Fault Tolerance in Distributed Paradigms

Sajjad Haider 1, Naveed Riaz Ansari 2, Muhammad Akbar 1, Mohammad Raza Perwez 1, Khawaja MoyeezUllah Ghori1

1 National University of Modern Languages, Islamabad,
2 Shaheed Zulfiqar Ali Bhutto Institute of Science & Technology, Islamabad

Abstract. Distributed systems are responsible for providing the main execution platform for High Performance Computing (HPC). As distributed systems can be homogeneous (cluster) as well as heterogeneous (grid and cloud etc), they are prone to different kinds of problems. The issues in distributed systems can be Security, Quality of Service, Resource Selection and Fault Tolerance etc. Fault tolerance is responsible for handling the reliability and availability of distributed systems. It is not feasible to ignore job failures in distributed environments where long and persistent commitments of resources are required.

In this paper we have presented a comprehensive classification of errors, failures and faults that can be encountered in a Distributed environment. Furthermore, we have examined different fault identification and tolerance techniques available in different Clustered and Grid Computing environments. Fault detection and tolerance techniques used in homogeneous and heterogeneous environments are different from each other and are not interoperable. We have proposed in this paper that a standard fault tolerant framework should be there capable of handling all the identified errors, failures and faults.

Keywords: Fault Tolerance; Distributed Systems; Cluster, Grid

1. Introduction

Distributed systems are well known for achieving high performance in computing. We distribute compute intensive jobs into portions and send them to the machines for computations that are part of the distributed system. Nodes that are part of the distributed system execute their portion of the job and submit the results to job submission node. Distributed systems are further classified into Clusters and Grids. If we want to establish a reliable and available distributed environment then a fault-tolerant mechanism should be there. The incorporation of faults handling mechanisms in Clusters and Grids play an important role for that environment to be reliable and available. Fault tolerance is a capability developed in the system so that it could perform its function correctly even in the presence of faults. Taking fault tolerance into consideration would result in increasing the dependability of a system [1].

According to [1] failure is encountered when a system moves away from its particular behaviour. The reason behind that failure is called error that also ultimately depicts some sort of fault or defect in that system. This means that fault is the actual cause of a failure, and error is just an indication or sign of a fault. Multiple errors could be due to a fault, and even a single error could be the cause of multiple failures. In fault tolerance we try to preserve the delivery of expected services in the presence of faults that can cause errors. Errors are detected and corrected, and permanent faults are located and removed while the system continues to deliver acceptable service [2].

2. Existing Fault Tolerance Techniques

Many fault tolerance techniques such as retry, replication, message logging and check pointing [3] are available in traditional distributed paradigms.

2.1. Retry

Retry is the simplest failure recovery technique in which we hope that whatever is the cause of failures, the effect will not be encountered in subsequent retries [7].

2.2. Replication
In replication based technique we have replicas of a task running on different machines and as long as not all replicated tasks crash (i.e. host crash etc), chances are that the task execution would succeed [7].

### 2.3. Message Logging

In message logging all participating nodes log incoming messages to stable storage and when a failure is encountered than these message logs are used to compute a consistent global state. Algorithms that take this approach can be further classified into those that use pessimistic and those that use optimistic message logging [8]. Different types of errors, failures and faults that are likely to occur in distributed paradigms are presented in Figure 1.

![Extended classification of errors, failures and faults](image)

**Fig. 1: Extended classification of errors, failures and faults** [2, 3, and 4]

### 2.4. Check-pointing

Check-pointing is relatively more popular fault tolerant approach used in distributed systems, where the state of the application is stored periodically on reliable and stable storage, normally a hard disk etc. In case
of problem during execution, i.e. after crash etc, the application is restarted from the last checkpoint rather than from the beginning [9].

3. Literature Review

Globus offers a software infrastructure that enables applications to handle distributed heterogeneous computing resources as a single virtual machine [4]. The Globus Heart Beat Monitor [7, 10] uses a generic failure detection service that enables applications to detect both host or network failure through a process of heartbeat missing, e.g. ‘the task died without un-registering’ notification message.

LA-MPI [11, 12] is a type of implementation performed on the Message Passing Interface (MPI) [13, 14] motivated by the increasing need for fault tolerance at the software level in bigger High Performance Computing (HPC) scenarios. LA-MPI uses the retry/retransmission mechanism as the fault tolerant technique [15].

FT-MPI is another try of handling problems in MPI. Current fault tolerance and recovery strategies in FT-MPI [16] can regularly take checkpoints during a workflow step, that is normally a scientific application, and when a failure is encountered, the application is restarted from the last checkpoint [17].

Legion offers software based infrastructure for a system of heterogeneous and geographically distributed machines to interact with each other. Legion uses ‘pinging and timeout’ mechanism to detect task failures [5]. There is no way of detecting problems in Legion like the task crash failures, user-defined exceptions. Similarly Legion can’t distinguish the pure task crash failure from host/network failures [23].

Condor-G [18] uses an ‘ad hoc’ failure detection mechanism and uses ‘periodic polling to generic Grid server’ to detect some specific types of failures e.g. host and or network failures. Condor-G can’t detect task crash failures or user-defined exceptions, as is the case in Legion. Condor-G uses Retry on the same machine for fault tolerance in Grid environment [7].

NetSolve [3] is a client/server application designed to solve computational science problems in a distributed environment. The Netsolve system is composed of loosely coupled distributed systems, connected via LAN or WAN. Netsolve uses a generic heartbeat mechanism for failure detection and uses Retry on another available machine for fault tolerance [7].

Mentat uses polling for its failure detection and uses Replication as a fault tolerance mechanism [7].

CoG Kits [7] does not have failure detection mechanism and missing the advanced features of fault tolerance such as replication and check-pointing etc.

4. Critical Evaluation

For performing critical evaluation, we have compared different Grid Fault Tolerance implementations with each other and found some advantages and disadvantages of those techniques. Same technique was performed for Clustered based Fault Tolerant implementation scenarios.

4.1. Grid based Fault Tolerant implementations

Globus [5] uses the generic failure detection service that can be incorporated into a distributed system, tool or application. Faults detected by Globus are Host and Network failure. Globus can’t handle user defined exceptions. Similarly only application level fault tolerance is provided in this environment.

The features offered by Legion are transparent scheduling, data management, fault tolerance, site autonomy and security [11, 14]. Legion uses pinging and timeout technique for detecting faults and doesn’t have capability of detecting and handling task crash failures or host/network failures. Legion uses checkpoint recovery as fault tolerance technique [6].

NetSolve [3] is a programming and runtime system for accessing high-performance libraries and resources transparently. Netsolve uses a generic heartbeat mechanism for failure detection and uses Retry on another available machine for fault tolerance [9]. NetSolve uses generic heartbeat mechanism as its fault detection technique but doesn’t support diverse failure recovery mechanisms [7].
Mentat uses polling for its failure detection and uses Replication as a fault tolerance mechanism [7]. CoG Kits [7] does not have failure detection mechanism and missing the advanced features of fault tolerance such as replication and check-pointing etc.

TABLE I summarizes fault detection and tolerance techniques used in parallel and distributed systems.

<table>
<thead>
<tr>
<th>System</th>
<th>Type</th>
<th>Fault Detection Technique</th>
<th>Faults detected</th>
<th>Proactive / Reactive</th>
<th>Fault Tolerance Technique</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Globus [5]</td>
<td>Grid</td>
<td>Heart beat monitor</td>
<td>Host, Network failure</td>
<td>Reactive</td>
<td>Replicate</td>
<td>Can't handle user defined exceptions</td>
</tr>
<tr>
<td>MDS-2 [7]</td>
<td>Grid</td>
<td>GRRP</td>
<td>Task Crash failure</td>
<td>Reactive</td>
<td>Retry</td>
<td>Can't handle user defined exceptions</td>
</tr>
<tr>
<td>LAM/MPI [20]</td>
<td>Cluster</td>
<td>Node / Application stops responding</td>
<td>Node Failure</td>
<td>Reactive</td>
<td>Replication of checkpoints</td>
<td>Communication increases by replication checkpoints on several machines</td>
</tr>
<tr>
<td>Co-Check-MPI</td>
<td>Cluster</td>
<td>Application failure</td>
<td>Application Failure</td>
<td>Reactive</td>
<td>Check-pointing</td>
<td>Bigger application will take more time in check-pointing</td>
</tr>
<tr>
<td>Legion [23]</td>
<td>Grid</td>
<td>Pinging and Timeout</td>
<td>Task Failures</td>
<td>Reactive</td>
<td>Check-point recovery</td>
<td>Can't distinguish between task crash failure and host/network failure.</td>
</tr>
<tr>
<td>Condor-G [18]</td>
<td>Grid</td>
<td>Polling</td>
<td>Host Crash, Network Crash</td>
<td>Reactive</td>
<td>Retry on same machine</td>
<td>Use of Condor client interfaces on top of GLOBUS</td>
</tr>
<tr>
<td>NetSolve [7]</td>
<td>Grid</td>
<td>Generic Heart beat</td>
<td>Host crash, Network failure, k crash</td>
<td>Reactive</td>
<td>Retry on another available machine</td>
<td>Doesn't support diverse failure recovery mechanism</td>
</tr>
<tr>
<td>Mentat [7]</td>
<td>Grid</td>
<td>Polling</td>
<td>Host crash, Network Crash</td>
<td>Reactive</td>
<td>Replication</td>
<td>Exploiting task’s stateless and idempotent nature</td>
</tr>
<tr>
<td>Proactive FT in MPI [22]</td>
<td>Cluster</td>
<td>Process/node failure</td>
<td>Process/node stops responding</td>
<td>Proactive</td>
<td>Replication</td>
<td>Predictable failures</td>
</tr>
</tbody>
</table>

Table 1: Fault Detection and Tolerance Techniques used in Distributed Systems

4.2. Cluster based Fault Tolerant implementations

J. Hursey et al. in [19] has presented the design and implementation of an infrastructure to support checkpoint/restart fault tolerance in the Open MPI. The design uses an abstract interface for providing and accessing fault tolerance services without sacrificing performance, robustness, or flexibility. The drawback of implementation is that the design supports only some initial checkpoint/restart mechanisms.

C. Wang et al. in [20] have developed a transparent mechanism for job pause within LAM/MPI+BLCR. Fault detection technique discussed in the paper consists of a single timeout mechanism. Excessive delay in response from any process to a message request is assumed to indicate process (node) failure. So, it becomes a drawback that different types of failures are treated as node failure or simply failure.

LA-MPI [11, 12] is a network-fault-tolerant implementation of MPI designed for terascale clusters. The two important goals in LA-MPI are fault tolerance and performance. In LA-MPI process fault tolerance has completely been ignored and it only focuses on network relevant fault tolerance.

Co-Check MPI [21] developed by the Technical University of Munich was the first MPI implementation that used Condor library for implementation of fault tolerance using the technique of check pointing in MPI based applications. The drawback of this technique is that whole application needs check-pointing synchronously, and a bigger application would take more time resulting issues regarding performance.

S. Chakravorty et al. in [22] presents a fault tolerance solution for parallel applications that proactively migrates execution from processors where failure is imminent. The drawback in this idea is that it works on the assumption that failures are predictable.

All the fault tolerant techniques discussed with respect to Clustered computing environment focuses on reactive / post-active fault tolerance techniques, except the one discussed in [22].
5. Conclusion

According to our knowledge and survey, most of the distributed systems (Cluster and Grids) are not handling errors, failures and faults identified in Figure 1 and thus are not reliable. In order to achieve high availability a standard framework for fault tolerance should be the core of each distributed middleware, application or tool and should have the capability to implement that standard framework independently.

6. References
