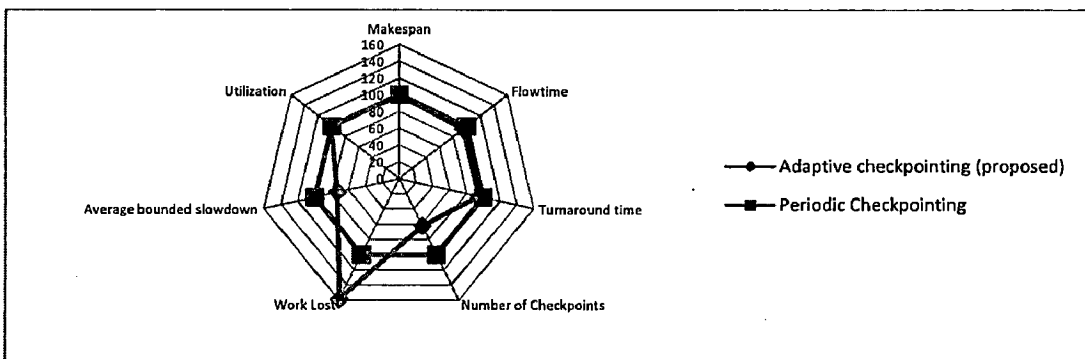


Figure 5.84 Overall comparison between GA based adaptive and periodic checkpointing (Overnet trace)

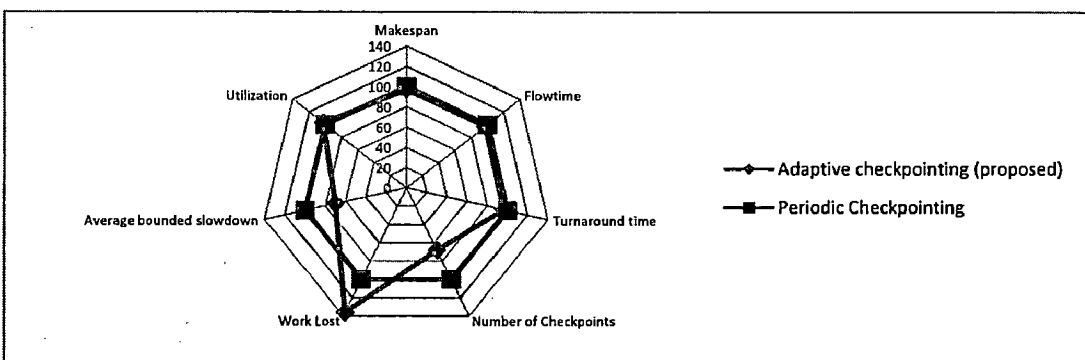


Figure 5.85 Overall comparison between GA based adaptive and periodic checkpointing (Skype trace)

GA based periodic checkpointing for trace 1 and trace 2. Values for adaptive checkpointing relative to periodic checkpointing for trace 1 are -1% for makespan, -5% for flowtime, -26.7% for average bounded slowdown, +.96% for utilization, +58.8% for work lost due to failures, -37.5% for number of checkpoints taken and -5% for average turnaround time. Values for adaptive checkpointing relative to periodic checkpointing for trace 2 are -2.6% for makespan, -2.7% for flowtime, -29.8% for average bounded slowdown, +2.4% for utilization, +36% for work lost due to failures, -32% for number of checkpoints taken and -3.88% for average turnaround time.

Figure 5.86 and 5.87, 5.88 respectively gives overall comparison between GA based adaptive checkpointing and GA based periodic checkpointing for trace 3, trace 4 and trace 5. Values for adaptive checkpointing relative to periodic checkpointing for trace 3 are -5% for makespan, -7.88% for flowtime, -42.7% for average bounded slowdown, +5.47% for utilization, +55.43% for work lost due to failures, -47% for number of checkpoints taken and -8.35% for average turnaround time. Values for adaptive checkpointing relative to periodic checkpointing for trace 4 are -4.4% for makespan, -2.7% for flowtime,

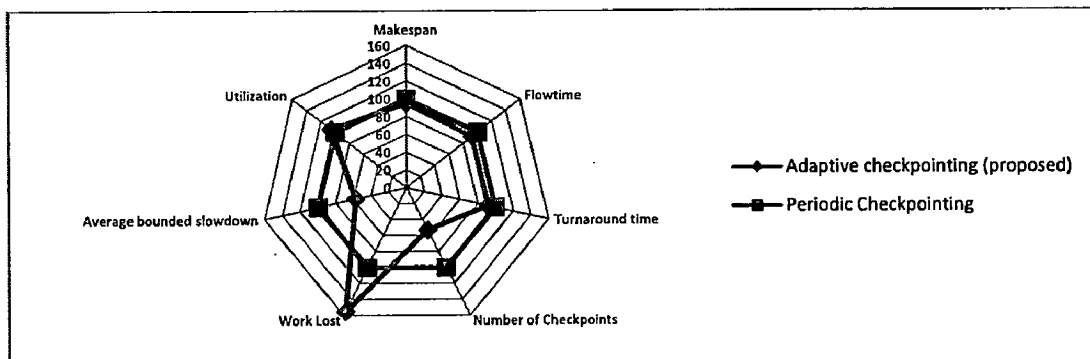


Figure 5.86 Overall comparison between GA based adaptive and periodic checkpointing (Ucb trace)

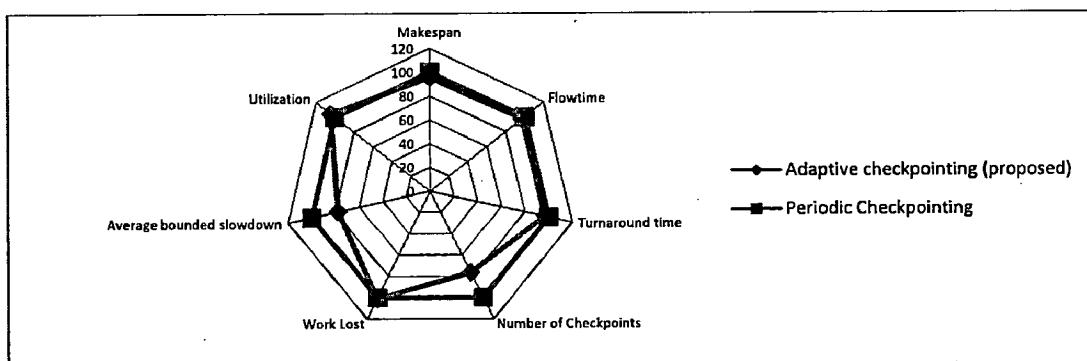


Figure 5.87 Overall comparison between GA based adaptive and periodic checkpointing (Notre trace)

-22% for average bounded slowdown, +4.69% for utilization, +1.4% for work lost due to failures, -23.5% for number of checkpoints taken and -3.2% for average turnaround time. Values for adaptive checkpointing relative to periodic checkpointing for trace 5 are -3.8% for makespan, -5.7% for flowtime, -33.43% for average bounded slowdown, +3.92% for utilization, +91% for work lost due to failures, -32% for number of checkpoints taken and -6.7% for average turnaround time.

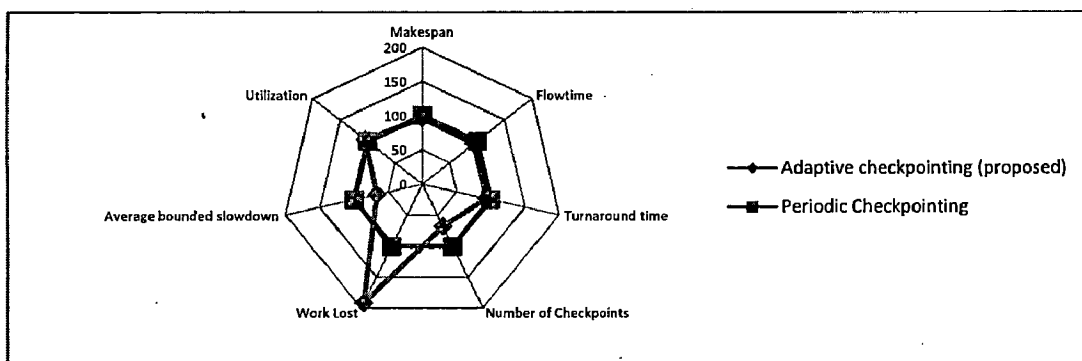


Figure 5.88 Overall comparison between GA based adaptive and periodic checkpointing (Glow trace)

### 5.12 Performance Comparison of ACO based checkpointing techniques using failure traces

This section compares ACO based adaptive checkpointing based on fault ratios of resources with ACO based periodic checkpointing. Failure traces with configurations as given in section 5.11 are taken for experimentation.

Figure 5.89 gives comparison between adaptive checkpointing and periodic checkpointing for makespan parameter for all failure traces. As is clear from the figure adaptive checkpointing has better performance (lesser makespan values) compared to periodic checkpointing for all failure traces.

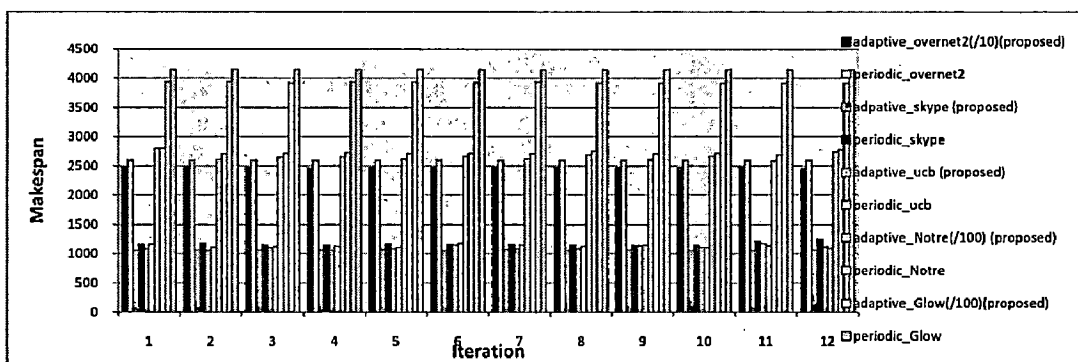


Figure 5.89 Comparison between ACO based adaptive checkpointing and ACO based periodic checkpointing for makespan parameter

Figure 5.90 compares adaptive checkpointing and periodic checkpointing for work lost due to failures. Adaptive checkpointing technique tends to lose more work due to adaptive increase in checkpoint interval which also results in more work lost for each failure.

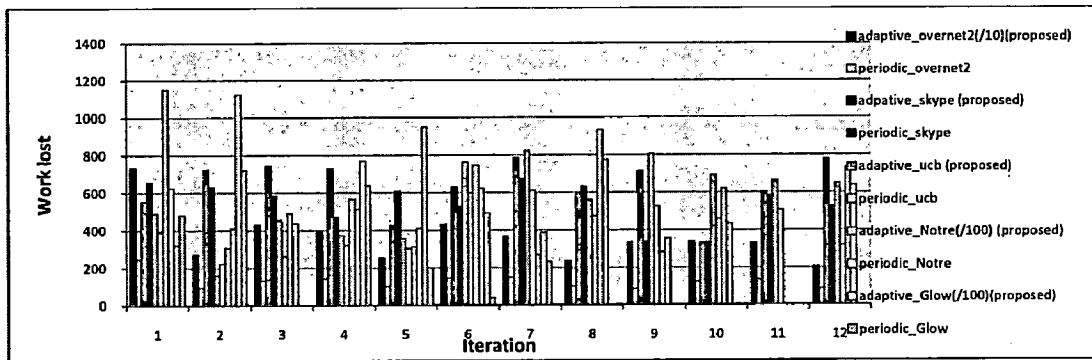


Figure 5.90 Comparison for work lost due to failures parameter

Figure 5.91 compares ACO based adaptive checkpointing with ACO based periodic checkpointing for flowtime parameter for all failure traces. Adaptive checkpointing technique has superior performance (lower flowtime) for this parameter for all failure traces.

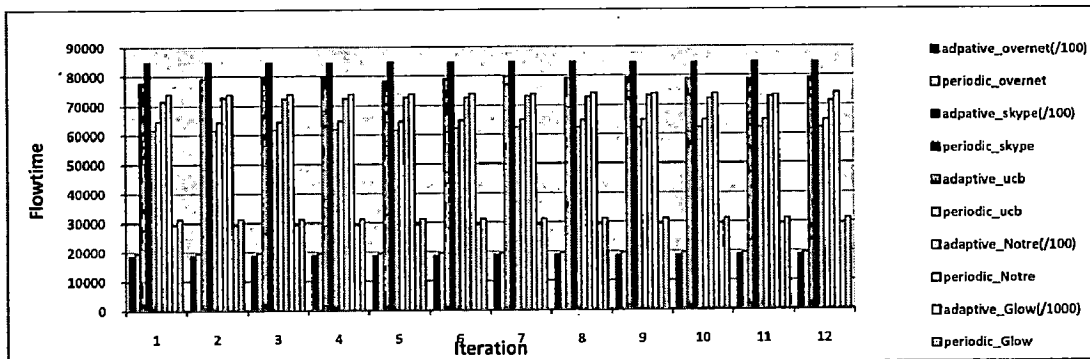


Figure 5.91 Comparison for flowtime parameter

Figure 5.92 compares the two techniques for number of checkpoints taken. Adaptive checkpointing technique takes fewer checkpoints compared to periodic checkpointing for all failure traces. The reason for fewer checkpoints is adaptive increase in checkpoint interval due to less failure rate of resources.

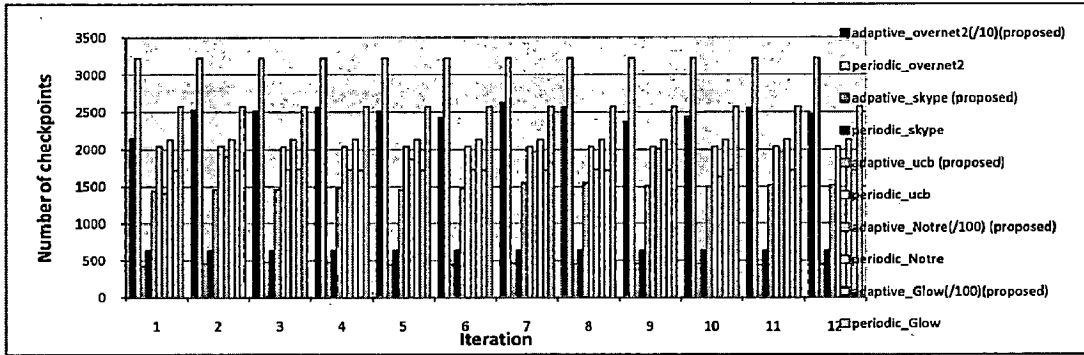


Figure 5.92 Comparison for number of checkpoints taken parameter

Figure 5.93 compares the two techniques for average bounded slowdown parameter. Adaptive checkpointing performs better for this parameter with lesser average bounded slowdown.

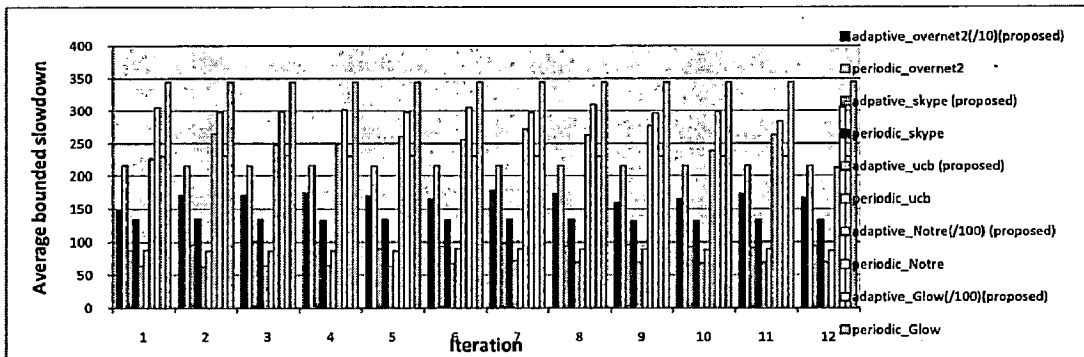


Figure 5.93 Comparison for average bounded slowdown parameter

Figure 5.94 compares the two techniques for utilization parameter. Adaptive checkpointing has slightly higher utilization of resources compared to periodic checkpointing.

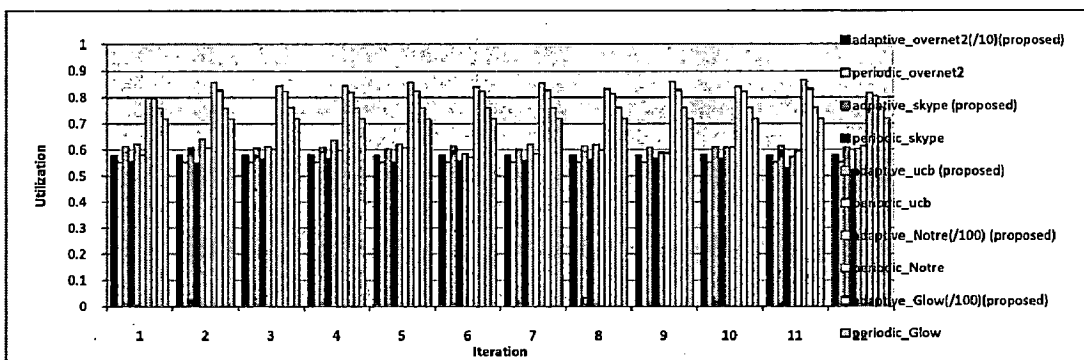


Figure 5.94 Comparison for utilization parameter



Figure 5.95 compares the two techniques for turnaround time parameter. Adaptive checkpointing has lesser turnaround time compared to periodic checkpointing.

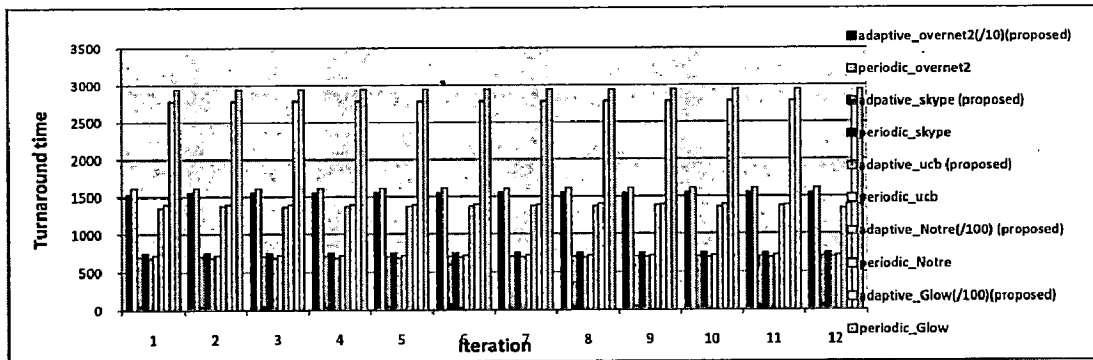


Figure 5.95 Comparison for turnaround time parameter

Figure 5.96 and 5.97 respectively gives overall comparison between ACO based adaptive checkpointing

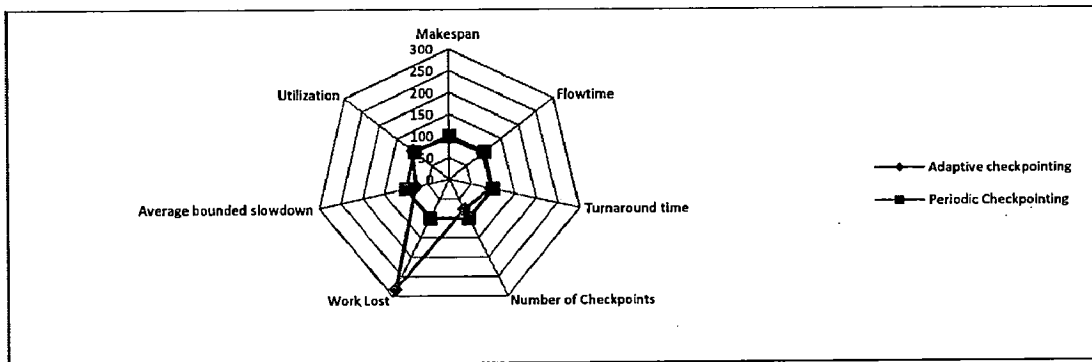


Figure 5.96 Overall comparison between ACO based adaptive and periodic checkpointing (Overnet trace)

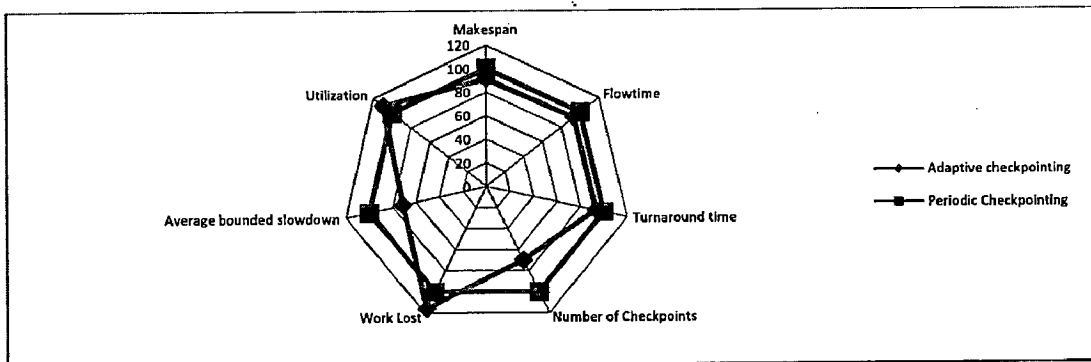


Figure 5.97 Overall comparison between ACO based adaptive and periodic checkpointing (Skype trace)

and ACO based periodic checkpointing for trace 1 and trace 2. Values for adaptive checkpointing relative to periodic checkpointing for trace 1 are -4.8% for makespan, -3.7% for flowtime, -22.3% for average bounded slowdown, +5% for utilization, +182% for work lost due to failures, -23% for number of checkpoints taken and -4.1% for average turnaround time. Values for adaptive checkpointing relative to periodic checkpointing for trace 2 are -2.5% for makespan, -2% for flowtime, -15.7% for average bounded slowdown, +2.4% for utilization, +14% for work lost due to failures, -17.5% for number of checkpoints taken and -1.86% for average turnaround time.

Figure 5.98 and 5.99, 5.100 respectively gives overall comparison between ACO based adaptive checkpointing and ACO based periodic checkpointing for trace 3, trace 4 and trace 5. Values for adaptive checkpointing relative to periodic checkpointing for trace 3 are -2.3% for makespan, -4% for flowtime, -25% for average bounded slowdown, +2.34% for utilization, +35.5% for work lost due to failures, -26.7% for number of checkpoints taken and -4% for average turnaround time. Values for adaptive checkpointing relative to periodic checkpointing for trace 4 are -2.5% for makespan, -2% for

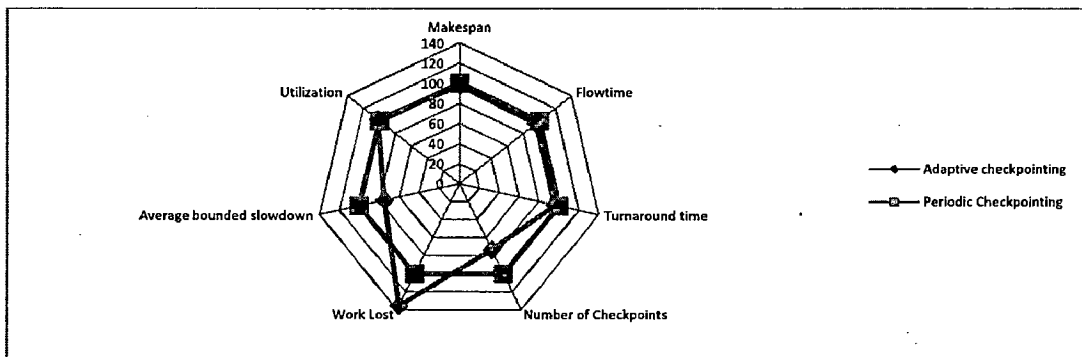


Figure 5.98 Overall comparison between ACO based adaptive and periodic checkpointing (Ucb trace)

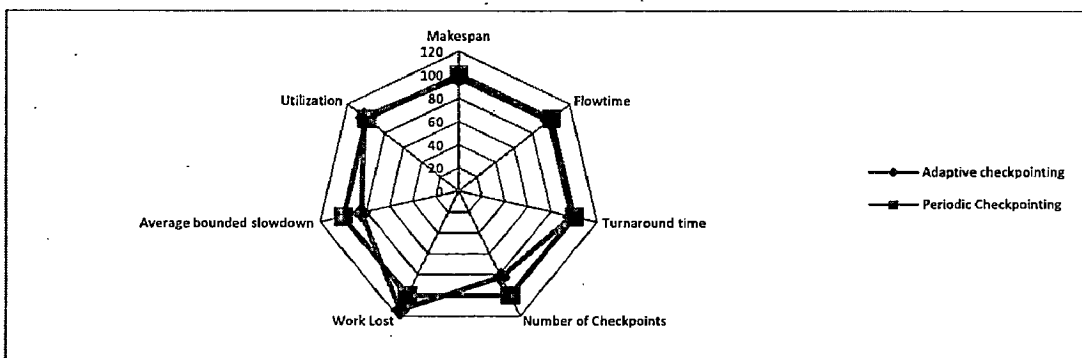


Figure 5.99 Overall comparison between ACO based adaptive and periodic checkpointing (Notre trace)

flowtime, -15.7% for average bounded slowdown, +2.4% for utilization, +14.13% for work lost due to failures, -17.5% for number of checkpoints taken and -1.8% for average turnaround time. Values for adaptive checkpointing relative to periodic checkpointing for trace 5 are -5.4% for makespan, -5.49% for flowtime, -33.14% for average bounded slowdown, +5.7% for utilization, +86.7% for work lost due to failures, -33.14% for number of checkpoints taken and -5.5% for average turnaround time.

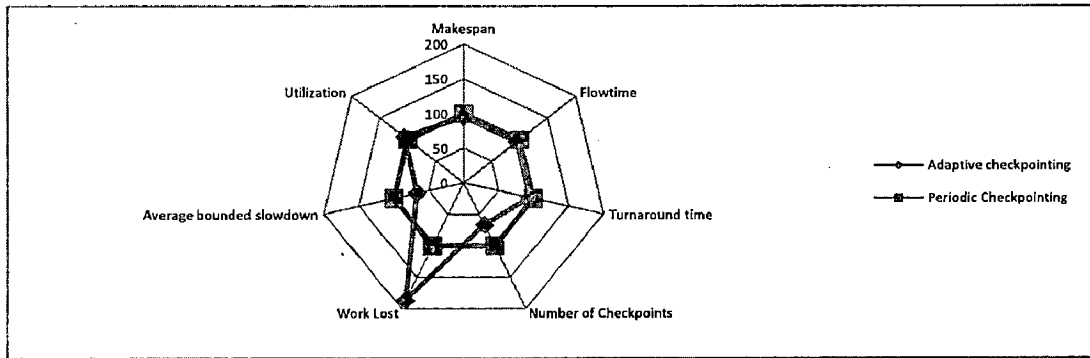


Figure 5.100 Overall comparison between ACO based adaptive and periodic checkpointing (Glow trace)

### 5.13 GA-based checkpointing techniques performance comparison using workload traces

This section compares GA-based adaptive checkpointing using fault ratios of resources with GA-based periodic checkpointing for workload traces from workload trace archives [51, 52]. Alea [41] simulator is used for simulation.

#### a) Trace 1(HPC2)

This trace consisted of 3000 jobs with each job having different number of PE requirement ranging from 1 PE to 128 PEs. Four clusters each with 4 SMPs with each SMP having 16 processors were used for execution of jobs. Jobs are allocated processors only from one cluster. Note that there are a maximum of 64 processors in any cluster and job requirement can be higher than that. In that case resources are not allocated to job. Resources failures were simulated with first and third cluster having much higher failure rate compared to second and fourth. Coordinated checkpointing is used for performing checkpointing operation. Complete job is restarted from last successfully saved checkpoint even if only one PE allocated to it fails.

Figure 5.101 shows the average cluster usage per day for each cluster for adaptive checkpointing technique. As can be seen utilization of resources is very low.

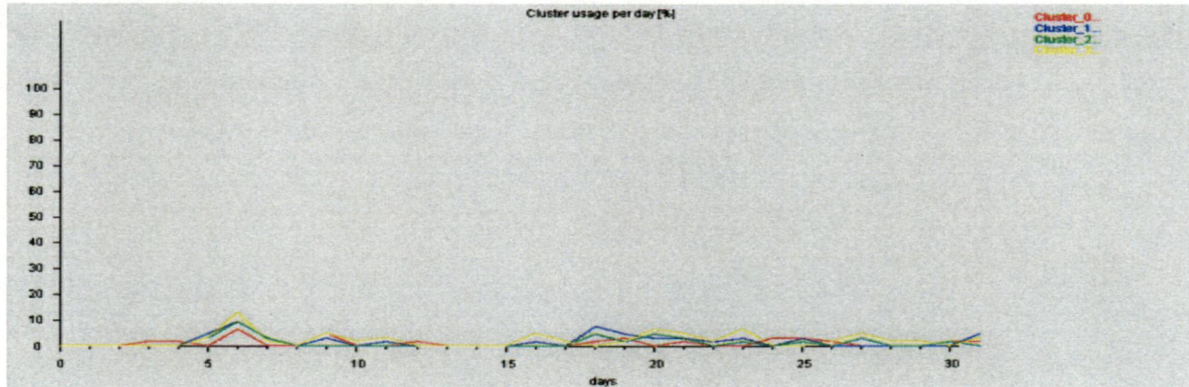


Figure 5.101 Cluster usage per day for workload trace 1 (Adaptive checkpointing)

Figure 5.102 shows number of requested and used CPUs per day for GA-based adaptive checkpointing technique. Requested CPUs are much less than the available CPUs

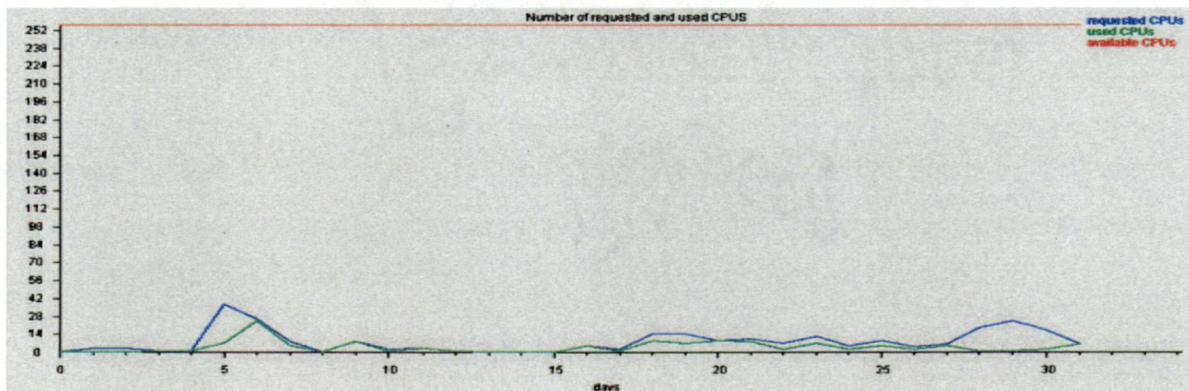


Figure 5.102 Number of requested and used CPUs per day for trace 1 (Adaptive checkpointing)

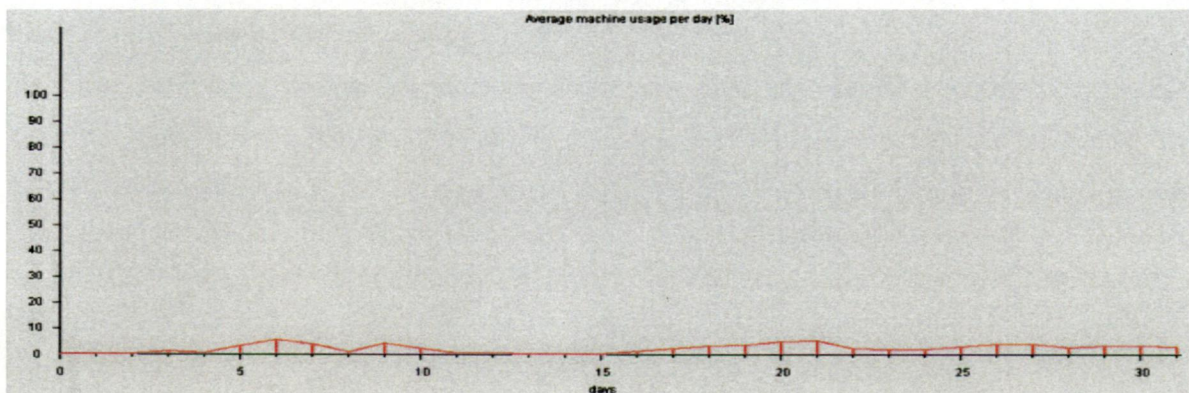


Figure 5.103 Average machine usage per day for trace 1 (Adaptive checkpointing)



Figure 5.103 gives average machine usage per day for adaptive checkpointing technique. Figure 5.104 gives cluster usage per day for periodic checkpointing technique. Cluster usage is very low.

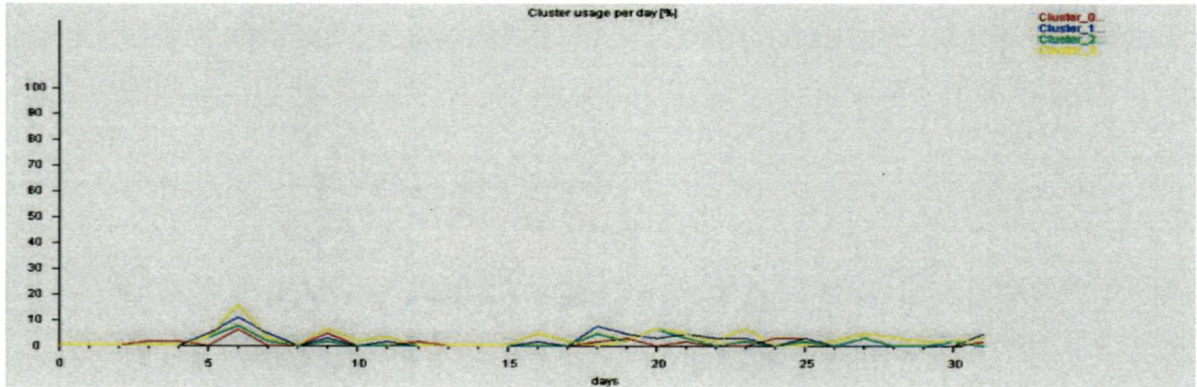


Figure 5.104 Cluster usage per day for workload trace 1 (Periodic checkpointing)

Figure 5.105 gives number of requested and used CPUs per day for periodic checkpointing technique. Requested CPUs is much lower than that available.

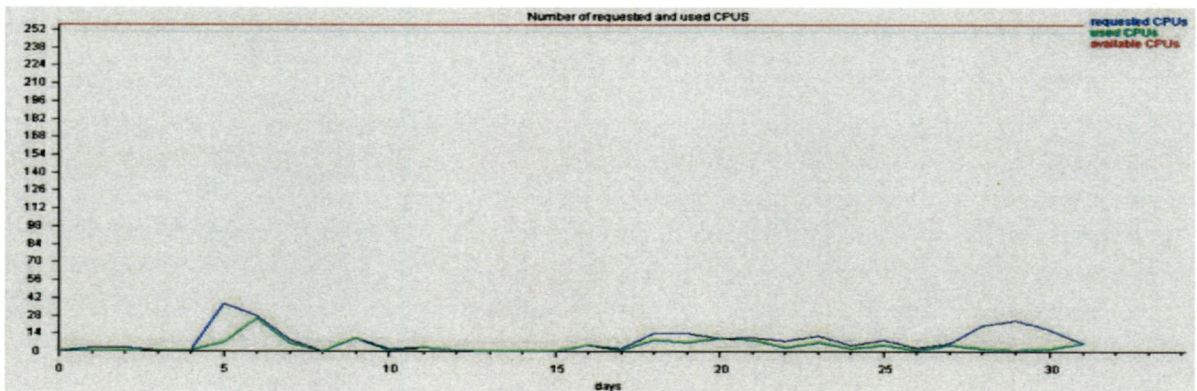


Figure 5.105 Number of requested and used CPUs per day for trace 1 (Periodic checkpointing)

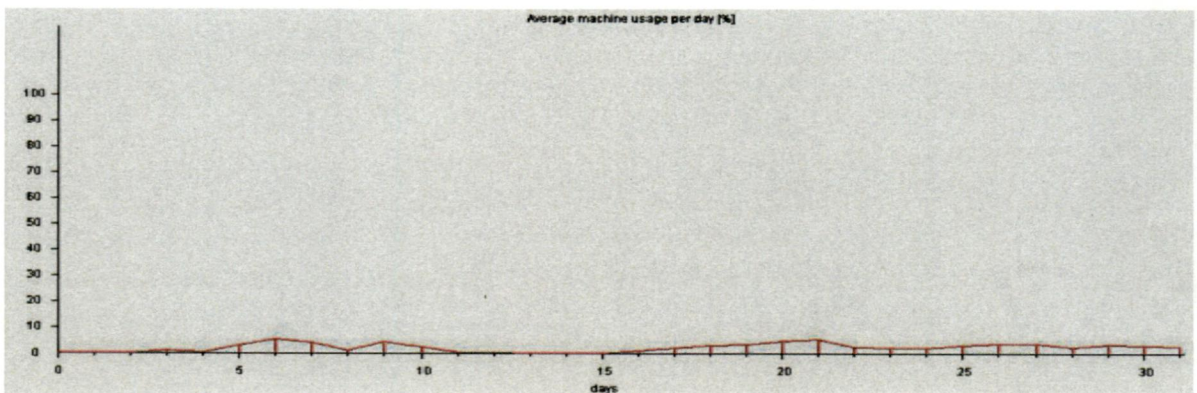


Figure 5.106 Average machine usage per day for trace 1 (Periodic checkpointing)



Figure 5.106 gives average machine usage per day for GA-based periodic checkpointing. Figure 5.107 gives average waiting time of jobs submitted to a cluster for each day for each of periodic checkpointing and adaptive checkpointing technique.

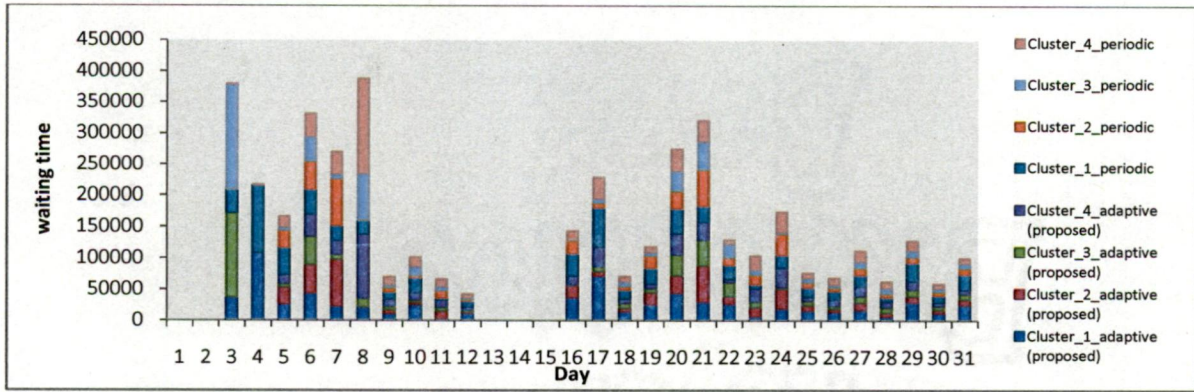


Figure 5.107 Average waiting time per cluster per day for trace 1

Figure 5.108 gives average response time of jobs submitted to a cluster for each day for both techniques.

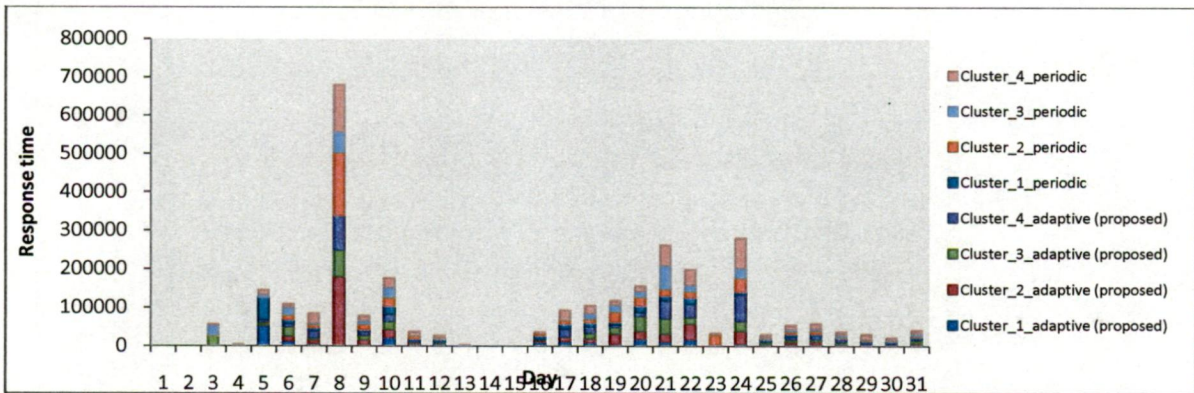


Figure 5.108 Average response time per cluster per day for trace 1

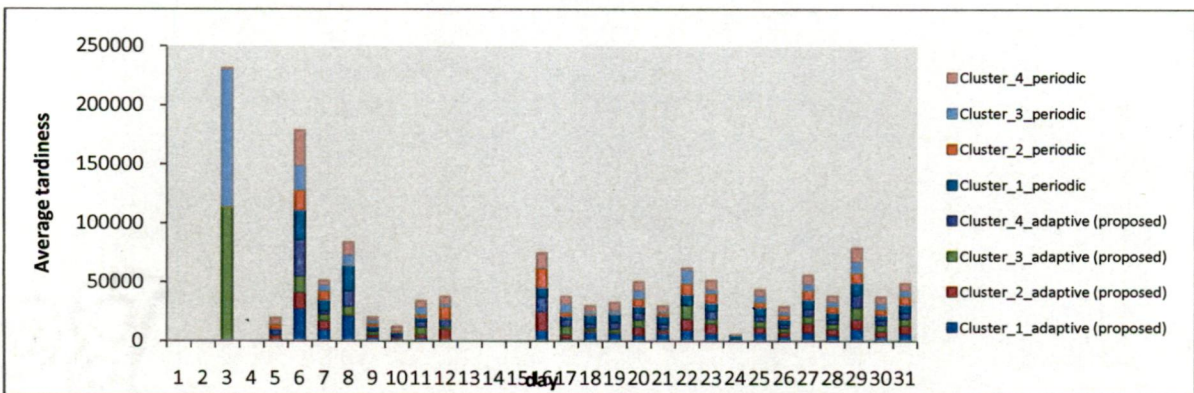


Figure 5.109 Average tardiness time per cluster per day for trace 1

Figure 5.109 gives average tardiness time of jobs submitted to a cluster for each day for each of periodic checkpointing and adaptive checkpointing technique. Figure 5.110 gives number of checkpoints performed by jobs on each cluster for each day for each of periodic checkpointing and adaptive checkpointing technique.

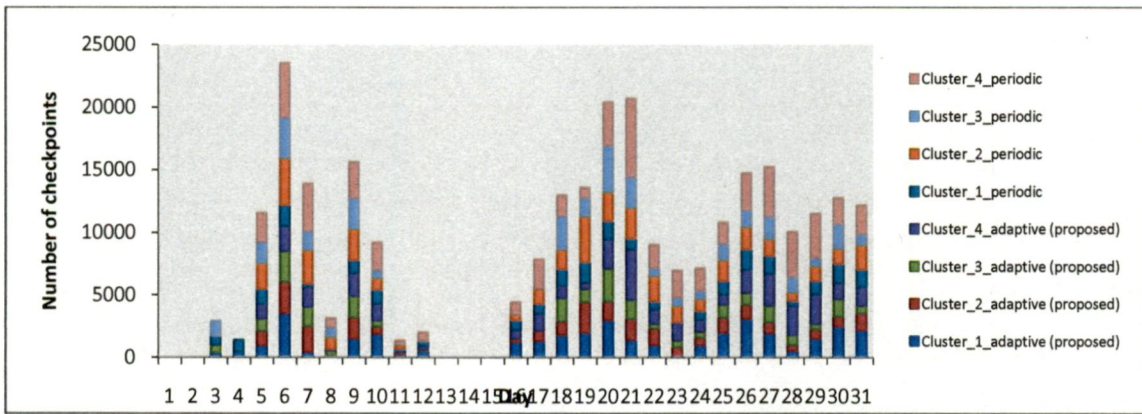


Figure 5.110 Number of checkpoints per cluster per day for trace 1

Figure 5.111 gives work lost due to failures on each cluster for each day for each of periodic checkpointing and adaptive checkpointing technique.

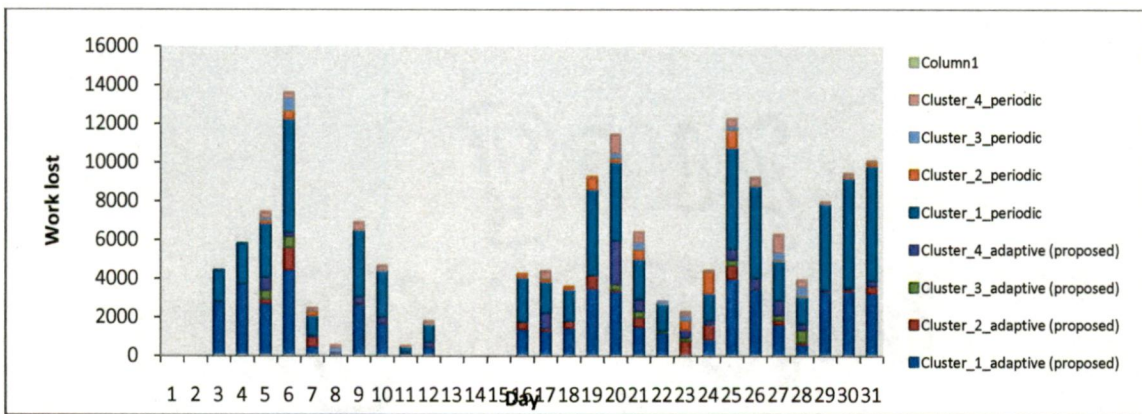


Figure 5.111 Work lost due to failures per cluster per day for trace 1

Figure 5.112 gives overall comparison between GA-based adaptive checkpointing using fault ratios of resources and GA-based periodic checkpointing. Values of adaptive checkpointing technique relative to periodic checkpointing are 0% for makespan, 0% for flowtime, -20.5% for average bounded slowdown, -11.7% for work lost due to failures, -21% for number of checkpoints taken and -4.3% for average turnaround time



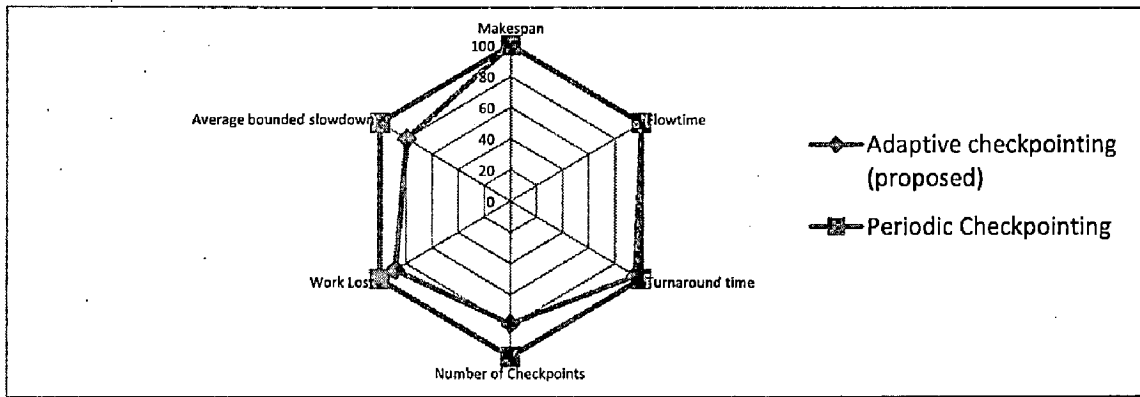


Figure 5.112 Overall comparison between adaptive and periodic checkpointing for workload trace 1

### b) Trace 2 (LCG)

This trace consisted of 3000 jobs with each job having only 1PE requirement. Six clusters each with 2 SMPs with each SMP having 16 processors were used for execution of jobs. Jobs are allocated processors only from one cluster. Resources failures were simulated with first and third cluster having much higher failure rate. Coordinated checkpointing is used for performing checkpointing operation. Complete job is restarted from last successfully saved checkpoint even if only one PE allocated to it fails.

Figure 5.113 shows the average machine usage per hour for adaptive checkpointing technique.

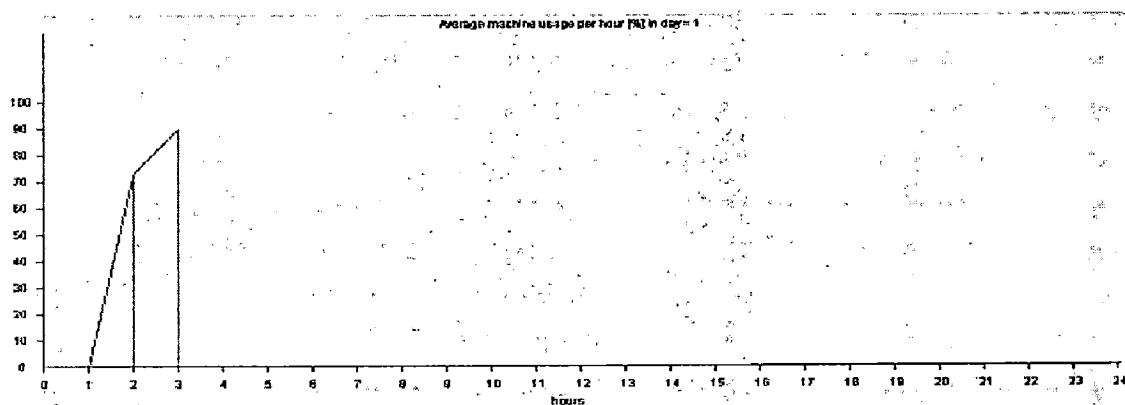


Figure 5.113 Average machine usage per hour for trace 2 (Adaptive checkpointing)



Figure 5.114 gives cluster usage %age for each cluster for adaptive checkpointing technique.

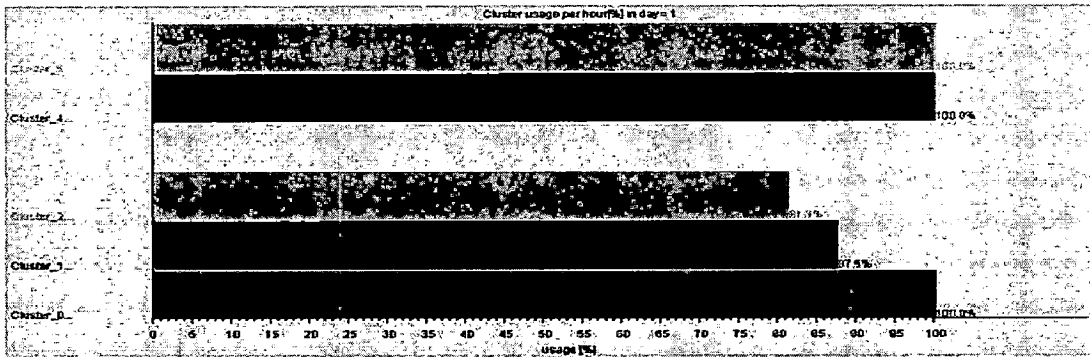


Figure 5.114 Cluster usage per hour for trace 2 (Adaptive checkpointing)

Figure 5.115 shows the average machine usage per hour for periodic checkpointing technique.

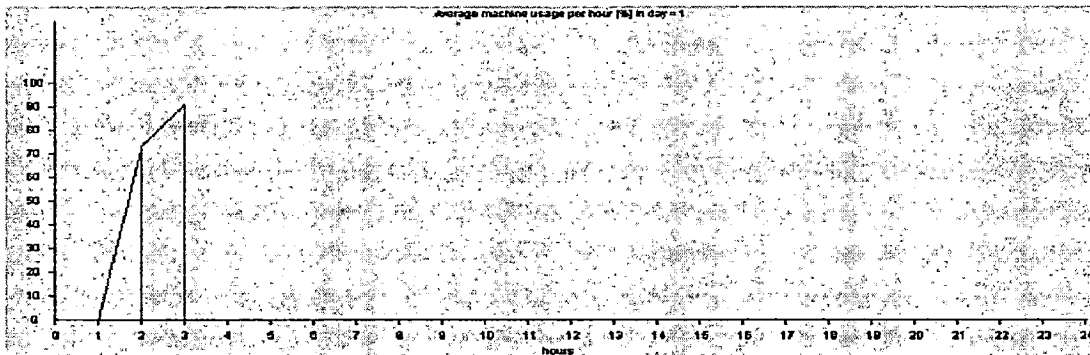


Figure 5.115 Average machine usage per hour for trace 2 (Periodic checkpointing)

Figure 5.116 gives cluster usage %age for each cluster for periodic checkpointing technique.

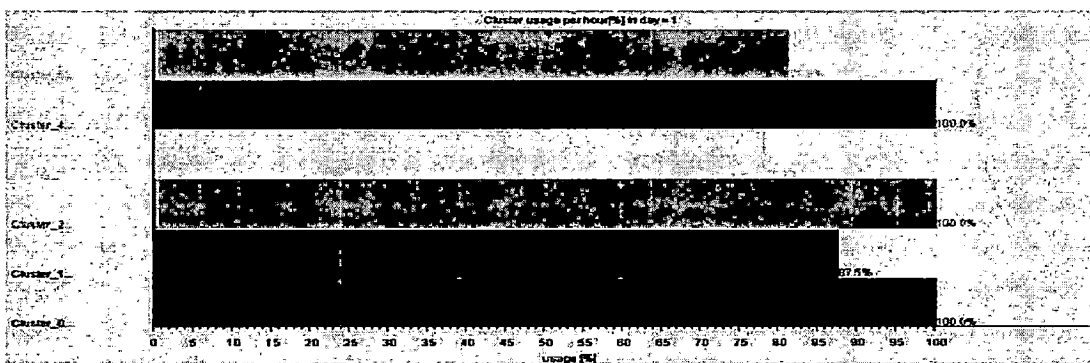


Figure 5.116 Cluster usage per hour for trace 2 (Periodic checkpointing)

Figure 5.117 gives overall comparison between GA-based adaptive checkpointing using fault ratios of resources and GA-based periodic checkpointing. Values of adaptive checkpointing technique relative to periodic checkpointing are 0% for makespan, 0% for flowtime, -15% for average bounded slowdown, 31% for work lost due to failures, -40.5% for number of checkpoints taken and -1.3% for average turnaround time.

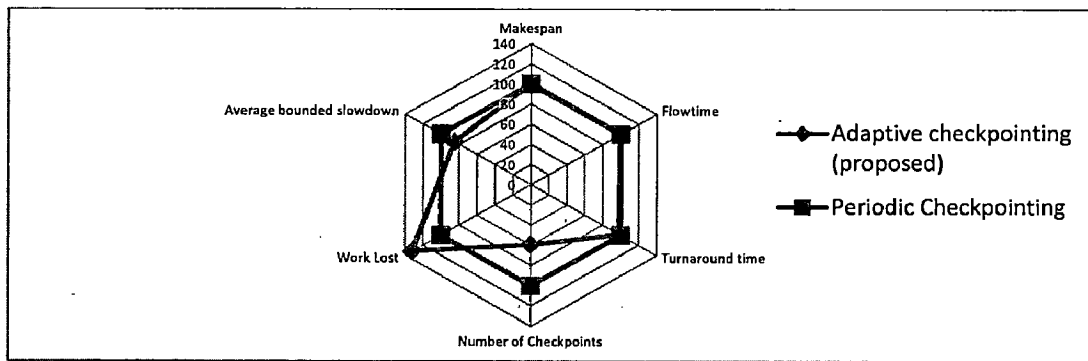


Figure 5.117 Overall comparison between adaptive and periodic checkpointing for workload trace 2

### c) Trace 3 (NASA-iPSC-1993)

This trace consisted of 10000 jobs with each job having different number of PE requirement ranging from 1 PE to 128 PEs. Three clusters each with 8 SMPs with each SMP having 16 processors were used for execution of jobs. Jobs are allocated processors only from one cluster. Resources failures were simulated with first cluster having much higher failure rate compared to second and third. Coordinated checkpointing is used for performing checkpointing operation. Complete job is restarted from last successfully saved checkpoint even if only one PE allocated to it fails.

Figure 5.118 shows the average cluster usage per day for each cluster for adaptive checkpointing technique. As can be seen utilization of resources is very low.

Figure 5.119 shows number of requested and used CPUs per day for GA-based adaptive checkpointing technique. Requested CPUs are much less than the available CPUs.

Figure 5.120 gives average machine usage per day for adaptive checkpointing technique.

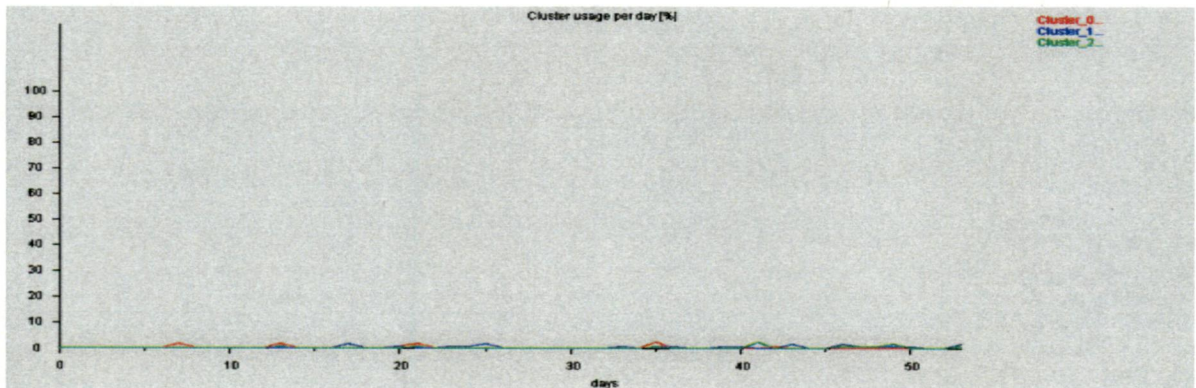


Figure 5.118 Cluster usage per day for workload trace 3 (Adaptive checkpointing)

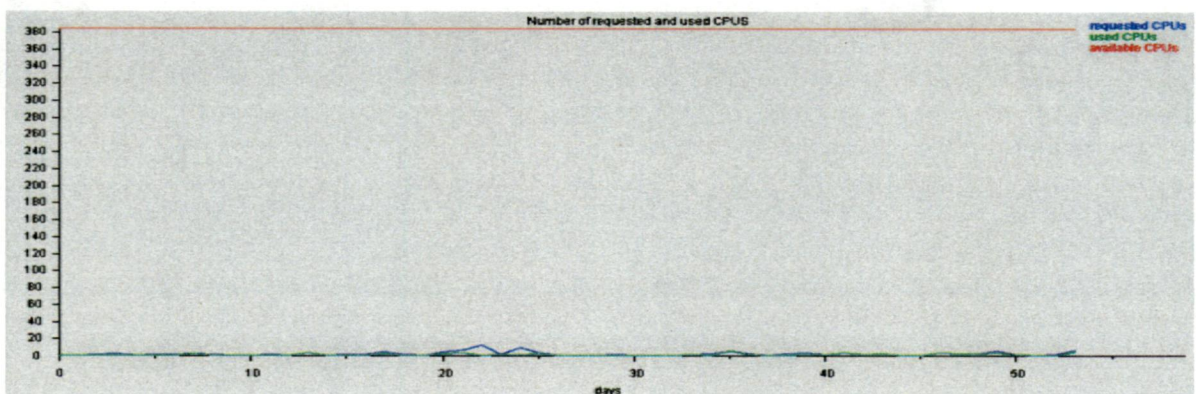


Figure 5.119 Number of requested and used CPUs per day for trace 3 (Adaptive checkpointing)

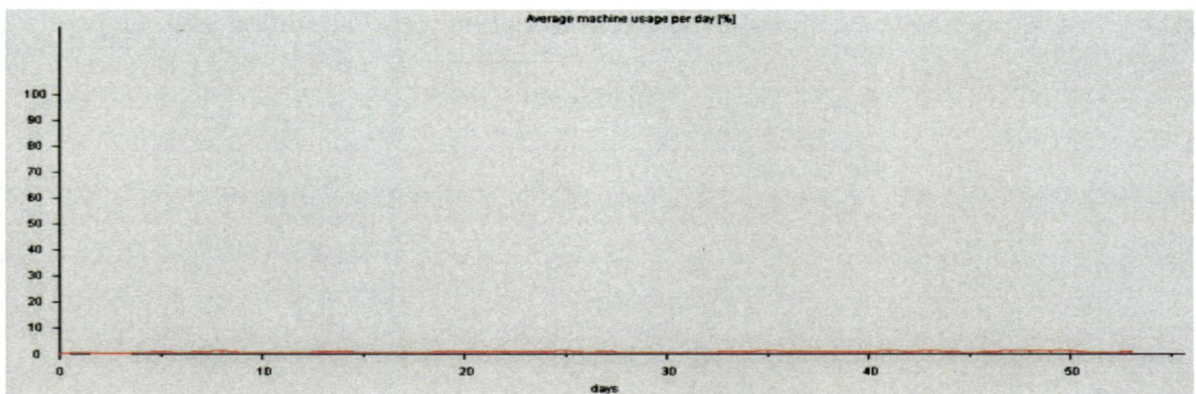


Figure 5.120 Average machine usage per day for trace 3 (Adaptive checkpointing)

Figure 5.121 shows the average cluster usage per day for each cluster for periodic checkpointing technique.

Figure 5.122 shows number of requested and used CPUs per day for GA-based periodic checkpointing technique. Requested CPUs are much less than the available CPUs.



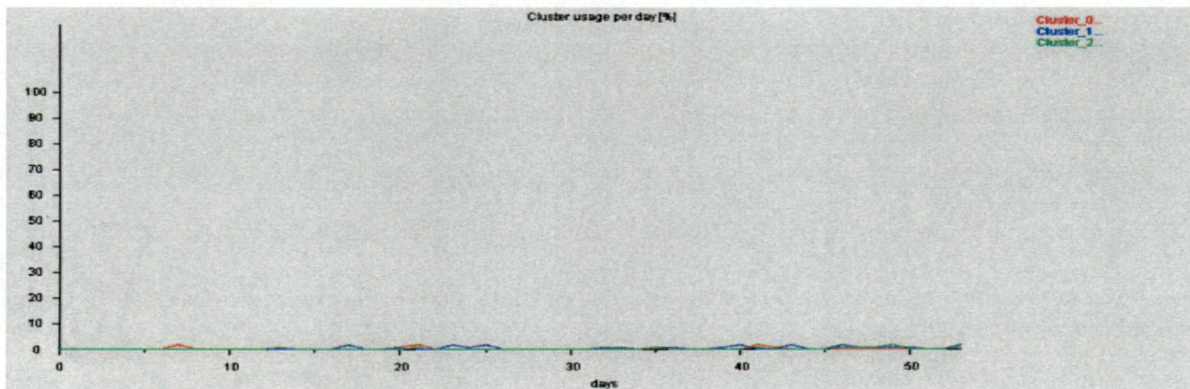


Figure 5.121 Cluster usage per day for workload trace 3 (Periodic checkpointing)

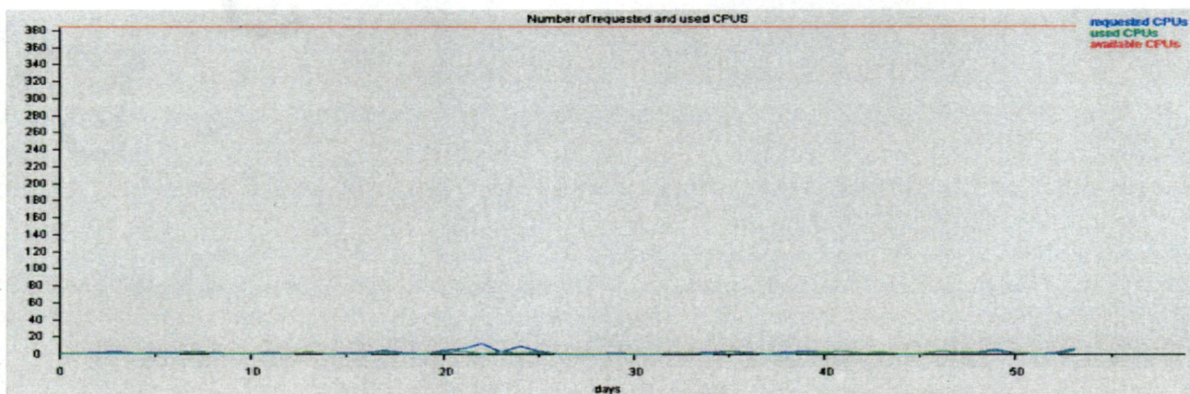


Figure 5.122 Number of requested and used CPUs per day for trace 3 (Periodic checkpointing)

Figure 5.123 gives average machine usage per day for periodic checkpointing technique.

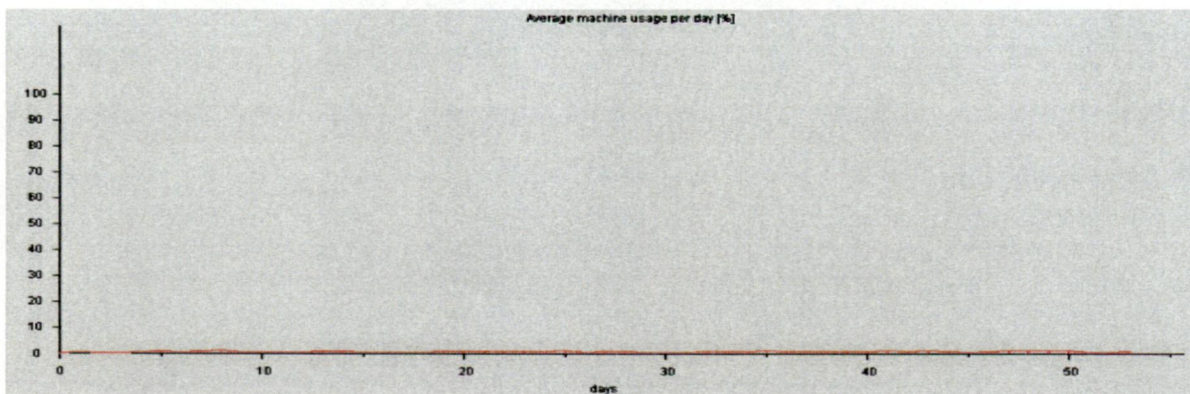


Figure 5.123 Average machine usage per day for trace 3 (Periodic checkpointing)

Figure 5.124 gives average waiting time of jobs submitted to a cluster for each day for each of periodic checkpointing and adaptive checkpointing technique.



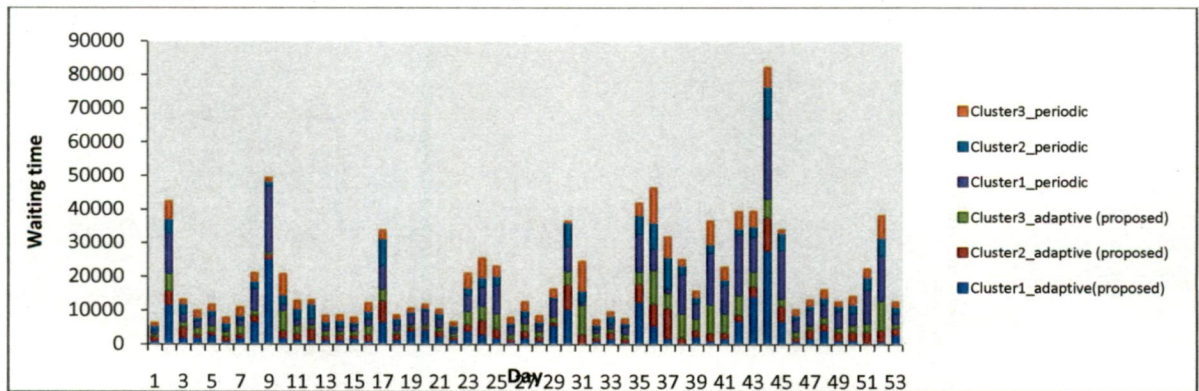


Figure 5.124 Average waiting time per cluster per day for trace 3

Figure 5.125 gives average response time of jobs submitted to a cluster for each day for each of periodic checkpointing and adaptive checkpointing technique.

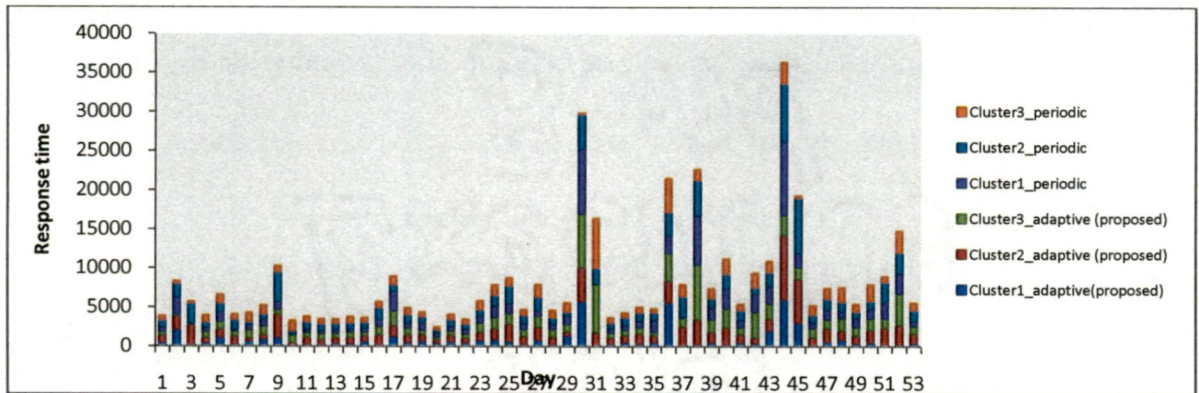


Figure 5.125 Average response time per cluster per day for trace 3

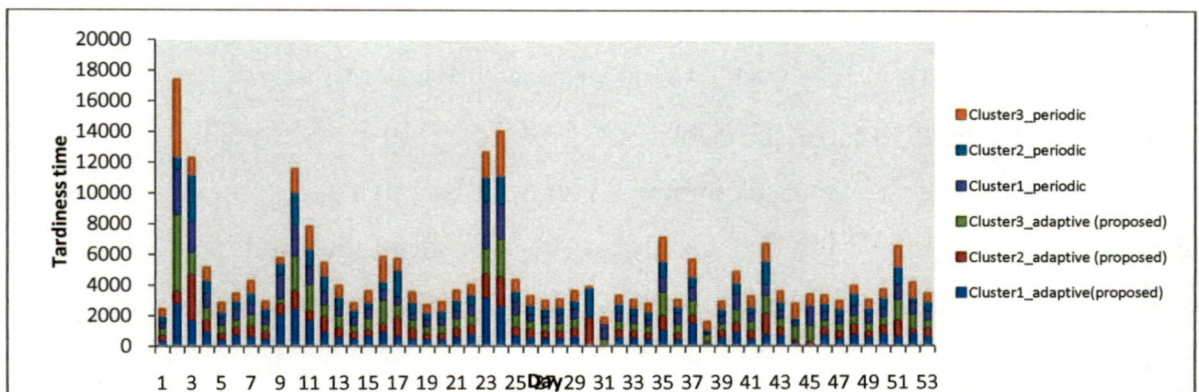


Figure 5.126 Average tardiness time per cluster per day for trace 3

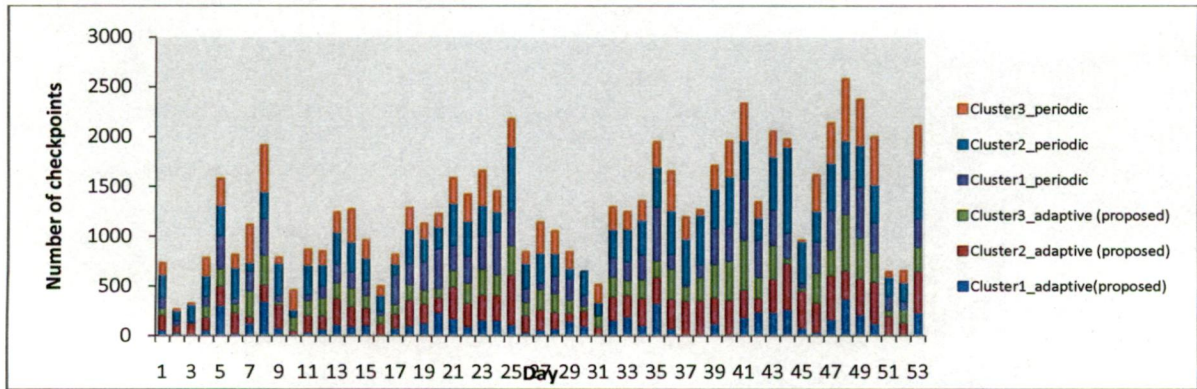


Figure 5.127 Number of checkpoints per cluster per day for trace3

Figure 5.126 gives average tardiness time of jobs submitted to a cluster for each day for each technique. Figure 5.127 gives number of checkpoints performed by jobs on each cluster for each day for each of periodic checkpointing and adaptive checkpointing technique. Figure 5.128 gives work lost due to failures on each cluster for each day for each of periodic checkpointing and adaptive checkpointing technique.

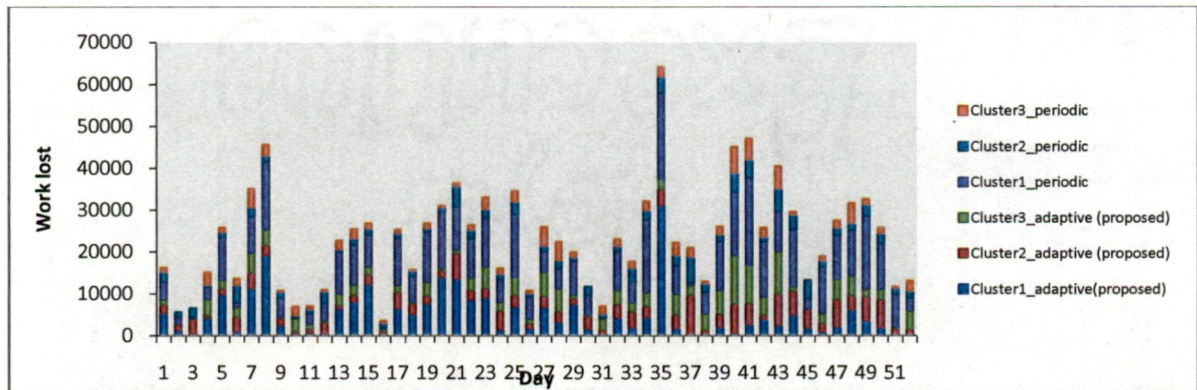


Figure 5.128 Work lost due to failures per cluster per day for trace 3

Figure 5.129 gives overall comparison between GA-based adaptive checkpointing using fault ratios of resources and GA-based periodic checkpointing. Values of adaptive checkpointing technique relative to periodic checkpointing are 0% for makespan, 0% for flowtime, -19.76% for average bounded slowdown, -13% for work lost due to failures, -29% for number of checkpoints taken and -4.85% for average turnaround time.



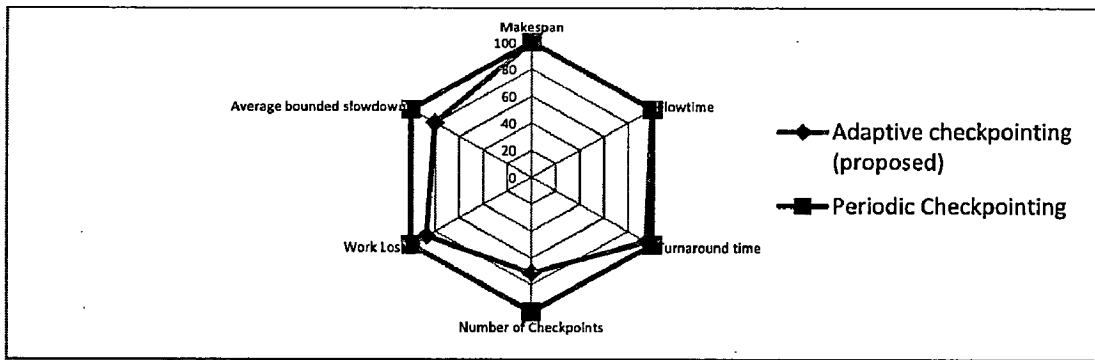


Figure 5.129 Overall comparison between adaptive and periodic checkpointing for workload trace 3

**d) Trace 4 (LCG-2005)**

This trace consisted of 10000 jobs with each job having requirement of 1PE only. Six clusters each with 4 SMPs with each SMP having 16 processors were used for execution of jobs. Jobs are allocated processors only from one cluster. Resources failures were simulated with first cluster having much higher failure rate. Coordinated checkpointing is used for performing checkpointing operation. Complete job is restarted from last successfully saved checkpoint even if only one PE allocated to it fails.

Figure 5.130 shows the average machine usage per hour for adaptive checkpointing technique.

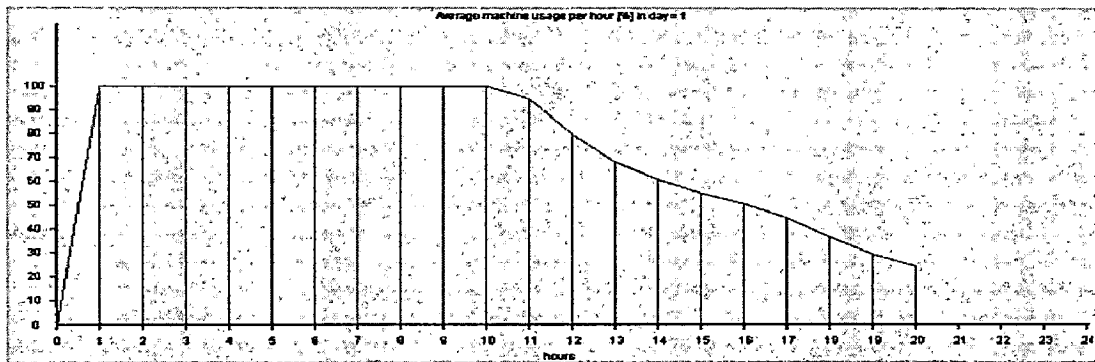


Figure 5.130 Average machine usage per hour for workload trace 4 (Adaptive checkpointing)

Figure 5.131 gives cluster usage %age for each cluster for adaptive checkpointing technique.

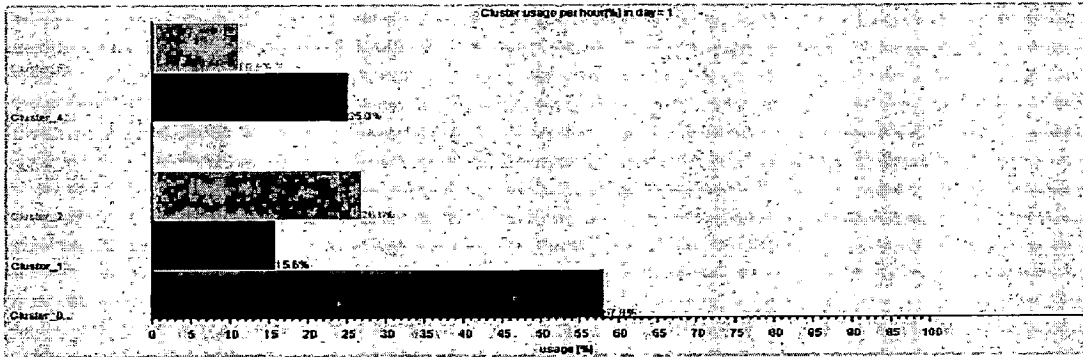


Figure 5.131 Average cluster usage % for workload trace 4 (Adaptive checkpointing)

Figure 5.132 shows the average machine usage per hour for periodic checkpointing technique.

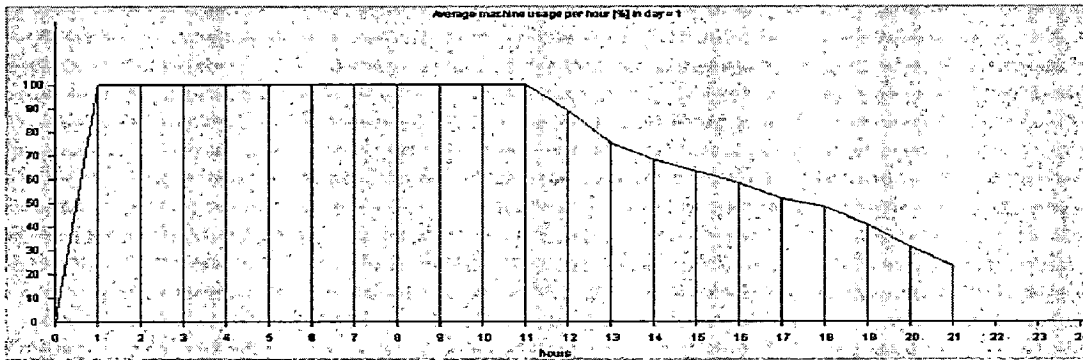


Figure 5.132 Average machine usage per hour for workload trace 4 (Periodic checkpointing)

Figure 5.133 gives cluster usage %age for each cluster for periodic checkpointing technique.

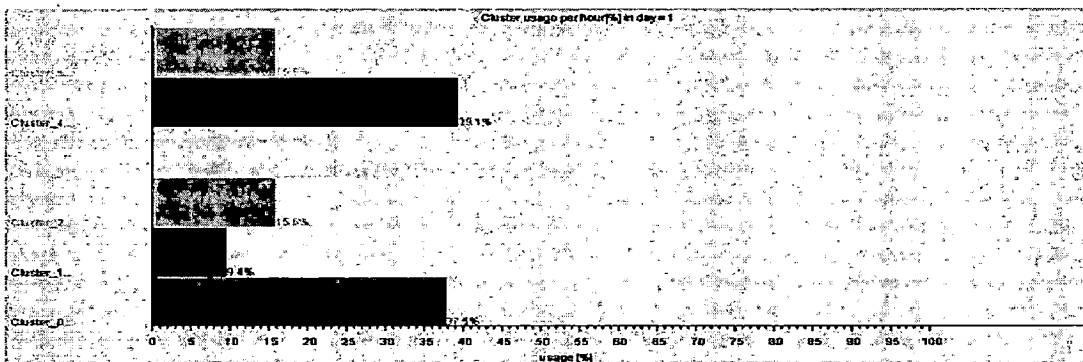


Figure 5.133 Average cluster usage % for workload trace 4 (Periodic checkpointing)



Figure 5.134 gives overall comparison between GA-based adaptive checkpointing using fault ratios of resources and GA-based periodic checkpointing. Values of adaptive checkpointing technique relative to periodic checkpointing are -2.2% for makespan, -2.1% for flowtime, -20.3% for average bounded slowdown, -4.5% for work lost due to failures, -20.5% for number of checkpoints taken and -4% for average turnaround time.

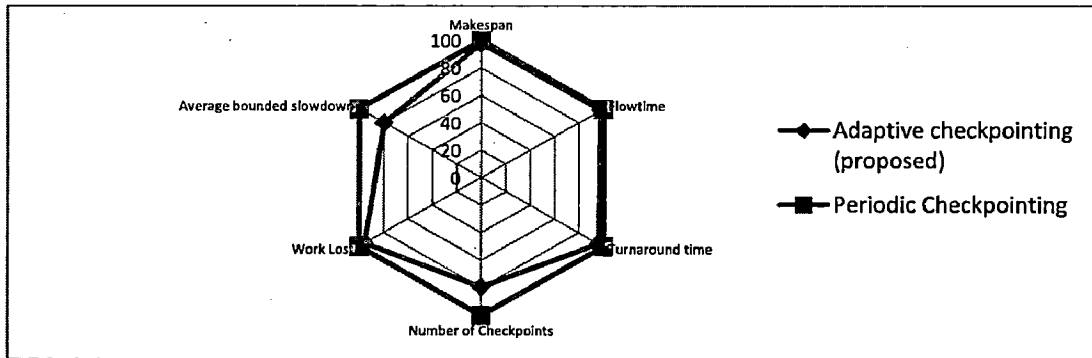


Figure 5.134 Overall comparison between adaptive and periodic checkpointing for workload trace 4

**e) Trace 5 (LLNL-Thunder-2007)**

This trace consisted of 1000 jobs with each job having PE requirement ranging from 4 to 128 PEs. Four clusters each with 8 SMPs with each SMP having 16 processors were used for execution of jobs. Jobs are allocated processors only from one cluster. Resources failures were simulated with first cluster having much higher failure rate. Coordinated checkpointing is used for performing checkpointing operation.

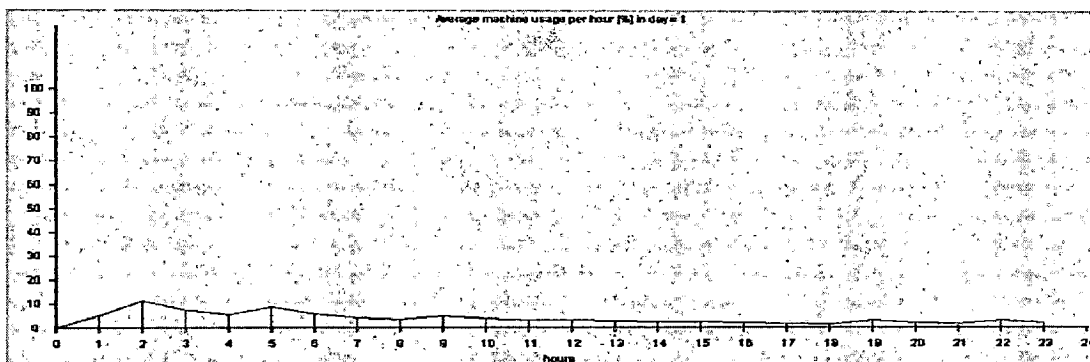


Figure 5.135 Average machine usage per hour for workload trace 5 (Adaptive checkpointing)

Figure 5.135 shows the average machine usage per hour for adaptive checkpointing technique. Figure 5.136 gives cluster usage %age for each cluster for adaptive checkpointing technique.

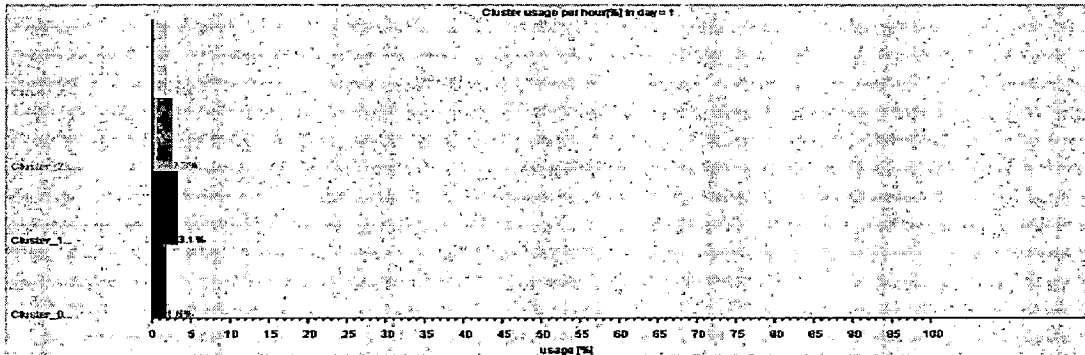


Figure 5.136 Average cluster usage % for workload trace 5 (Adaptive checkpointing)

Figure 5.137 shows the average machine usage per hour for periodic checkpointing technique.

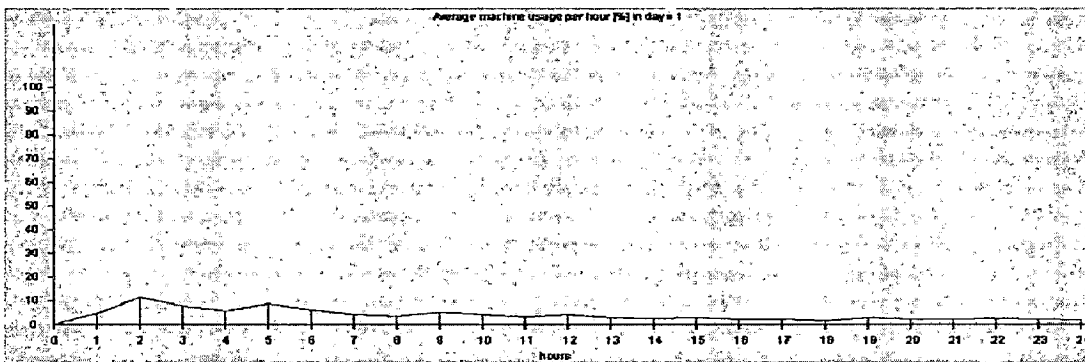


Figure 5.137 Average machine usage per hour for workload trace 5 (Periodic checkpointing)

Figure 5.138 gives cluster usage %age for each cluster for periodic checkpointing technique.

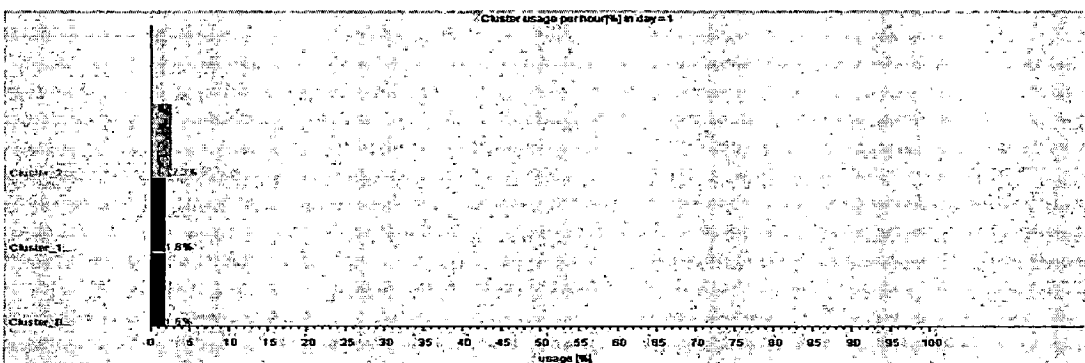


Figure 5.138 Average cluster usage % for workload trace 5 (Periodic checkpointing)

Figure 5.139 gives overall comparison between GA-based adaptive checkpointing using fault ratios of resources and GA-based periodic checkpointing. Values of adaptive checkpointing technique relative to periodic checkpointing are 0% for makespan, 0% for flowtime, -13.6% for average bounded slowdown, -7.2% for work lost due to failures, -26.8% for number of checkpoints taken and -1.85% for average turnaround time.

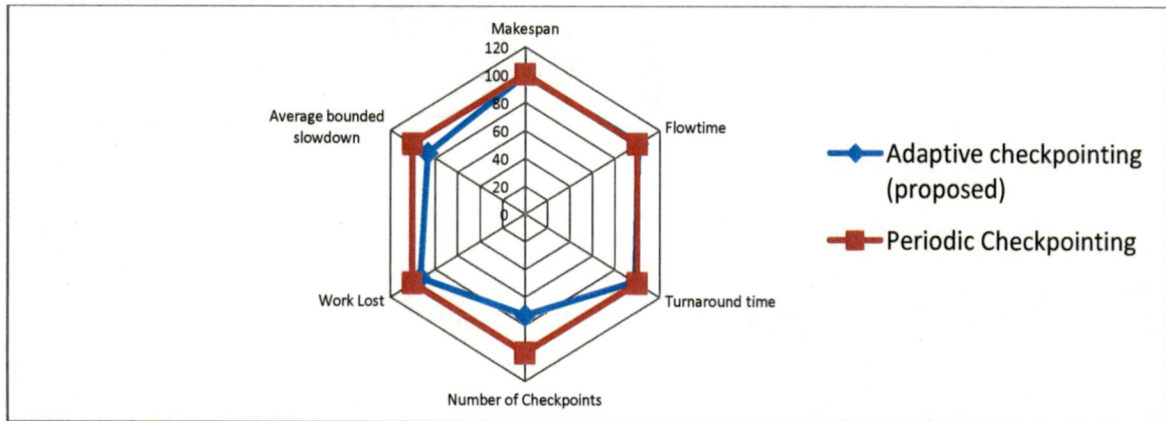


Figure 5.139 Overall comparison between adaptive and periodic checkpointing for workload trace 5

#### 5.14 ACO-based checkpointing techniques performance comparison using workload traces

This section compares ACO-based adaptive checkpointing using fault ratios of resources with ACO-based periodic checkpointing for workload traces from workload trace archives.

##### a) Trace 1(HPC2)

Configuration and parameters are same as in subsection 5.13

Figure 5.140 shows average cluster usage per day for each cluster for adaptive checkpointing technique.

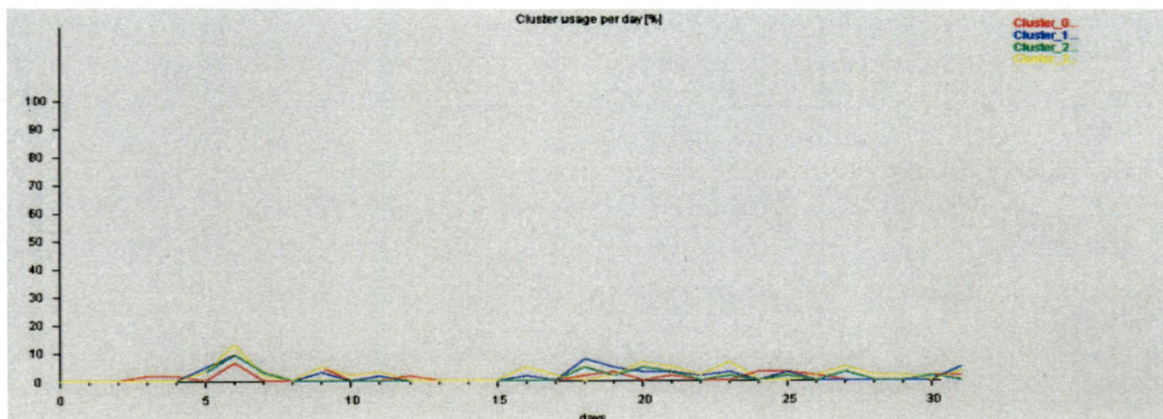


Figure 5.140 Cluster usage per day for workload trace 1 (Adaptive checkpointing)



Figure 5.141 shows number of requested and used CPUs per day for adaptive checkpointing technique.

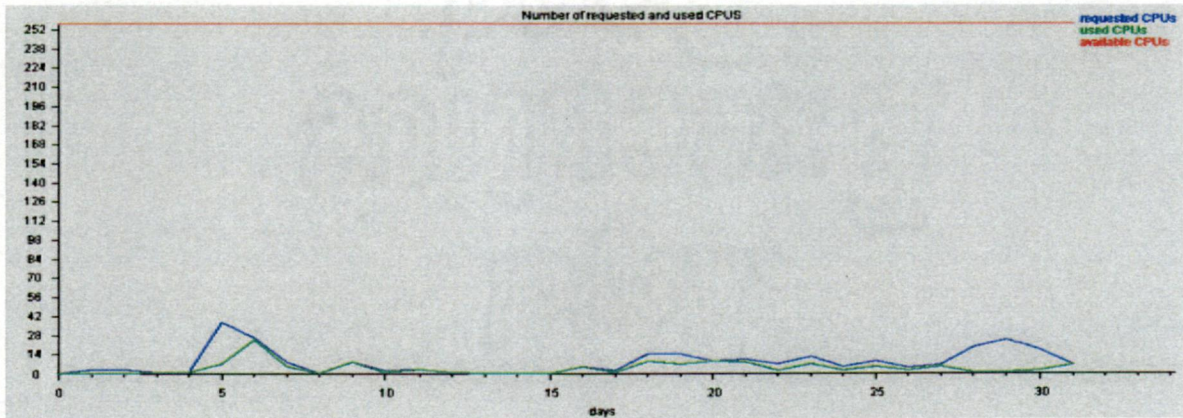


Figure 5.141 Number of requested and used CPUs per day for trace 1 (Adaptive checkpointing)

Figure 5.142 gives average machine usage per day for adaptive checkpointing technique.

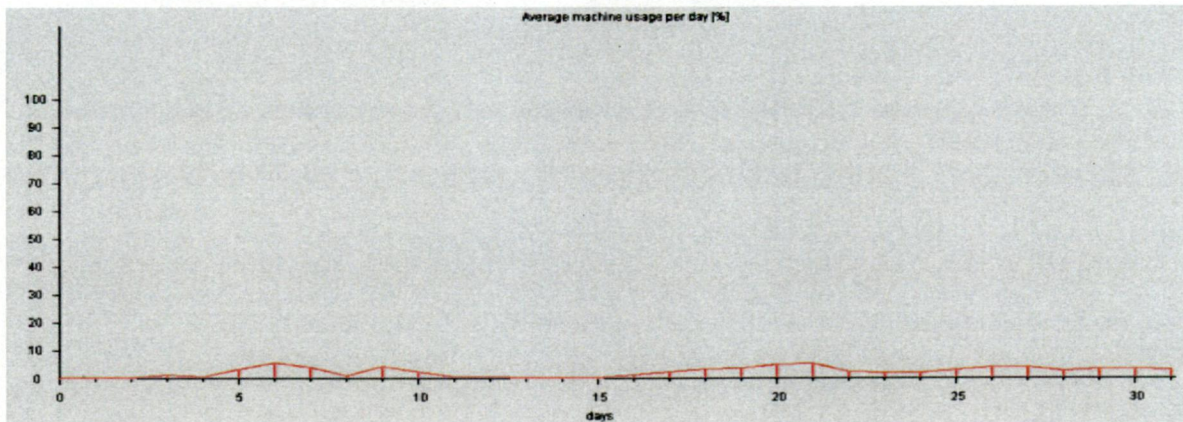


Figure 5.142 Average machine usage per day for trace 1 (Adaptive checkpointing)

Figure 5.143 gives cluster usage per day for periodic checkpointing technique. Cluster usage is very low.

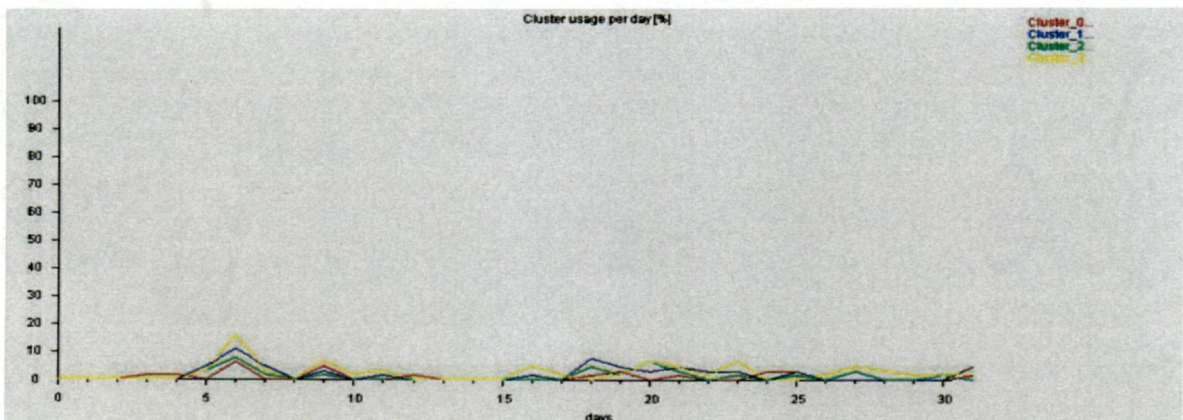


Figure 5.143 Cluster usage per day for workload trace 1 (Periodic checkpointing)



Figure 5.144 shows number of requested and used CPUs per day for periodic checkpointing technique.

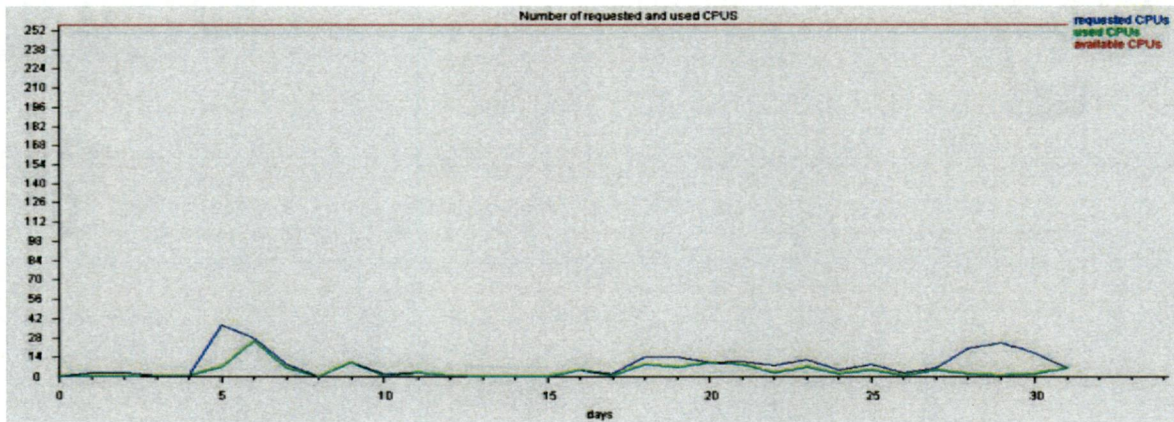


Figure 5.144 Number of requested and used CPUs per day for trace 1 (Periodic checkpointing)

Figure 5.145 gives average machine usage per day for periodic checkpointing technique.

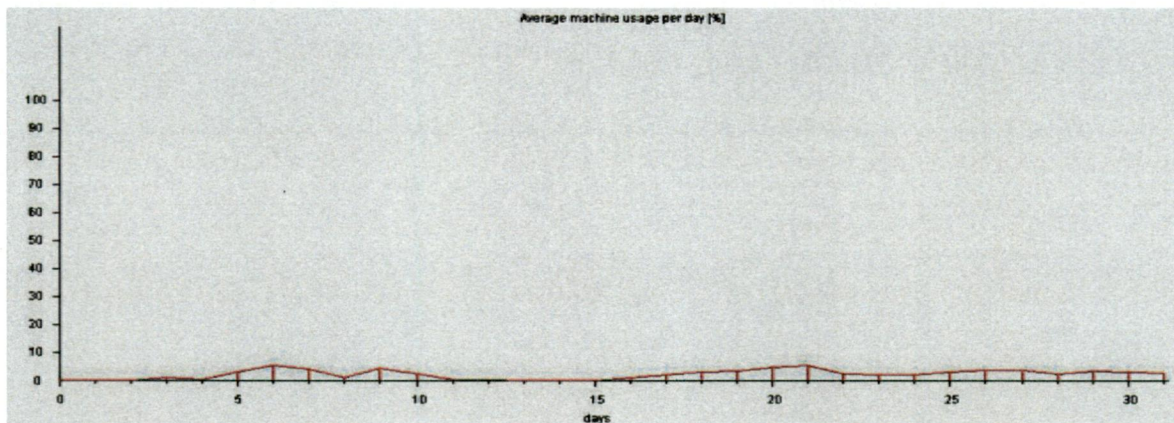


Figure 5.145 Average machine usage per day for trace 1 (Periodic checkpointing)

Figure 5.146 gives average waiting time of jobs submitted to a cluster for each day for each of periodic checkpointing and adaptive checkpointing technique.

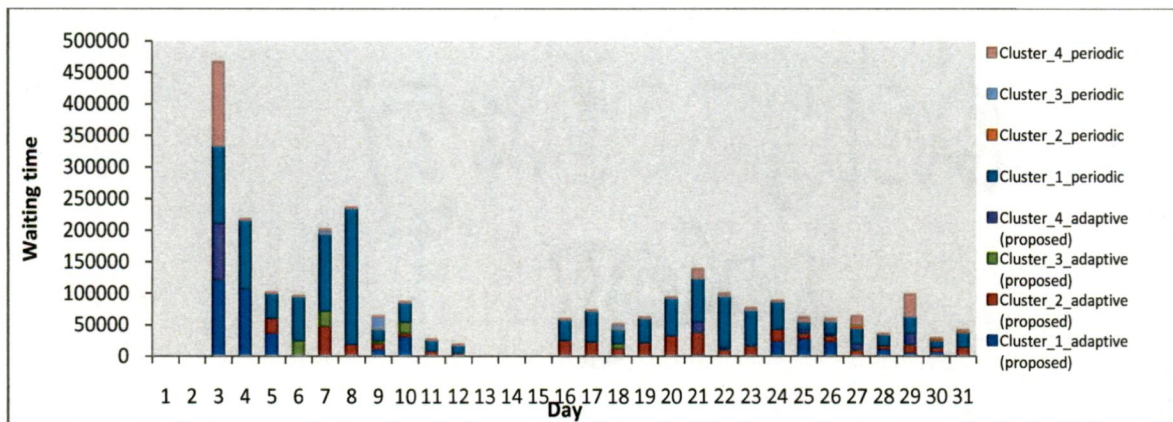


Figure 5.146 Average waiting time per cluster per day for trace 1



Figure 5.147 gives average response time of jobs submitted to a cluster for each day for each of periodic checkpointing and adaptive checkpointing technique.

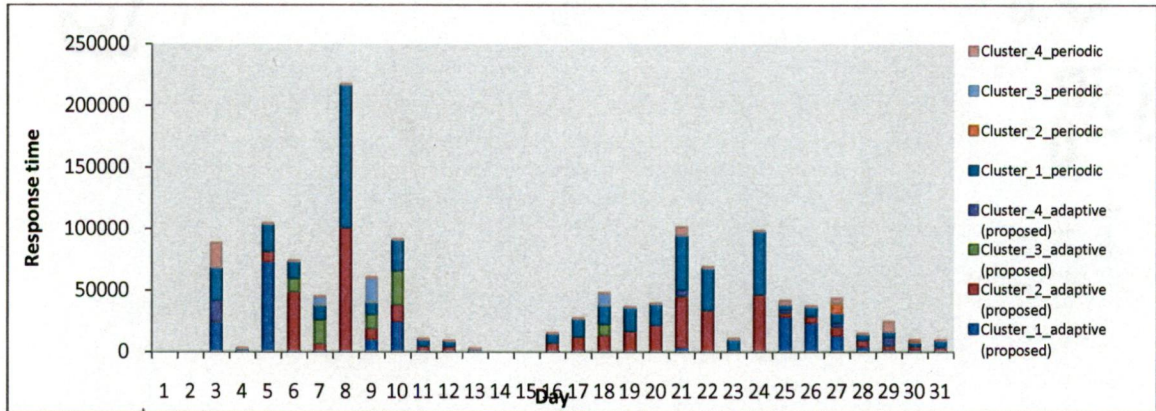


Figure 5.147 Average response time per cluster per day for trace 1

Figure 5.148 gives average tardiness time of jobs submitted to a cluster for each day for each of periodic checkpointing and adaptive checkpointing technique.

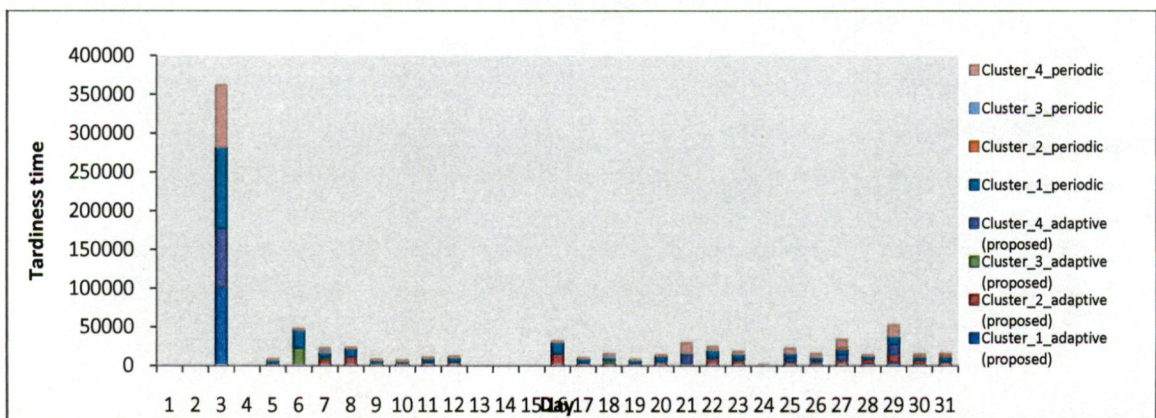


Figure 5.148 Average tardiness time per cluster per day for trace 1

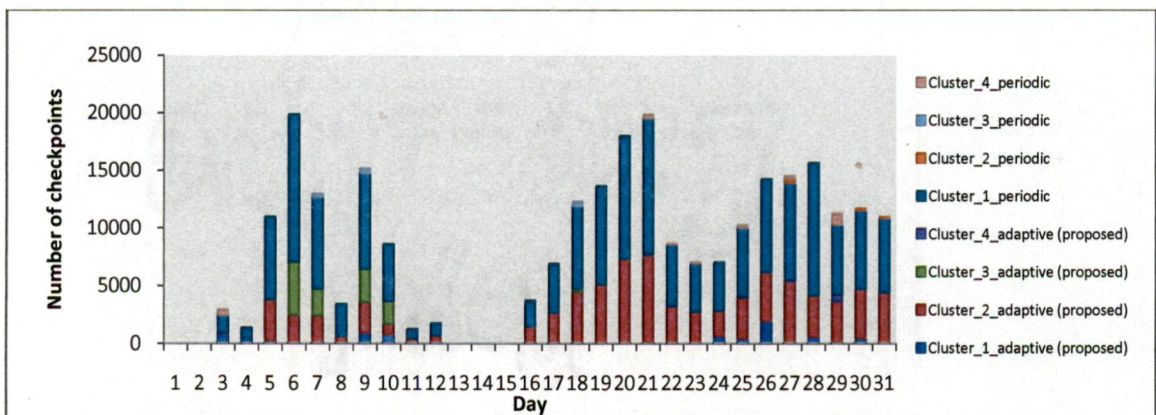


Figure 5.149 Number of checkpoints per cluster per day for trace 1

Figure 5.149 gives number of checkpoints taken on each cluster for each day for each of periodic checkpointing and adaptive checkpointing technique.

Figure 5.150 gives work lost due to failures on each cluster for each day for each of periodic checkpointing and adaptive checkpointing technique.

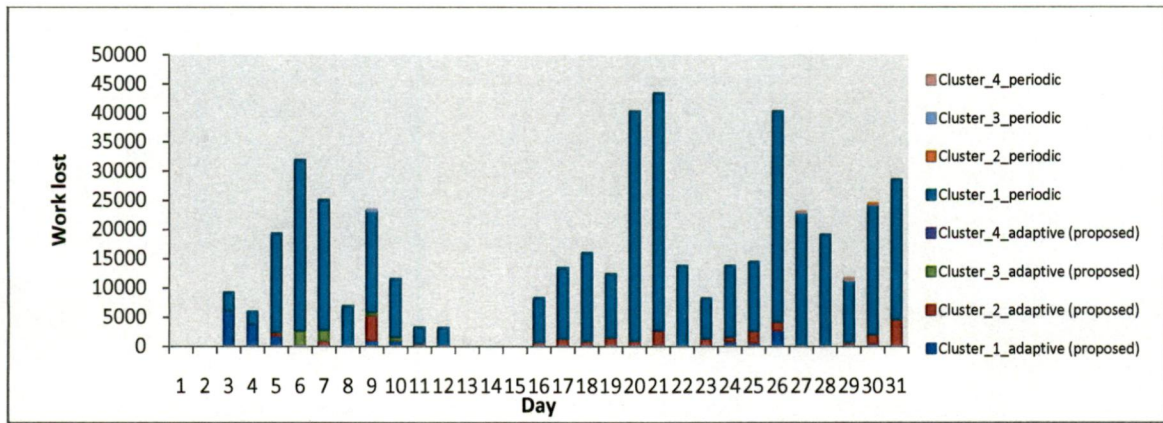


Figure 5.150 Work lost due to failures per cluster per day for trace 1

Figure 5.151 gives overall comparison between ACO-based adaptive checkpointing using fault ratios of resources and ACO-based periodic checkpointing. Values of adaptive checkpointing technique relative to periodic checkpointing are 0% for makespan, 0% for flowtime, -39.6% for average bounded slowdown, -87% for work lost due to failures, -38% for number of checkpoints taken and -9% for average turnaround time.

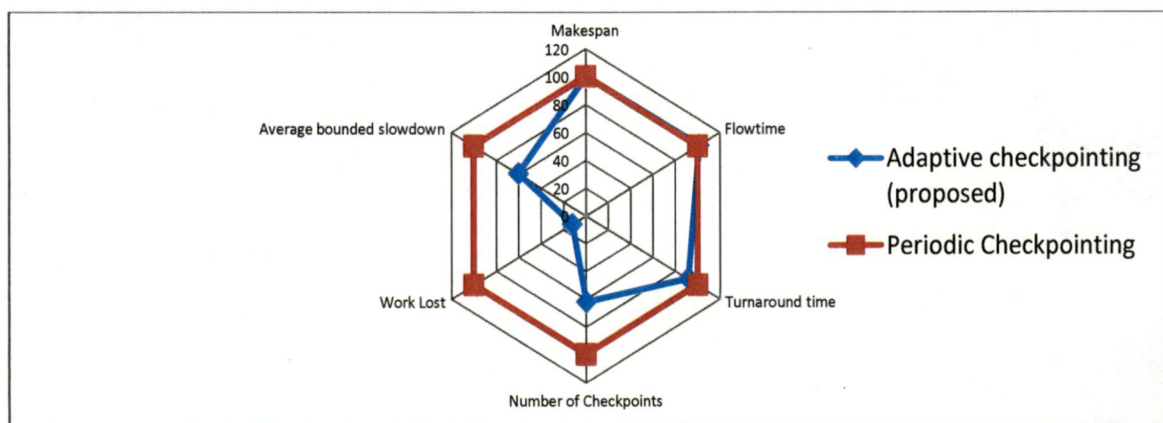


Figure 5.151 Overall comparison between adaptive and periodic checkpointing for workload trace 1

## b) Trace 2 (LCG)

Configuration and parameters are same as in subsection 4.11

Figure 5.152 shows the average machine usage per hour for adaptive checkpointing technique.

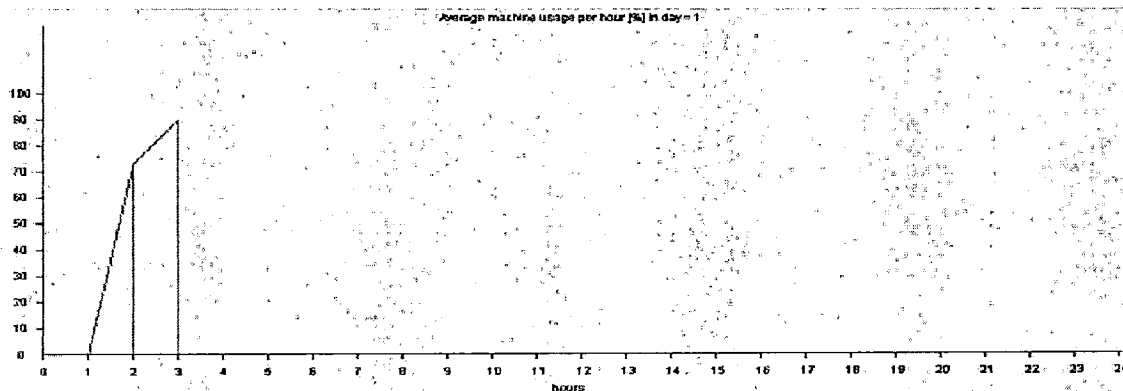


Figure 5.152 Average machine usage per hour for workload trace 2 (Adaptive checkpointing)

Figure 5.153 gives cluster usage %age for each cluster for adaptive checkpointing technique.

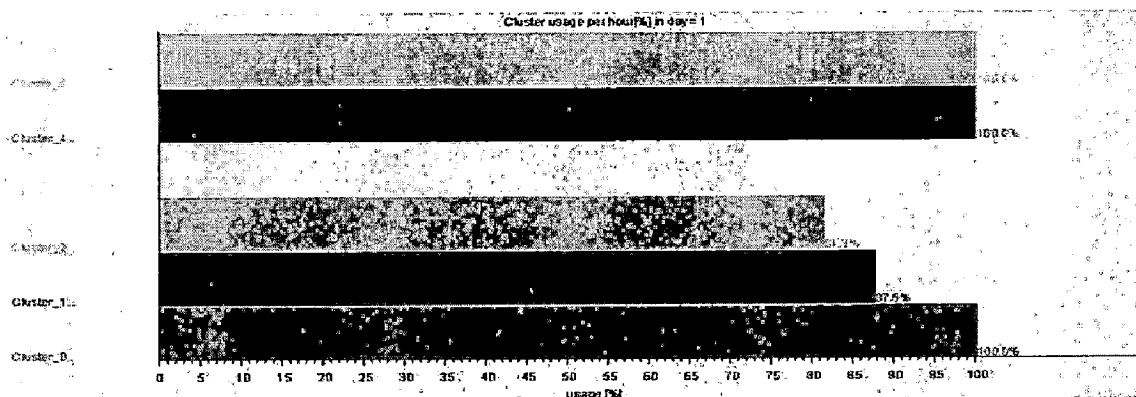


Figure 5.153 Average cluster usage % for workload trace 2 (Adaptive checkpointing)

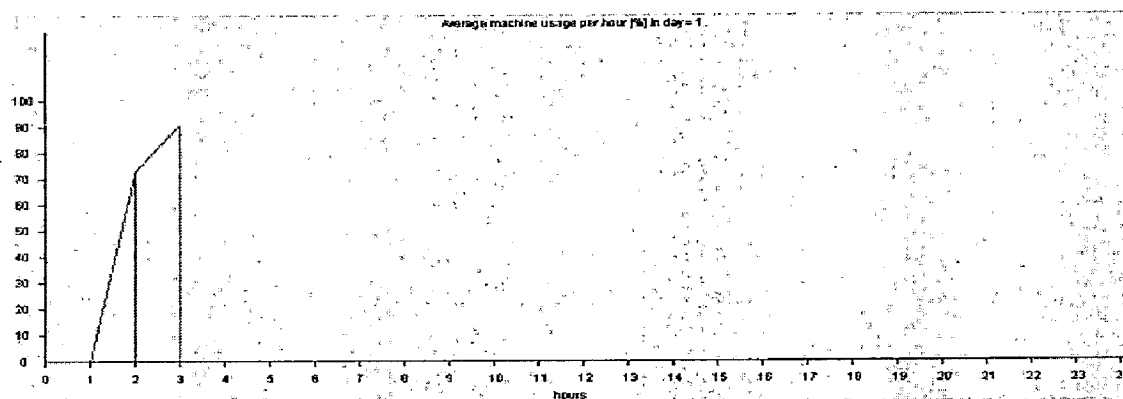


Figure 5.154 Average machine usage per hour for workload trace 2 (Periodic checkpointing)



Figure 5.154 shows the average machine usage per hour for periodic checkpointing technique. Figure 5.155 gives cluster usage %age for each cluster for periodic checkpointing technique.

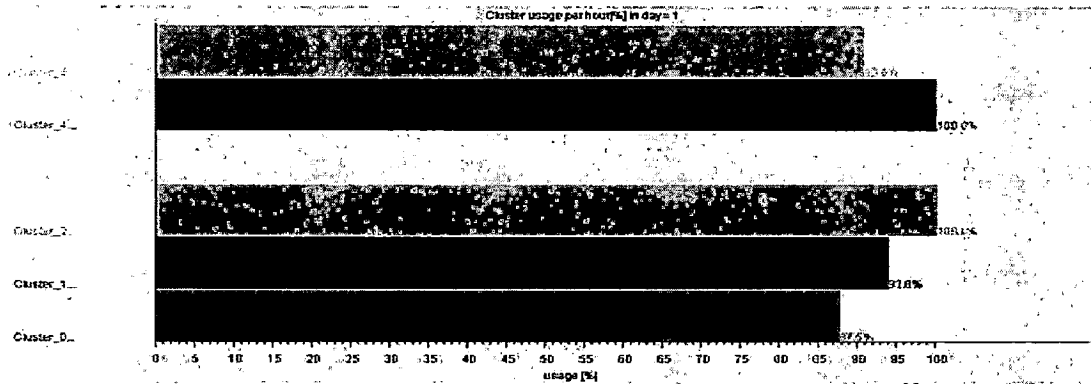


Figure 5.155 Average cluster usage % for workload trace 2 (Periodic checkpointing)

Figure 5.156 gives overall comparison between ACO-based adaptive checkpointing using fault ratios of resources and ACO-based periodic checkpointing. Values of adaptive checkpointing technique relative to periodic checkpointing are 0% for makespan, -3% for flowtime, -22% for average bounded slowdown, +34% for work lost due to failures, -51% for number of checkpoints taken and -2% for average turnaround time.

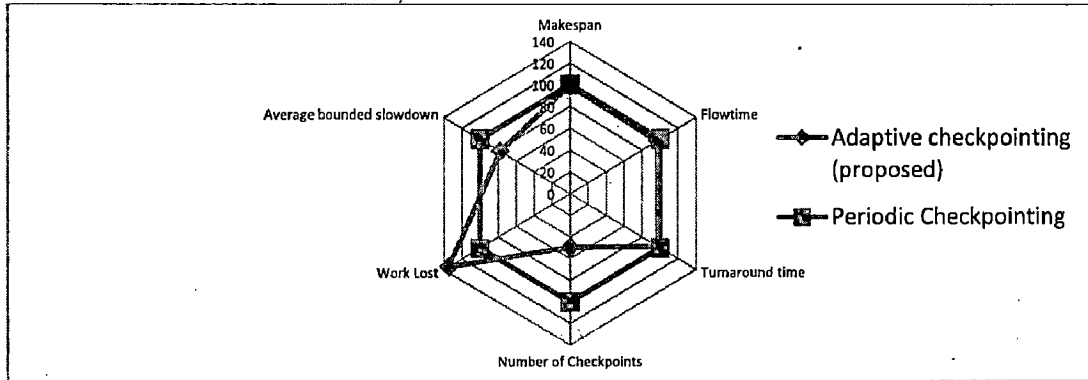


Figure 5.156 Overall comparison between adaptive and periodic checkpointing for workload trace 2

### c) Trace 3 (NASA-iPSC-1993)

Configuration and parameters are same as in subsection 4.11

Figure 5.157 shows average cluster usage per day for each cluster for adaptive checkpointing technique. As can be seen utilization of resources is very low

Figure 5.158 shows number of requested and used CPUs per day for adaptive checkpointing technique.

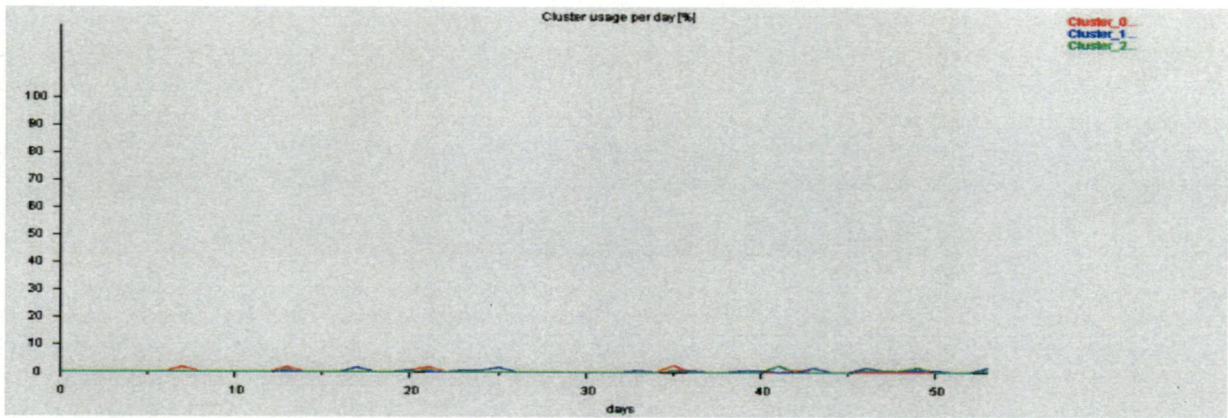


Figure 5.157 Cluster usage per day for workload trace 3 (Adaptive checkpointing)

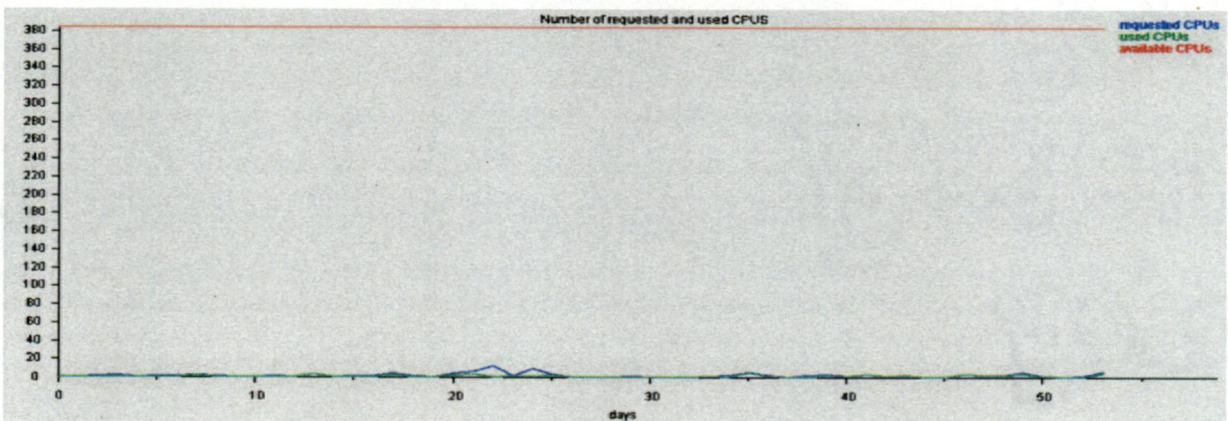


Figure 5.158 Number of requested and used CPUs per day for trace 3 (Adaptive checkpointing)

Figure 5.159 gives average machine usage per day for adaptive checkpointing technique.

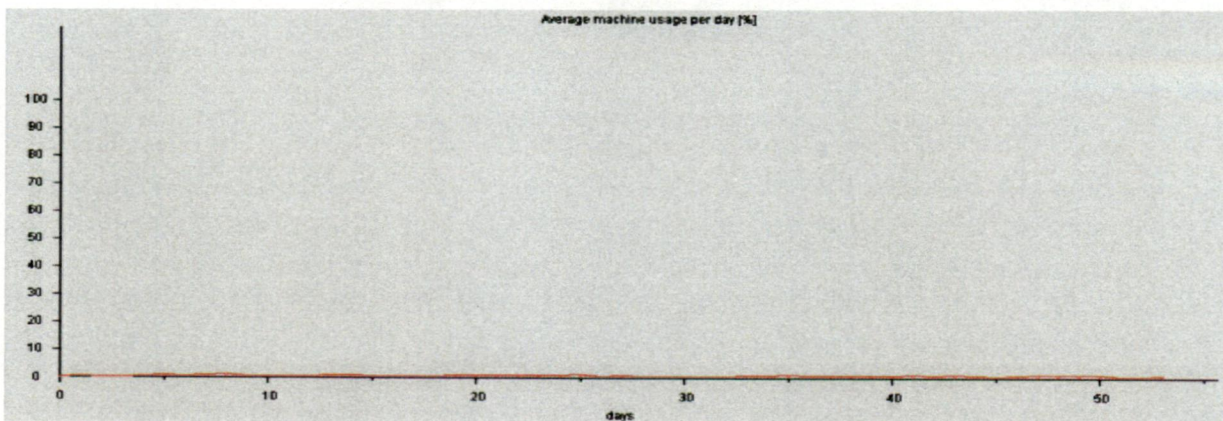


Figure 5.159 Average machine usage per day for trace 3 (Adaptive checkpointing)

Figure 5.160 shows average cluster usage per day for each cluster for periodic checkpointing technique. As can be seen utilization of resources is very low

Figure 5.161 shows number of requested and used CPUs per day for periodic checkpointing technique.



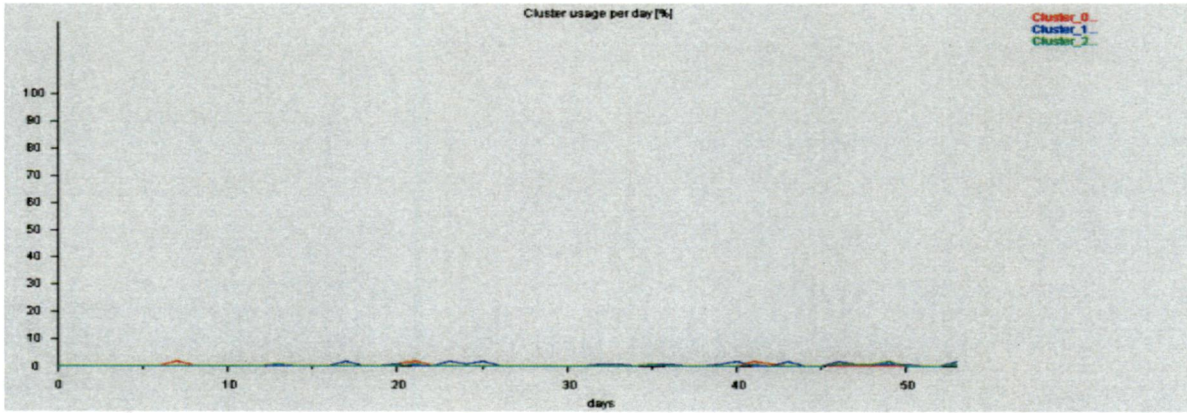


Figure 5.160 Cluster usage per day for workload trace 3 (Periodic checkpointing)

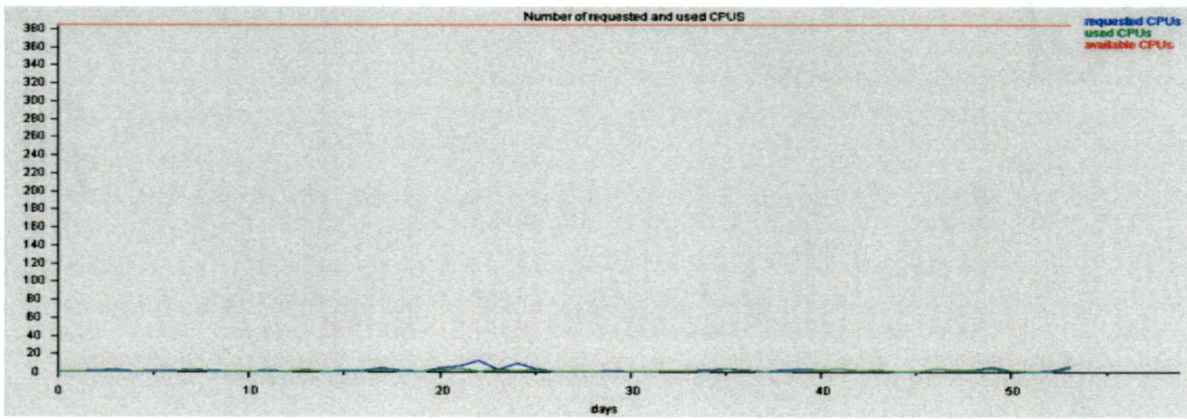


Figure 5.161 Number of requested and used CPUs per day for trace 3 (Periodic checkpointing)

Figure 5.162 gives average machine usage per day for periodic checkpointing technique.

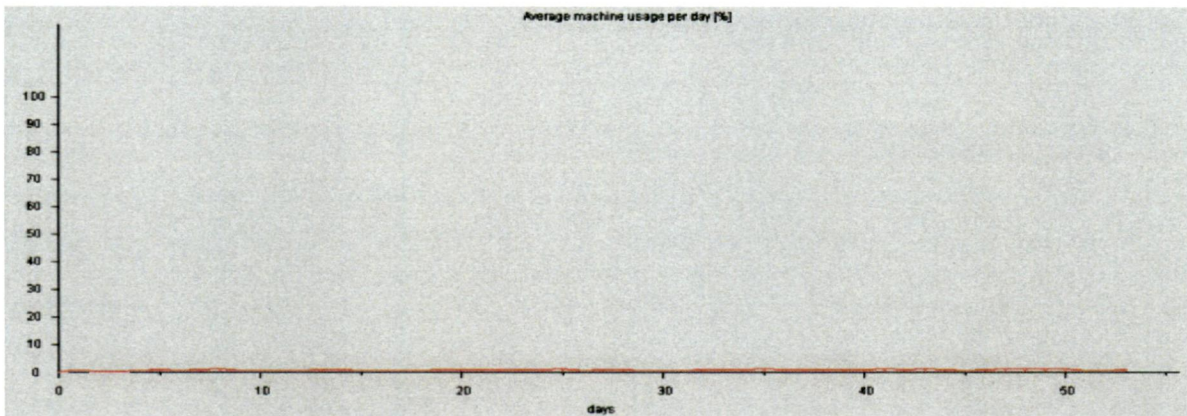


Figure 5.162 Average machine usage per day for trace 3 (Periodic checkpointing)

Figure 5.163 gives average waiting time of jobs submitted to a cluster for each day for each of periodic checkpointing and adaptive checkpointing technique. Figure 5.164 gives average response time of jobs submitted to a cluster for each day for each of periodic checkpointing and adaptive checkpointing technique.



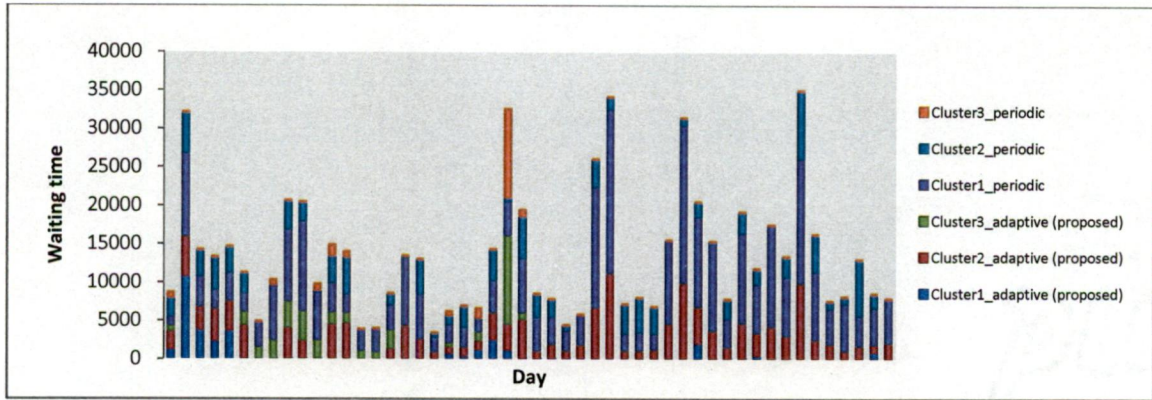


Figure 5.163 Average waiting time per cluster per day for trace 3

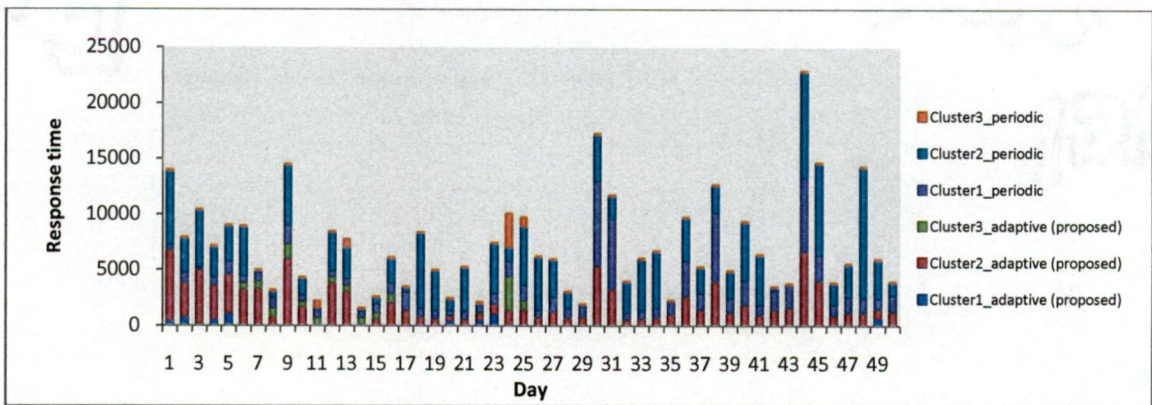


Figure 5.164 Average response time per cluster per day for trace 3

Figure 5.165 gives average tardiness time of jobs submitted to a cluster for each day for each of periodic checkpointing and adaptive checkpointing technique.

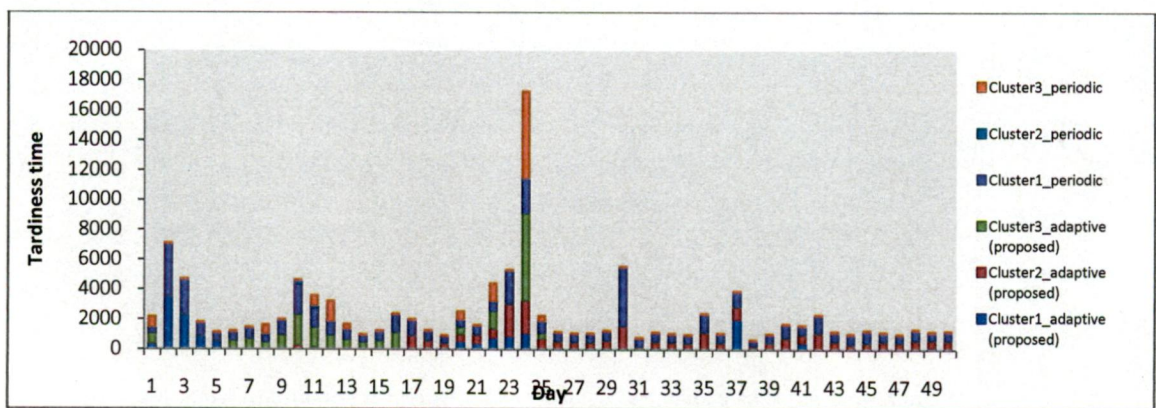


Figure 5.165 Average tardiness time per cluster per day for trace 3

Figure 5.166 gives number of checkpoints on each cluster for each day for each periodic checkpointing and adaptive checkpointing technique. Figure 5.167 gives work lost due to failures on each cluster for

each day for each of periodic checkpointing and adaptive checkpointing technique.

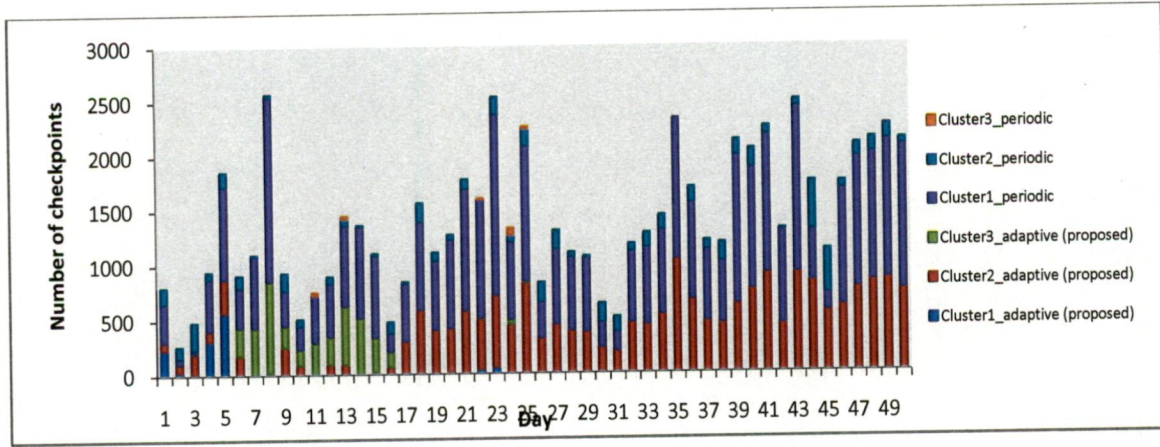


Figure 5.166 Number of checkpoints per cluster per day for trace 3

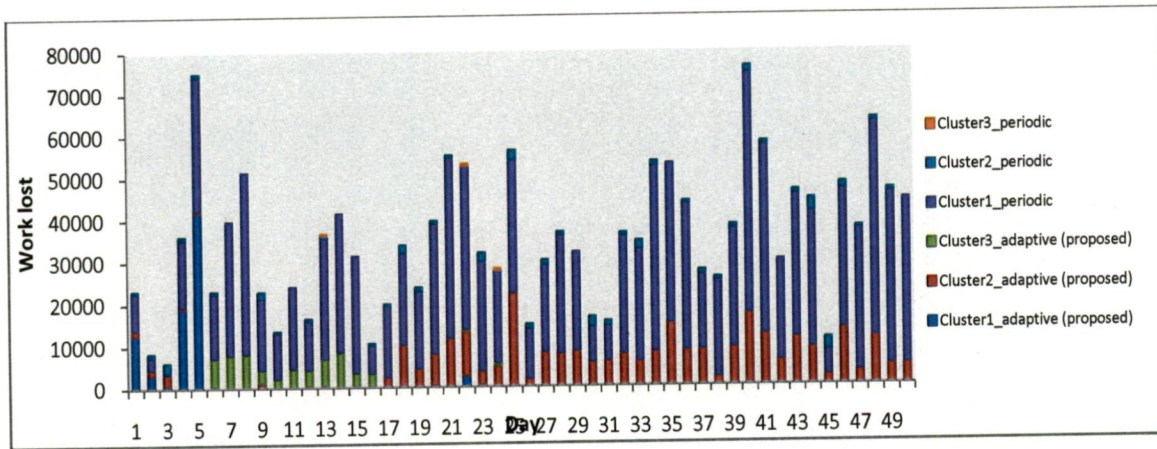


Figure 5.167 Work lost due to failures per cluster per day for trace 3

Figure 5.168 gives overall comparison between ACO-based adaptive checkpointing using fault ratios of resources and ACO-based periodic checkpointing. Values of adaptive checkpointing technique relative to periodic checkpointing are 0% for makespan, -1.6% for flowtime, -43.6% for average bounded

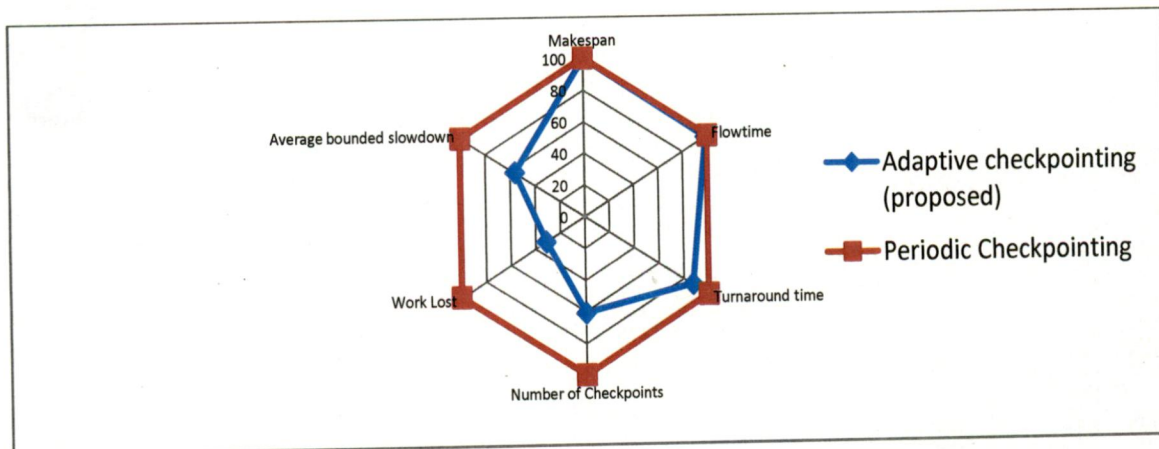


Figure 5.168 Overall comparison between adaptive and periodic checkpointing for workload trace 3



slowdown, -69% for work lost due to failures, -39.1% for number of checkpoints taken and -13% for average turnaround time.

**d) Trace 4 (LCG-2005)**

Configuration and parameters are same as in subsection 4.11

Figure 5.169 shows the average machine usage per hour for adaptive checkpointing technique.

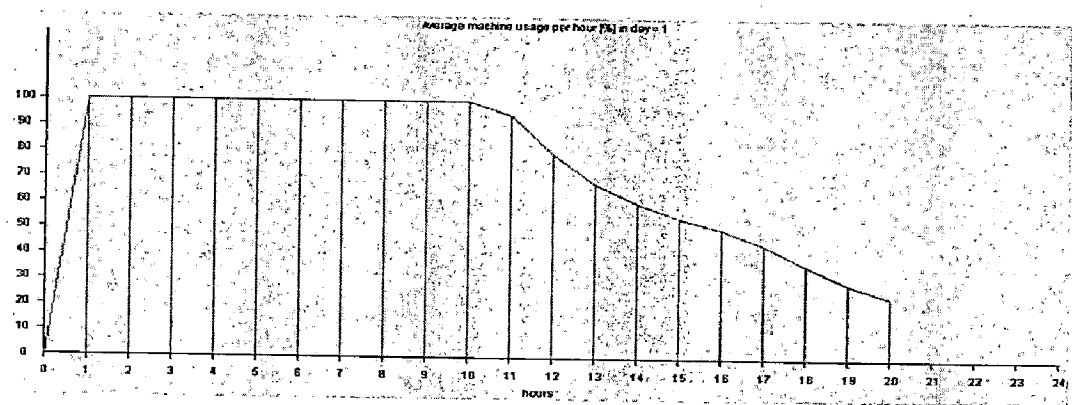


Figure 5.169 Average machine usage per hour for workload trace 4 (Adaptive checkpointing)

Figure 5.170 gives cluster usage %age for each cluster for adaptive checkpointing technique.

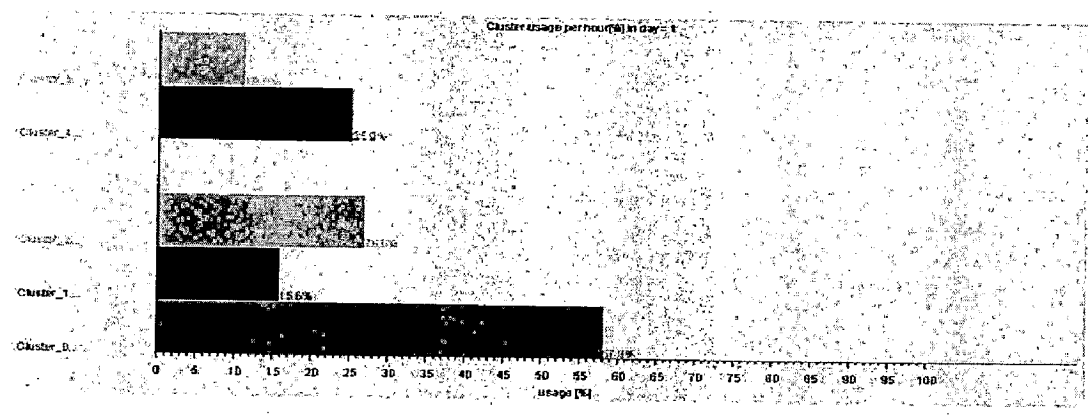


Figure 5.170 Average cluster usage % for workload trace 4 (Adaptive checkpointing)

Figure 5.171 shows the average machine usage per hour for periodic checkpointing technique. Figure

5.172 gives cluster usage %age for each cluster for periodic checkpointing technique.

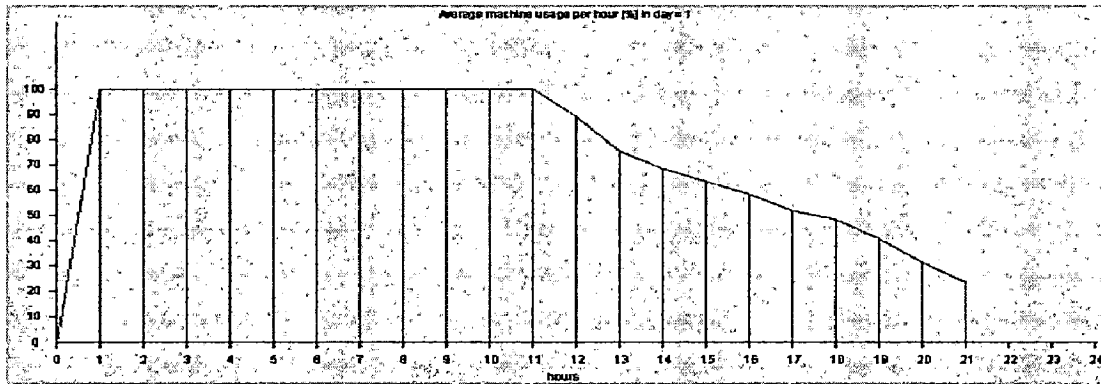


Figure 5.171 Average machine usage per hour for workload trace 4(Periodic checkpointing)

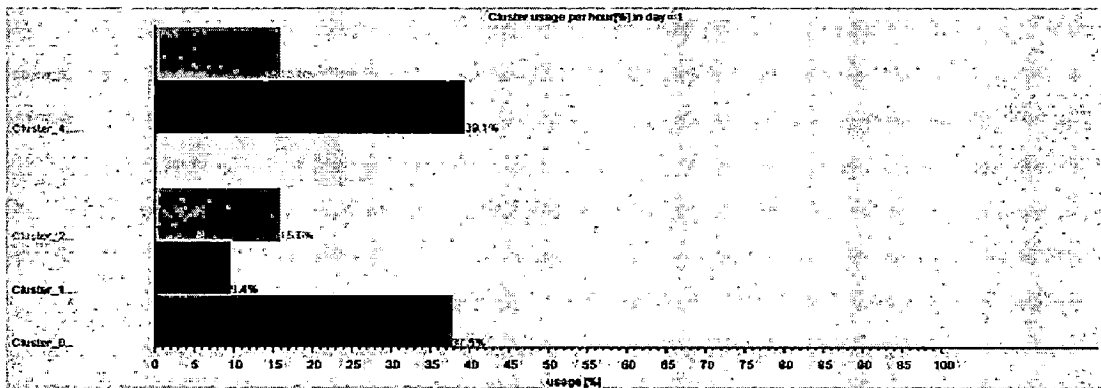


Figure 5.172 Average cluster usage % for workload trace 4 (Periodic checkpointing)

Figure 5.173 gives overall comparison between ACO-based adaptive checkpointing using fault ratios of resources and ACO-based periodic checkpointing. Values of adaptive checkpointing technique relative to periodic checkpointing are -2% for makespan, -2.3% for flowtime, -20.3% for average bounded slowdown, -9.56% for work lost due to failures, -24.5% for number of checkpoints taken and -3.5% for

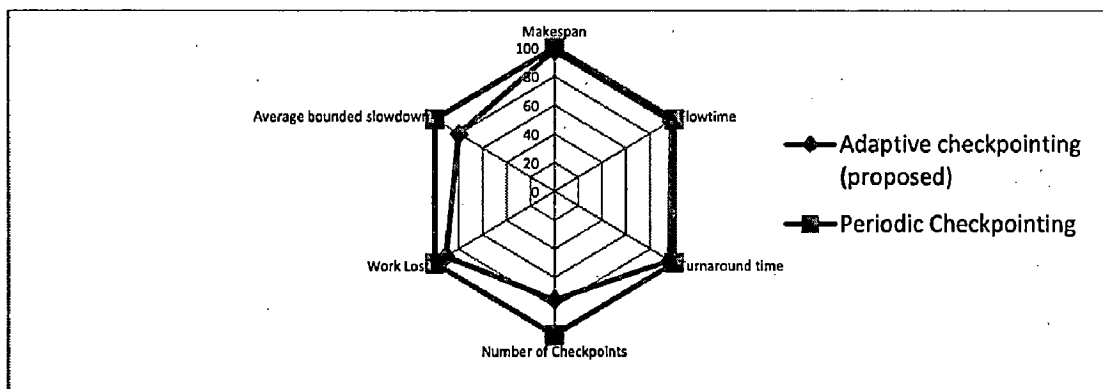


Figure 5.173 Overall comparison between adaptive and periodic checkpointing for workload trace 4

average turnaround time.

e) Trace 5 (LLNL-Thunder-2007).

Configuration and parameters are same as in subsection 4.11

Figure 5.174 shows the average machine usage per hour for adaptive checkpointing technique.

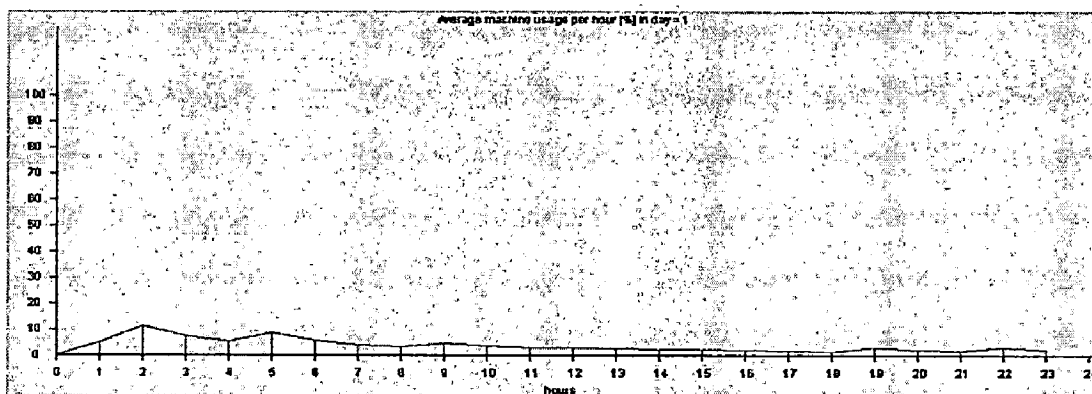


Figure 5.174 Average machine usage per hour for workload trace 5 (Adaptive checkpointing)

Figure 5.175 gives cluster usage %age for each cluster for adaptive checkpointing technique.

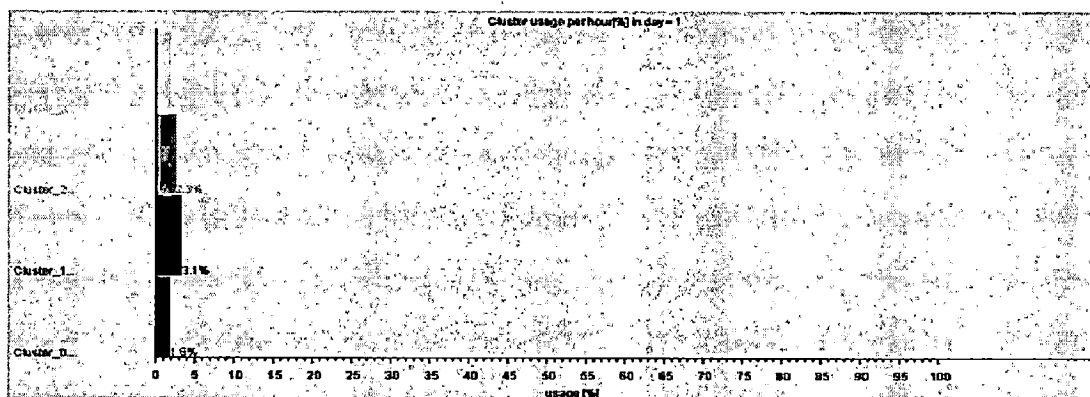


Figure 5.175 Average cluster usage % for workload trace 5 (Adaptive checkpointing)

Figure 5.176 shows the average machine usage per hour for periodic checkpointing technique. Figure 5.177 gives cluster usage %age for each cluster for periodic checkpointing technique.

Figure 5.178 gives overall comparison between ACO-based adaptive checkpointing using fault ratios of resources and ACO-based periodic checkpointing. Values of adaptive checkpointing technique relative to periodic checkpointing are 0% for makespan, +9% for flowtime, -31.5% for average bounded



slowdown, -38 % for work lost due to failures, -43% for number of checkpoints taken and -6% for average turnaround time.

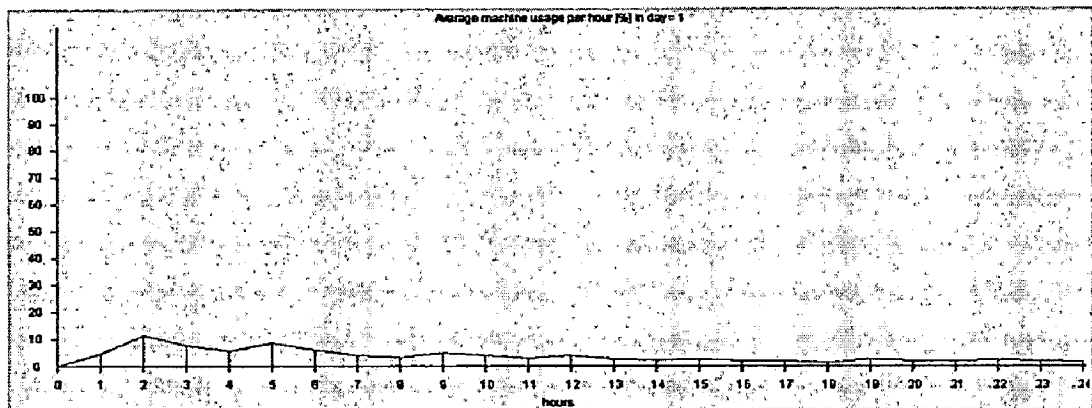


Figure 5.176 Average machine usage per hour for workload trace 5 (Periodic checkpointing)

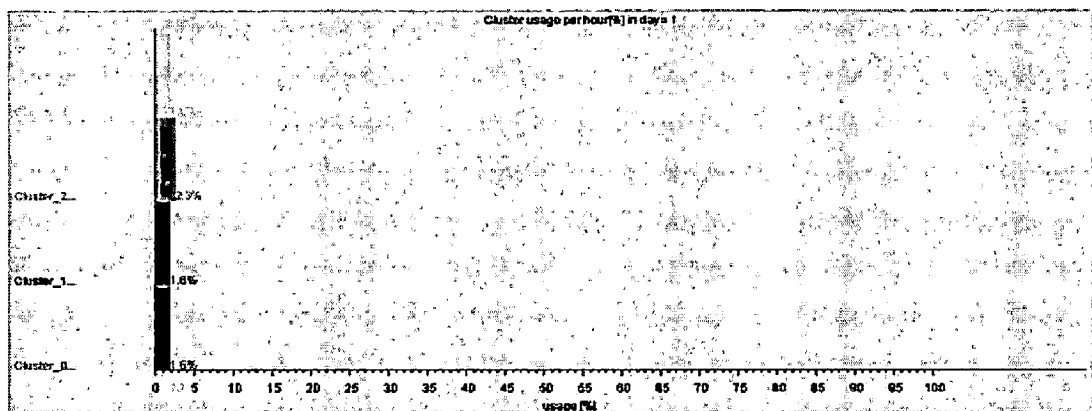


Figure 5.177 Average cluster usage % for workload trace 5 (Periodic checkpointing)

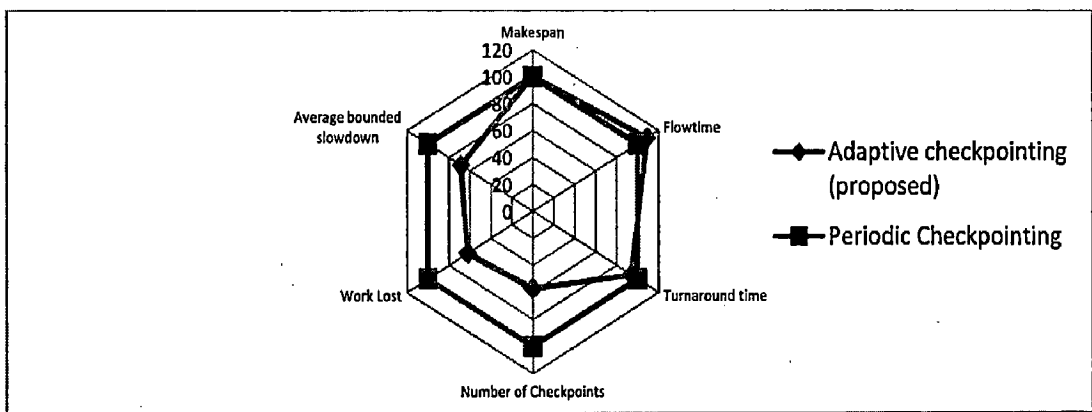


Figure 5.178 Overall comparison between adaptive and periodic checkpointing for workload trace 5

## Chapter 6

### Conclusions and Future Work

---

The work done in this dissertation report can be summarized in following points:

- Design of adaptive checkpointing based fault tolerant heuristics and their incorporation in Genetic Algorithm (GA). These heuristics are based on information related to reliability of resources such as MTBF, fault index and fault ratios. All adaptive checkpointing heuristics have been compared with GA-based periodic checkpointing for a wide range of scenarios.
- Incorporating heuristics designed in Ant Colony Optimization based scheduling in Grid.
- Design of fault index based periodic skip technique and its performance comparison with periodic skip.
- Design of adaptive checkpointing based on information about MTBF and last failure time of resources.
- Design of experimental scenarios for testing performance of various techniques for temporally and spatially correlated failures.
- Performance comparison of ACO-based and GA-based fault tolerance techniques using real failure traces available from Failure Trace Archive.
- Performance comparison of ACO-based and GA- based fault tolerance techniques for real workload traces available from various parallel workloads archives.

The techniques developed dynamically adapt the checkpoint interval depending on current resource conditions. This is particularly crucial for Grid due to being a highly dynamic environment. Adapting the checkpoint interval helps in reducing the time wasted due to high checkpoint overhead or due to work lost due to failures. This helps in delivering acceptable QoS such as that related to turnaround time, makespan etc to users. Another major contribution from this work is scheduling support for these adaptive checkpointing techniques. Scheduling support takes fault information of resources on account while allocating jobs to them. This helps in correctly deciding the number of jobs to be allocated for various resources based on their availability information. For this two very popular metaheuristics – Genetic Algorithm and Ant

Colony Optimization based Grid scheduling have been designed with fault tolerance support. Also the range of experimental scenarios developed is this work unique in itself with experiments developed for real workload traces, real failure traces, temporally and spatially correlated failures.

Following conclusions can be derived from the work done.

- Adaptive checkpointing techniques developed give significant performance improvement over periodic checkpointing technique in terms of makespan, flowtime, average bounded slowdown, and turnaround time.
- Periodic skip, exponential skip performed better for low checkpointing intervals. This is due to less checkpointing overhead.
- Incorporating of fault tolerance and adaptive checkpointing in scheduling decisions helped in finding appropriate schedule for a batch of jobs. This particularly helped when failures are assumed to be temporally and spatially correlated.
- Makespan and flowtime parameters for some workload traces didn't give much performance improvement. This is due to jobs having highly varying arrival times.
- Adaptive periodic skip performed better than periodic skip. This is due to adaptive adjustment of the checkpointing interval depending on failure conditions of resources.
- Work lost due to failures was very high for adaptive checkpointing techniques for some experiments. This is not an issue as total work lost is a combination of work lost due to failures and amount of work done in checkpointing operations (total checkpointing overhead). The combination of these two parameters is very low for adaptive checkpointing techniques.

Future work directions are based on the work left due to time constraints. These are

- Traces – workload trace and failure traces used in this work are small portions of available traces. Future works will focus on using complete trace for evaluation
- Experiments have been performed for workload and failure traces separately. Future works will use both workload trace and failure trace in an experiment.
- Downtime (MTTR) of resources is ignored in this work and resource is assumed to recover immediately from failure. This assumption will be removed in future.

- Checkpointing technique used considers restart after failure on the same resource. Another technique can be checkpoint with migration where job is restarted on a different resource. This technique along with its various issues such as spare node allocation is to be pondered upon.
- Heuristics developed for fault tolerance are not restricted to metaheuristics. Rather they can be incorporated in any scheduling algorithm. Future work will look into that.
- This work considers only transient faults on resources. Other fault classes are not considered.
- Finally future work will focus on working in an actual Grid setup rather than simulated one.

## References

- [1] Townend, P. and Xu, J. 2003. Fault tolerance within a grid environment. As component of e-Demand project at the University of Durham, United Kingdom.
- [2] Foster I., Kesselman C., *The Grid: Blueprint for a New Computing Infrastructure*, The Elsevier Series in GridComputing.
- [3] Foster, I. 2001. The anatomy of the grid: enabling scalable virtual organizations. In *Proceedings of the First IEEE/ACM International Symposium on Cluster Computing and the Grid 2001* (Brisbane, Australia May 15-18, 2001). CCGRID '01. IEEE Computer Society, Washington, DC, USA, 6-7.
- [4] Foster I. "What is the Grid? A three point checklist," Argonne National Laboratory, [fp.mcs.anl.gov/~foster/Articles/WhatIsTheGrid.pdf](http://fp.mcs.anl.gov/~foster/Articles/WhatIsTheGrid.pdf), 2002.
- [5] Avizienis, A.; Laprie, J.-C.; Randell, B.; Landwehr, C.; , "Basic concepts and taxonomy of dependable and secure computing," *IEEE Transactions on Dependable and Secure Computing*, vol.1, no.1, pp. 11- 33, Jan.-March 2004.
- [6] Huda, M.T.; Schmidt, H.W.; Peake, I.D., "An agent oriented proactive fault-tolerant framework for grid computing," *First International Conference on e-Science and Grid Computing, 2005*, pp. 8-15, July 2005.
- [7] Hofer, J.; Fahringer, T., "A Multi-Perspective Taxonomy for Systematic Classification of Grid Faults," *16th Euromicro Conference on Parallel, Distributed and Network-Based Processing, 2008. PDP 2008*, pp.126-130, 13-15 Feb. 2008.
- [8] Jafar, S.; Krings, A.; Gautier, T.; , "Flexible Rollback Recovery in Dynamic Heterogeneous Grid Computing," *Dependable and Secure Computing, IEEE Transactions on* , vol.6, no.1, pp.32-44, Jan.-March 2009
- [9] Avizienis, A., "The N-Version Approach to Fault-Tolerant Software," *IEEE Transactions on Software Engineering*, vol. SE-11, no.12, pp. 1491- 1501, Dec. 1985.

- [10] Schroeder B. and G. A. Gibson (2006). "A large-scale study of failures in high-performance Computing systems," *International Conference on Dependable Systems and Networks, DSN 2006*.
- [11] Hayashibara N, Cherif A, Katayama T. "Failure detectors for large-scale distributed systems," *Proceedings of the 21<sup>st</sup> IEEE Symposium on Reliable Distributed Systems*. IEEE Computer Society Press: Los Alamitos, CA, 2002, 404-409 October 2002.
- [12] Elnozahy E, Johnson D, Wang Y. "A survey of rollback recovery protocols in message-passing systems," *ACM Computing Surveys 2002*, 34(3), 375–408.
- [13] Alvisi L. and K. Marzullo (1998). "Message logging: pessimistic, optimistic, causal, and optimal," *IEEE Transactions on Software Engineering*, 24(2), 149-159.
- [14] H-C Nam, J. Kim, SJ. Hong and S. Lee. "Probabilistic checkpointing," *In Proceedings of the Twenty Seventh International Symposium on Fault-Tolerant Computing (FTCS-27)*, pp.48—57, June 1997.
- [15] Gabriel Rodríguez, Xoán C. Pardo, María J. Martín, Patricia González, Performance evaluation of an application-level checkpointing solution on grids, *Future Generation Computer Systems*, Volume 26, Issue 7, July 2010, Pages 1012-1023, ISSN 0167-739X, 10.1016/j.future.2010.04.016.
- [16] Oliner, A.J.; Sahoo, R.K.; Moreira, J.E.; Gupta, M.; , "Performance implications of periodic checkpointing on large-scale cluster systems," *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International* , vol., no., pp. 8 pp., 4-8 April 2005
- [17] Plank, J.S.; Elwasif, W.R.; , "Experimental assessment of workstation failures and their impact on checkpointing systems," *Fault-Tolerant Computing, 1998. Digest of Papers. Twenty-Eighth Annual International Symposium on* , vol., no., pp.48-57, 23-25 Jun 1998.
- [18] Adam J. Oliner, Larry Rudolph, and Ramendra K. Sahoo. 2006. Cooperative checkpointing: a robust approach to large-scale systems reliability. In *Proceedings of the 20th annual international conference on Supercomputing (ICS '06)*. ACM, New York, NY, USA, 14-23.

- [19] Oliner, A.; Sahoo, R.; , "Evaluating cooperative checkpointing for supercomputing systems," *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International* , vol., no., pp.8 pp., 25-29 April 2006.
- [20] Oliner, A.; Rudolph, L.; Sahoo, R.; , "Cooperative checkpointing theory," *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International* , vol., no., pp.10 pp., 25-29 April 2006.
- [21] Zhiling Lan; Yawei Li; , "Adaptive Fault Management of Parallel Applications for High-Performance Computing," *Computers, IEEE Transactions on* , vol.57, no.12, pp.1647-1660, Dec. 2008.
- [22] Chtepen M., F. H. A. Claeys, et al. (2009). "Adaptive Task Checkpointing and Replication: Toward Efficient Fault-Tolerant Grids," *IEEE Transactions on Parallel and Distributed Systems*, vol.20, no.2, pp.180-190, Feb. 2009.
- [23] Nazir B., K. Qureshi, et al. (2009). "Adaptive checkpointing strategy to tolerate faults in economy based grid," *The Journal of Supercomputing*, 50(1), 1-18, 2009.
- [24] Antonios Litke, Konstantinos Tserpes, Konstantinos Dolkas, and Theodora Varvarigou. 2005. A task replication and fair resource management scheme for fault tolerant grids. In *Proceedings of the 2005 European conference on Advances in Grid Computing (EGC'05)*, Peter A. Sloot, Alfons G. Hoekstra, Thierry Priol, Alexander Reinefeld, and Marian Bubak (Eds.). Springer-Verlag, Berlin, Heidelberg, 1022-1031.
- [25] Qin Z., B. Veeravalli, et al. (2009). "On the Design of Fault-Tolerant Scheduling Strategies Using Primary-Backup Approach for Computational Grids with Low Replication Costs," *IEEE Transactions on Computers*, vol. 58, no.3, pp.380-393, March 2009.
- [26] Hwang S., and Kesselman C., "A flexible framework for fault tolerance in the grid," *Journal of Grid Computing*, vol. 1, no. 3, pp. 251-272, 2003.
- [27] Lopes R. F. and F. J. da Silva e Silva (2006). "Fault tolerance in a mobile agent based computational grid," *Sixth IEEE International Symposium on Cluster Computing and the Grid Workshops, 2006*, vol. 2, 16-19 May 2006.

- [28] Kandaswamy G., A. Mandal, et al. (2008). "Fault Tolerance and Recovery of Scientific Workflows on Computational Grids," *8th IEEE International Symposium on Cluster Computing and the Grid, 2008. CCGRID'08*, pp.777-782, 19-22 May 2008.
- [29] Yang Z., A. Mandal, et al. (2009). "Combined Fault Tolerance and Scheduling Techniques for Workflow Applications on Computational Grids," *9th IEEE/ACM International Symposium on Cluster Computing and the Grid. CCGRID'09*, pp.244-251, 18-21 May 2009.
- [30] SungJin C., B. MaengSoon, et al. (2004). "Volunteer availability based fault tolerant scheduling mechanism in desktop grid computing environment," *Proceedings of the third IEEE International Symposium on Network Computing and Applications, 2004, (NCA 2004)*, pp. 366-371, 30 Aug.-1 Sept. 2004.
- [31] Hou E. S. H., N. Ansari, et al. (1994). "A genetic algorithm for multiprocessor scheduling," *IEEE transactions on Parallel and Distributed Systems*, vol.5, no.2, pp.113-120, Feb 1994.
- [32] Song, S., Hwang, K., and Kwok, K. 2006. Risk-resilient heuristics and genetic algorithms for security-assured grid job scheduling. *IEEE Transactions on Computers* 55, 6 (June 2006), 703-719.
- [33] Khanli, L. M., Far, M. E., and Rahmani, A. M. 2010. RFOH: A New Fault Tolerant Job Scheduler in Grid Computing. In *Proceedings of the Second International Conference on Computer Engineering and Applications* (Bali Island, Indonesia, March 19 – 21, 2010).ICCEA '10. IEEE Computer Society, Washington, DC, USA, 422-425.
- [34] Priya, S.B., Prakash, M., Dhawan, K.K. 2007. Fault Tolerance-Genetic Algorithm for Grid Task Scheduling using Checkpoint In *Proceedings of the Sixth International Conference on Grid and Cooperative Computing* ( Los Alamitos, CA, Aug. 16 – 18, 2007). GCC '07. 676-680.
- [35] Abdulal, W., and Ramachandram, S 2011. Reliability-Aware Genetic Scheduling Algorithm in Grid Environment. In *Proceedings of the International Conference on Communication Systems and Network Technologies* (Katra, Jammu India , June 03 – 05). 673-677.
- [36] Wu, C., Lai K., and Sun R. 2008. GA-Based Job Scheduling Strategies for Fault Tolerant Grid Systems. In *Proceedings of the Asia-Pacific Conference on Services Computing* (Dec. 09–



12, 2008). IEEE, 27-32.

[37] Dorigo, M.; Gambardella, L.M.; "Ant colony system: a cooperative learning approach to the traveling salesman problem," *Evolutionary Computation, IEEE Transactions on* , vol.1, no.1, pp.53-66, Apr 1997

[38] Zhihong Xu; Xiangdan Hou; Jizhou Sun; , "Ant algorithm-based task scheduling in grid computing," *Electrical and Computer Engineering, 2003. IEEE CCECE 2003. Canadian Conference on* , vol.2, no., pp. 1107- 1110 vol.2, 4-7 May 2003.

[39] Hui Yan; Xue-Qin Shen; Xing Li; Ming-Hui Wu; , "An improved ant algorithm for job scheduling in grid computing," *Machine Learning and Cybernetics, 2005. Proceedings of 2005 International Conference on* , vol.5, no., pp.2957-2961 Vol. 5, 18-21 Aug. 2005.

[40] Yanyong Zhang, Mark S. Squillante, Anand Sivasubramaniam, and Ramendra K. Sahoo. 2004. Performance implications of failures in large-scale cluster scheduling. In *Proceedings of the 10th international conference on Job Scheduling Strategies for Parallel Processing (JSSPP'04)*, Dror G. Feitelson, Larry Rudolph, and Uwe Schwiegelshohn (Eds.). Springer-Verlag, Berlin, Heidelberg, 233-252.

[41] Dalibor Klusáček and Hana Rudová. 2010. Alea 2: job scheduling simulator. In *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques (SIMUTools '10)*. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), ICST, Brussels, Belgium, Belgium.

[42] S. Lorpunmanee , Mohd Sap , A.H.Abdullah and C. C. Inwai , "An Ant Colony Optimization for Dynamic Job Scheduling in GridEnvironment" , *International Journal of Computer and Information Science and Engineering* , 2007.

[43] Jing Hu, Mingchu Li, Weifeng Sun, Yuanfang Chen, "An Ant Colony Optimization for Grid Task Scheduling with Multiple QoS Dimensions," *Grid and Cloud Computing, International Conference on*, pp. 415-419, 2009 Eighth International Conference on Grid and Cooperative Computing, 2009

- [44] Ruay-Shiung Chang, Jih-Sheng Chang, Po-Sheng Lin, An ant algorithm for balanced job scheduling in grids, *Future Generation Computer Systems*, Volume 25, Issue 1, January 2009, Pages 20-27, ISSN 0167-739X, 10.1016/j.future.2008.06.004.
- [45] Wei-Neng Chen; Jun Zhang; , "An Ant Colony Optimization Approach to a Grid Workflow Scheduling Problem With Various QoS Requirements," *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on* , vol.39, no.1, pp.29-43, Jan. 2009
- [46] Gosia Wrzesinska, Rob V. van Nieuwpoort, Jason Maassen, Henri E. Bal (2005). "Fault-Tolerance, Malleability and Migration for Divide-and-Conquer Applications on the Grid," *Proceedings of the 19<sup>th</sup> IEEE International Symposium on Parallel and Distributed Processing, 2005*, pp. 13a, 04-08 April 2005.
- [47] Buyya R, Murshed M (2002) GridSim: a toolkit for the modeling and simulation of distributed re-resource management and scheduling for grid computing. *Concurr Comput Pract Exp (CCPE)* 14(13–1175–1220).
- [48] Caminero, A.; Sulistio, A.; Caminero, B.; Carrion, C.; Buyya, R.; , "Extending GridSim with an architecture for failure detection," *International Conference on Parallel and Distributed Systems, 2007*, vol.2, no., pp.1-8, 5-7 Dec. 2007 doi: 10.1109/ICPADS.2007.4447756.
- [49] "Failure Trace Archive" [Online]. "<http://fta.inria.fr/apache2-default/pmwiki/index.php>".
- [50] Derrick Kondo, Bahman Javadi, Alexandru Iosup, and Dick Epema. 2010. The Failure Trace Archive: Enabling Comparative Analysis of Failures in Diverse Distributed Systems. In *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGRID '10)*. IEEE Computer Society, Washington, DC, USA, 398-407.
- [51] "Parallel Workload Archive" [Online]. "<http://www.cs.huji.ac.il/labs/parallel/workload/>".
- [52] "Grid Workload Archive" [Online]. "<http://gwa.ewi.tudelft.nl/pmwiki/>".
- [53] "MATLAB R2010a" [Online]. "[http://www.mathworks.in/help/techdoc/rn/br\\_03sl.html](http://www.mathworks.in/help/techdoc/rn/br_03sl.html)".

## **List of Publications**

[1] Upadhyay, N., and Misra, M. 2011. Incorporating fault tolerance in GA-based Scheduling in Grid environment. In *Proceedings of World Congress on Information and Communication Technologies* (Mumbai India , Dec 11 – 14). WICT 2011. IEEE. 776 – 781.