ABSTRACT: Grid computing provides service of sharing data storage capacity and computer power over the Internet. Resource management scenarios often include resource discovery, resource monitoring, resource inventories, resource provisioning, fault isolation, variety of autonomic capabilities and service level management activities. Out of these scenarios, fault tolerance and Load Balancing are main research areas. In this paper First issue our main focus is on the development of fault tolerance system for computational grids and applies the Load Balancing. For this we had setup a computational grid based on the Alchemi middleware. In case of failure of the central manager, backup manager will take its control and avoids the grid to fail. This makes load balancing and fault tolerance more important in case of computing grid. This paper second issue introduces an algorithm which balances the load among the resources and also increases the reliability of the grid environment. The main goal of load balancing is to provide a distributed, low cost, scheme that balances the load across all the processors. Keywords: Grid computing, Grid Architecture, Load balancing, Fault Tolerance.

I. INTRODUCTION
Grid is a system that coordinates resources that are not subject to centralized control using standard, open, general-purpose interfaces and protocols to deliver non-trivial qualities of service. A grid is a type of parallel and distributed system that enables the allocation, selection and aggregation of resources distributed across multiple administrative domains based on their (resources) availability, capacity, performance, cost and quality of service requirements. A grid is a collection of machine sometimes referred to as nodes, resources, donors, members, clients, hosts, engines and many other such terms [2]. Grid computing is enabled by relatively high-performance computers, robust computer networks, grid management software, and the divisibility of difficult scientific problems. In Grid computing, individual users can access computers and data, transparently, without having to consider location, operating system, account administration, and other details. Grids tend to be more loosely coupled, heterogeneous, and geographically distributed. In Grid computing details are abstracted, and the resources are virtualized [1].

II. LITERATURE SURVEY
FAULT TOLERANCE:
As a result of the complex nature of heterogeneous networks, fault tolerance [5] is a major concern for the network administrators, and there are various ways that detection of such occurrences can be accomplished. When a fault occurs, it is important to: rapidly determine exactly where the fault is, Isolate the rest of the network from the failure so that it can continue to function without interference, Reconfigure or modify the network in such a way as to minimize the impact of operation without the failed component or components, and repair or replace the failed components to restore the network to its initial state.

A. Function of Fault Tolerance
The fault tolerance is "to preserve the delivery of expected services despite the presence of fault-caused errors[6] within the system itself. Errors are detected and corrected, and permanent faults are located and removed while the system continues to deliver acceptable service."

From a user's point of view, a distributed application should continue despite failures. The fault tolerance has become the main topic of research. Till now there is no single system that can be called as the complete system that will handle all the faults in grids. Grid is a dynamic system and the nodes can join and leave voluntarily. For making fault tolerance system a success, we must consider:

- How new nodes join the system,
- How computing resources are shared,
- How the resources are managed and distributed

B. Fault Tree Analysis
Because of computational Grid heterogeneity, scale and complexity, faults become likely. Therefore, Grid infrastructure must have mechanisms to deal with faults while also providing efficient and reliable services to its end users. The fault tree analysis[10] classifies faults that may take place in Grid Computing. In the figure various kinds of faults that can occur have been shown. There are mainly six classes of faults as discussed below:
LOAD BALANCING:
Load balancing is a technique to enhance resources, utilizing parallelism, exploiting throughput improvisation, and to cut response time through an appropriate distribution of the applications [9]. To minimize the decision time is one of the objectives for load balancing which has yet not been achieved. Job migration is the only efficient way to guarantee that submitted jobs are completed reliably and efficiently in case of process failure, processor failure, node crash, network failure, system performance degradation, communication delay; addition of new machines dynamically even though a resource failure occurs which changes the distributed environment [11].

As following Figure 2 load balancing feature can prove invaluable for handling occasional peak loads of activity in parts of a larger organization. These are important issues in Load Balancing:
An unexpected peak can be routed to relatively idle machines in the Grid.
If the Grid is already fully utilized, the lowest priority work being performed on the Grid can be temporarily suspended or even cancelled and performed again later to make room for the higher priority work.

III. PROBLEM STATEMENT
In this paper First issue our main focus is on the development of fault tolerance system for computational grids. For this we had setup a computational grid based on the Alchemi middleware. Alchemi is a .NET-based grid computing framework that provides the Runtime machinery and programming environment required to construct computational grid. In case of failure of the central manager, backup manager will take its control and avoids the grid to fail. In grid computing the resources are shared and dynamic in nature, which affects application performance. To improve the global throughput of these environments, effective and efficient load balancing algorithms are fundamentally important. Load Balancing is one of the most important factors which can affect the performance of the grid application. Main purpose of load balancing is to analyze problems due to which load balancing is required in grid computing and to transfer heavy loaded nodes to lightly loaded nodes based on job migration policy.

IV. PROPOSED WORK
PROPOSING THE BACKUP MANAGER
While working with the setup grid, the failure of the centralized manager causes the whole grid to hangs down. Till the manager doesn’t restart, the grid remains inaccessible. So to avoid such kind of grid failure, we have introduced the concept of the backup manager. The backup manager also uses the heart beat phenomenon to query the status of manager. The steps for implementing the backup manager are:
After every heartbeat interval, the leader node sends a packet to the backup manager. Figure 3 shows the Backup manager concept. Packet received by the backup manager contains information about each of the nodes in the group and its arrival indicates that the leader is up and running. The backup leader updates its database using the data obtained from the received packet. In case of absence of packet for certain predefined time interval lets the backup manager to consider the master has failed, and it itself takes the control as new master. This change is multicast to the executors and they update their connections database in order to accept Backup Manager as their new master. Figure 3 shows the new connection established between the executors and the backup manager. We try to use these steps for providing the support for backup manager for Alchemi based computational grids. From there the backup manager accesses the database to control the grid.

Figure 1 Type of Faults

Figure 2 Job Migration [5]

Figure 3: Backup Manager Concept
LOAD BALANCING

In this paper we propose a dynamic load balancing algorithm for improving performance of grid computing. Here we mention are four steps: load monitoring, synchronization, rebalancing criteria job migration. In proposed load balancing algorithm the activities can be categorized as following:

Arrival of any new job and queuing of that job to any particular node, Completion of execution of any job, Arrival of any new resource, Withdrawal of any existing resource.

Code for to algorithm: Function: LoadBalancing_start
Return Type: Boolean
Start:
If (CPU Idle and Free Memory of Node is Min and Queue Length is Max)
HeavilyLoaded_Node
HeavilyLoaded_list= HeavilyLoaded_list + 1 (new selected node);
End if
If (CPU Idle and Free Memory of Node is Max and Queue Length is Min)
LightlyLoaded_Node
End if
Migrate Heavy Loaded_Node_Job to Lightly Loaded_Node
End

LoadBalancing_start (): this function also return Boolean value. If on the basis of given parameters (CPU utilization and queue length) load balancing will be required it will return true else it will return false. This function also updates two lists: HeavilyLoaded_list and LightlyLoaded_list.

Main difference between existing Load Balancing algorithm and proposed Load Balancing is in implementation of three policies: Information Policy, Triggering Policy and Selection Policy. For implementation of Information Policy all existing Load Balancing algorithm use periodic approach, which is time consuming. The proposed approach uses activity based approach for implementing Information policy. For Triggering Load Balancing proposed algorithm uses two parameters which decide Load Index. On the basis of Load Index Load Balancer decide to activate Load Balancing process. For implementation of Selection Policy Proposed algorithm uses Job length as a parameter, which can be used more reliably to make decision about selection of job for migration from heavily loaded node to lightly loaded node.

V. IMPLEMENTATION STRATEGY AND RESULTS

Create Replication Database

Figure 5 Screenshot of Running Application
Implementation of Backup manager

Figure 6 Screenshot of Backup Manager

Run Application

Figure 7 Prime Number Application

Figure 8 manager fail and connect to backup manager

Experiment results

<table>
<thead>
<tr>
<th>Input Range to find out prime number</th>
<th>Total No of Executor Running</th>
<th>Total Time to Complete execution</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(In Second)(BM Application)</td>
<td>(In Minute)(Existing BM Application)</td>
</tr>
<tr>
<td>Prime of 1000000</td>
<td>2</td>
<td>00:04:0872</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>00:04:0404</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>00:04:4460</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>00:04:5240</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>00:09:6096</td>
</tr>
<tr>
<td>Prime of 10000000</td>
<td>2</td>
<td>00:04:0900</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>00:04:0600</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>00:04:6240</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>00:13:8060</td>
</tr>
</tbody>
</table>

Table 1 comparison of Different Number of Executor with Different input range

VI. CONCLUSIONS AND FUTURE ENHANCEMENTS

In this research I analyzed existing Load Balancing algorithm and proposed an enhanced algorithm which more efficiently implements three out of five policies implemented in existing Load Balancing algorithm. I studied the problems and challenges included with faults in computational grid and proposed algorithm use for finding the various kind of faults based on Alchemi. It also calculated the efficiency of our proposed system under various situations. Future work includes, find optimal approach for better performance of applications running in grid. And also we will try to achieve more efficiency with new police for load balancing. Furthermore, the proposed algorithm can also be replaced with the existing work to improve the performance of grid.
REFERENCES