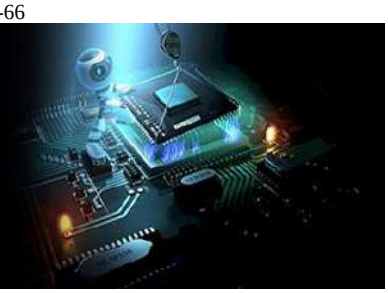
International Journal of Engineering in Computer Science



E-ISSN: 2663-3590
P-ISSN: 2663-3582
IJECS 2019; 1(2): 59-66
Received: 24-05-2019
Accepted: 25-06-2019

Neeraj Rathore
Assistant Professor,
Shri G.S. Institute of Technology & Science, Indore, Madhya Pradesh, India.

Jyoti Rathore
PhD Scholar, Jaypee University of engineering & technology, Guna, Madhya Pradesh, India.

Correspondence
Neeraj Rathore
Assistant Professor, Shri G.S. institute of technology & science, Indore, Madhya Pradesh, India.

Efficient checkpoint algorithm for distributed system

Neeraj Rathore and Jyoti Rathore

DOI: https://doi.org/10.33545/26633582.2019.v1.i2a.22

Abstract
The Grid is rapidly emerging as the means for coordinated resource sharing and problem solving in multi-institutional virtual organizations while providing dependable, consistent, pervasive access to global resources. The emergence of computational Grids and the potential for seamless aggregation and interactions between distributed services and resources, has led to the start of new era of computing. Tremendously large number and the heterogeneous nature of Grid Computing resource make the resource management a significantly challenging job. 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 this fault tolerance has become the main topic of research as till date there is no single system that can be called as the complete system that will handle all the faults in grids. Checkpointing is one of the fault-tolerant techniques to restore faults and to restart job fast. The algorithms for checkpointing on distributed systems have been under study for years. These algorithms can be classified into three classes: coordinated, uncoordinated and communication-induced algorithms. In this paper, a checkpointing algorithm that has minimum checkpointing counts equivalent to periodic checkpointing algorithm has been proposed. For relatively short rollback distance at faulty situations and produces better performance rather than other algorithms in terms of task completion time, in both fault-free and faulty situations. This algorithm has been implemented in Alchemi.NET because it did not currently support any fault tolerance mechanism.

Keywords: Checkpointing, Fault-Tolerance, Alchemi.NET, Grid Computing, GridSim

1. Introduction
Grid is a type of distributed system that supports the sharing and coordinated use of geographically distributed and multi- owner resources independently from their physical type and location in dynamic virtual organizations that share the same goal of solving large-scale applications.

2. Literature survey
The last decade has seen a considerable increase in commodity computer and network performance, mainly as a result of faster hardware and more sophisticated software. The early efforts in Grid Computing started as a project to link supercomputing sites, but now it has grown far beyond its original intent. In fact, there are many applications that can benefit from the grid infrastructure, including collaborative engineering, data exploration, high throughput computing, and of course distributed supercomputing.

2.1 Comparison between different checkpoint schemes
Based on the literature survey following comparisons have been made:
The comparison between disk based and disk less checkpointing for distributed and parallel system in certain parameter is described in Table 1.

Table 1: On disk and disk less checkpointing for parallel and distributed system [22-40]

Parameter	Disk Based	Diskless
Latency time	High	Low
CPU overhead	High	High
Memory requirement	Low	high
Stable storage requirement	High	Low
Toleration of wholesale failure	Yes	No
Reliability	High	Low
Efficiency	Low	High
Addition hardware	Not Required	Additional processors
Portability	High	Low

The comparison between Disk-based and Memory-based Checkpoint in certain parameter is described in Table 2.

Table 2: Comparison of Disk-based and Memory-based Checkpoint Schemes [22-42]

Fault tolerant protocols	Double in Memory	Double in Disk
Shrink/Expand	Yes	Yes
Portability	Low	Low
Foolproof	No	No
Diskless	Yes	No, local disk
Halts job	No	No
Bottleneck	No	No
Require backup processors	Not Necessarily	Not Necessarily
Transparent checkpoint	No	No
Synchronized checkpoint	Yes	Yes
Automatic restart	Yes	Yes

Table 3: Comparative studies of different checkpointing schemes [10-42]

Check pointing Methods	Uncoordinated Check pointing	Coordinated Check pointing	Communication Induced	Diskless check pointing	Double Check pointing
Efficiency	High for small process	Low	Low	High	High
Performance	Low	Low	Low	Higher for distributed applications	Faster
Portability	High	High	High	Low	Low
Cost	High	Low, negligible for low memory usage application	High	High	Very High
Scalable	No	Minimal	Not scale for large number of processors	Difficult to scale to large number of processors	Highly
Flexibility	Low	All processes Save their states at the same time	Process can be moved from one node to another by writing the process image directly to a remote node	Replace stable storage with memory and processor redundancy	Handle fault at a time and Availability of one checkpoint in case the other is lost.
Overhead	Large Storage, Very high Log management and Work in small Process	Minimum storage overhead And negligible overheads in failure-free Executions.	High latency and Memory and disk overhead	High memory overhead for storing checkpoints	Low memory overhead
Advantages	Most Convenient and Save their checkpoints Individually	Not suffer from rollback propagations and Processes Save their state together	Preventing domino effect, piggybacking And information Of regular message exchanged by the processes	Improve performance in distributed /parallel applications and Process migration save process image	Uses in small memory footprint on large number of processors. ex- scientific applications
Recovery	Checkpoint Of the faulty process is restored	Processes stop regular message activity to take their checkpoints and coordinated way to analyze and restore the last set of Checkpoints	Needed large number of forced checkpoints nullify the benefit of autonomous local checkpoints Using new process to restore the process image after failure	Using parity/backup and extra processors for storing parity as well as replace failed application processors.	Through automatic restart and synchronization by two identical checkpoint buddy processors to provide foolproof fault tolerance
Disadvantages	Unsuitable, Domino Effect, Wastage memory, unbounded & complex Garbage collection	Consistent checkpoint and Large latency for saving the checkpoints storage	Deteriorated parallel performance & Requires standby processors	Communication bottleneck	Depend on a central reliable storage and required Additional Hardware

On the comparison basis a better checkpoint technique in different grid environment has been discussed.

3. Proposed Checkpointing Algorithm

Checkpointing schemes associate each local checkpoint with a checkpoint sequence number and try to enforce consistency among local checkpoints with the same sequence number. This checkpointing schemes have the easiness, recovery time advantages and low overhead over other checkpointing schemes. Proposed checkpointing algorithm reduces the checkpoint overhead compared to previously suggested checkpointing algorithms and which have a relatively short rollback distance in faulty situation. In the proposed scheme, we use checkpoint sequence number to take a message checkpoint. However, unlike the other algorithms, only one message checkpoint can exist

between two consecutive periodic checkpoints. Therefore, our checkpointing algorithm has smaller number of checkpoints than other checkpointing algorithms. Moreover, like the other algorithms, the dependency among checkpoints is removed by message checkpoints. In result, our checkpointing algorithm has a relatively short rollback distance in faulty situation. The algorithm has a better performance than the others in terms of task completion time in both of fault free and faulty situations. New checkpointing algorithm is discussed in next section.

3.1 Checkpointing Algorithm

Step 1 The algorithm runs for n processes where the process value P(i) is varies from 0 to n-1.
Step 2 The flag value is initialized. If new checkpoint flag value is (CFV) =0 (False) that means there is no checkpoint flag in the current interval and if the value is CFV =1 (True) that means there is a checkpoint flag in the current interval.
Step 3 After assigning the value of the process, start with while loop when process P(i) till the end.
Step 4 In the checkpoint, if the current time value of the process equals to the checkpoint time than check
If (CFV==0)
Take a stable checkpoint with current process state; Increase checkpoint number by 1;
Checkpoint increase by current time plus checkpoint and Set new checkpoint flag = false;
Else, if condition does not match take a stable checkpoint with new checkpoint
Step 5 After new checkpoint flag is checked, process P (i) send and receive the message and check If senders checkpoint number is greater than current checkpoint number than again check
if (CFV==1)
Set current checkpoint number to senders;
else
Take a new checkpoint with current process state;
Set current checkpoint number to senders; And Set new checkpoint flag = True;
Step 6 At last, process received a message.
Step 7 If any failure occurs in between the execution, after resuming value pick from the memory and go to step 3.

3.2 Complexity Analysis of the Proposed Algorithm

The Complexity of the above algorithm is O(n). Because n processes run n time within while loop. For each process P(i), the if-else statements within the while loop gets executed. Assuming that first if-else loop runs j times, where j is a constant when the current time equals to the checkpoint time. After that the process P(i) has to process the received message if sender checkpoint number is greater than the current checkpoint. The if-else construct runs k times, where k is a constant and since the algorithm is running for the n processes so the total complexity of the algorithm is O(j+k)*n. which is approximately equal to the O(n). The O(n) complexity of the algorithm is better to implement any type of checkpointing algorithm in real time environment. For the requirement for the implementation of the new framework is given below.

3.3 Executor Algorithm

1. Start the Executor and import all the threads like System. Net. Sockets, System. Threading, and System.IO.
2. Initialize variables of Binary Reader and Writer, thread, Tcp Client Network Stream.
3. Start the Executor thread. Clint_thrd1. Start ()
4. Assign port number to socket and wait for the connection on the basis of same port number and connecting Manager IP Address.
   tcp_client.Connect ("local host", 45)
5. if (new_connection connected=True) then
   Connection established and sends Executor detail to Manager
   Else Connection Fail. Again check for new connection
   End if
6. After connection establish data Sending or receiving through the stream writer and stream reader and display the result through the message box.
   reader = New Binary Reader (net_stream)
   writer = New Binary Writer (net_stream)
7. All the communication data is parallel stored in the temporary file and display the result at both the Manager and the Executor site or after a specific time of checkpoint data will save at the permanent storage
   If (e.Key Code = Keys. Enter) Then writer. Write (inputtxt.Text)
   output txt.Text &= "Executor side:-" & input txt. Text & vbCr
   RichTextBox2.Text += vbCrLf & "Executor side:-" & inputtxt.Text & vbCrLf
   End If
8. If want to store data in the log file then saved data otherwise truncate data and terminate theconnection.RichTextBox1.
   Load File("d:\file", Rich Text Box Stream Type.Plain Text)
9. We can see the hole save data record through the database summery record. Databse_summery.Show ()
10. After this if u want to new entry connection then go to entry detail otherwise abort the connection.
    tcp_client.Close () and clint_thrd1.Abort ()

3.4 Executor Algorithm

1. Start the Manager and import System.Net.Sockets, System.Threading, System.IO threads.
2. Initialize variables of BinaryReader, BinaryWriter, thread, sockets and new_connection, Network Stream, Net.Sockets.TcpListener.
3. Run the Executor ser_thrd1.Start () thread.
4. Create tcp listener on the basis of assign port number to socket and wait for the connection to new Executor.
   tcp_listn = New Net.Sockets.TcpListener (System.Net.IPAddress.Any, 45)
   tcp_listn.Start ()
5. Establish new connection between Manager and Executor through TCP listener socket and
   If (new_connection.connected=True) then
   Connection established
   Else
   End if
   Connection Fail. Again check for new connection
6. After establish the connection registered the Executor into the Executor table in Alchemi.NET.NET database and start communication otherwise abort the connection.
7. Sending and receiving (read/write) data through the stream writer and stream reader and after the connection establish display the result through the message box.
   writer = New BinaryWriter (net_stream)
   reader = New BinaryReader (net_stream) outputtxt.Text &= "Connection Established" & vbCr Dim recive_data As String
   Do
   recive_data = reader.ReadString
   outputtxt.Text &= "Executor side:-" & recive_data & vbCr
   Loop While (recive_data <> "terminate")
8. All the communication data is parallel stored in the temporary file and display the result at both the Manager and the Executor site or after a specific time of checkpoint data will automatically saved in the permanent storage
   If (e.KeyCode = Keys.Enter) Then writer. Write (inputtxt.Text)
   outputtxt.Text &= "Manager Side:-" & inputtxt.Text & vbCr
   End If
9. If want to store data in the log file then saved data otherwise truncate data and terminate the connection.
10. After this if u want to new entry connection then go to entry detail otherwise abort the connection.
    tcp_listn.Stop ()
    ser_thrd1.Abort ()

In this section, we have proposed the new communication induced checkpointing algorithm which is use to make Alchemi.NET fault tolerant. We have also described the need of hardware and software requirement in the implementation, programming language which is use for our implementation work. After that we have discussed the proposed and designed framework and described the algorithm and flowchart, which is done, on the Alchemi.NET middleware.

4. Experimental results

This section describes the implementation of Checkpointing algorithm, in which data can be stored in permanent log file on the basis of checkpointing, which has been proposed in the previous section. An application developed in VB.Net, C# (front end) and SQL SERVER (back end database). The results are shown in the form of screen shots.

Alchemi.NET grid can be viewed as a virtual machine with multiple processors. A grid Application can take advantage of this by creating independent units of work to be executed in parallel on the grid (each unit of work is executed by a particular Executor).

These units of work are called grid threads and must be instances of a class that is derived from Alchemi.NET.Core.Owner, GThread. Code that is to be executed on the grid is defined in this class's void Start () method.

Follow these steps to set up a development environment:

- Construct a minimal grid (1 Manager and 1 Executor) on the development and test it by running Application.
- Download the Alchemi.NET SDK and extract to a convenient location
- Locate Alchemi.NETCore.dll for referencing in applications.

Checkpointing is implemented using the below mentioned steps:

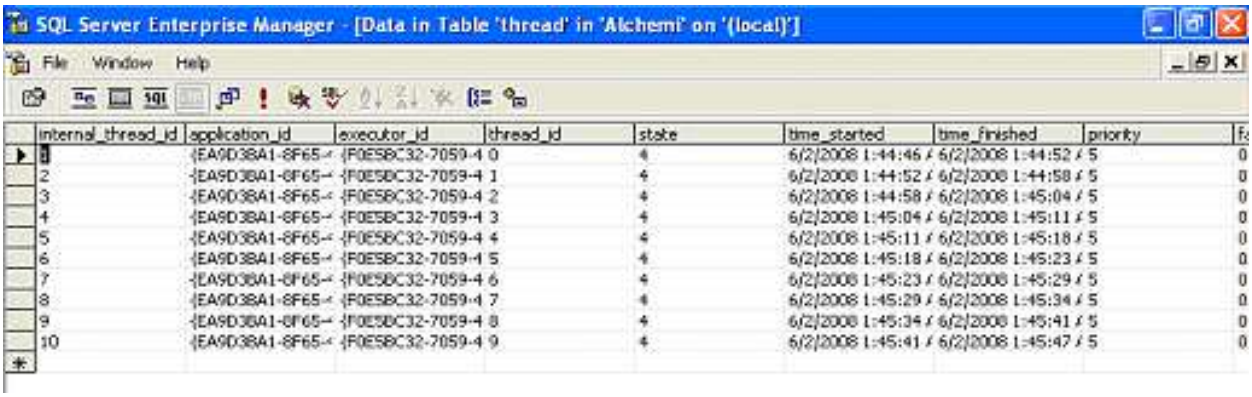
Step1: In this kind of application first check the status of the Manager then, start application to scan port 9001 of the remote system where Manager is running at regular interval of time.

The code for this application is written in c#. The two main classes used for this are:

'Socket' and 'IP End Point'. User has to provide the IP address of the system where Manager is running. In any such case when the Manager fails due to some reason, data is saved on the log tables. After recovering, it can again start from the checkpoint created.

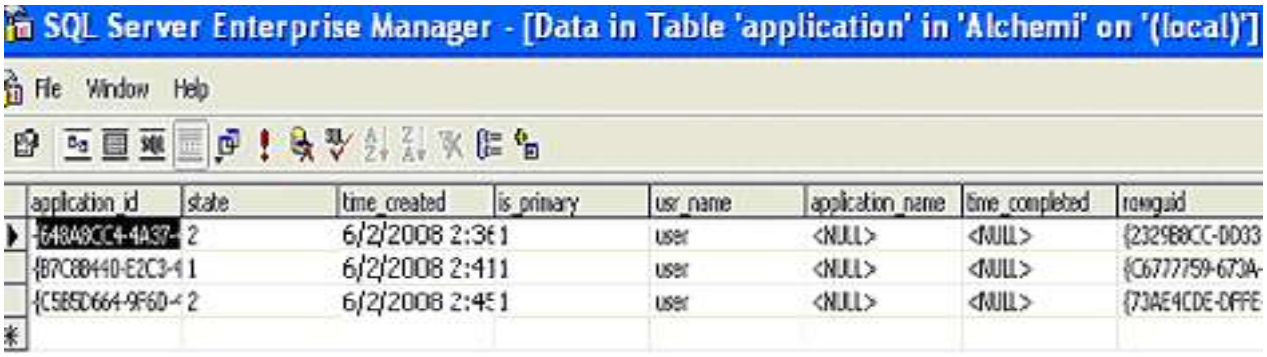
Step2: Creating the checkpoint Database Checkpoint is the process of stored data in regular time intervals from memory to permanent storage databases. Using checkpoint, small copies of data stored by us into the database can be shared by both Manager and Executor. The previous data records are safely stored in log file, which is present on the executor site. The database structure is replicated in the following table. These tables maintain all the information regarding executor, application and thread etc. that is presented in the screen shots below.

Table 4: Detail of running threads



internal_thread_id	application_id	executor_id	thread_id	state	time_started	time_finished	priority
1	{EA9D3BA1-8F65-…	{F0E5BC32-7059-4…	0	4	6/2/2008 1:44:46 …	6/2/2008 1:44:52 …	5
2	{EA9D3BA1-8F65-…	{F0E5BC32-7059-4…	1	4	6/2/2008 1:44:52 …	6/2/2008 1:44:58 …	5
3	{EA9D3BA1-8F65-…	{F0E5BC32-7059-4…	2	4	6/2/2008 1:44:58 …	6/2/2008 1:45:04 …	5
4	{EA9D3BA1-8F65-…	{F0E5BC32-7059-4…	3	4	6/2/2008 1:45:04 …	6/2/2008 1:45:11 …	5
5	{EA9D3BA1-8F65-…	{F0E5BC32-7059-4…	4	4	6/2/2008 1:45:11 …	6/2/2008 1:45:18 …	5
6	{EA9D3BA1-8F65-…	{F0E5BC32-7059-4…	5	4	6/2/2008 1:45:18 …	6/2/2008 1:45:23 …	5
7	{EA9D3BA1-8F65-…	{F0E5BC32-7059-4…	6	4	6/2/2008 1:45:23 …	6/2/2008 1:45:29 …	5
8	{EA9D3BA1-8F65-…	{F0E5BC32-7059-4…	7	4	6/2/2008 1:45:29 …	6/2/2008 1:45:34 …	5
9	{EA9D3BA1-8F65-…	{F0E5BC32-7059-4…	8	4	6/2/2008 1:45:34 …	6/2/2008 1:45:41 …	5
10	{EA9D3BA1-8F65-…	{F0E5BC32-7059-4…	9	4	6/2/2008 1:45:41 …	6/2/2008 1:45:47 …	5

Table 5: List of running application



application_id	state	time_created	is_primary	usr_name	application_name	time_completed	rowguid
{648A0CC4-4A37-…	2	6/2/2008 2:3…	1	user	<NULL>	<NULL>	{2329B8CC-0D33
{B7C8B440-E2C3-…	1	6/2/2008 2:41…	1	user	<NULL>	<NULL>	{C6777759-673A-
{C585D664-9F60-…	2	6/2/2008 2:4…	1	user	<NULL>	<NULL>	{73AE4CDE-0FFE

- The interface of the Alchemi.NET executor has three button first connect button for establish the connection between Manager and the executor, second disconnect button for terminate the connection between them and the last one database summery for shown the all communication which was held between both. It can show the permanent log file data that are stored in the disk
- The interface of the Alchemi.NET Manager three buttons first is the Start Manager button, for starting the Manager. Second is the stop button for terminating the connection between them and third button for displaying new form in which we can enter the executor details to store in the executor table. It also has two rich text box, one for input query in which we can communicate to executor and other one shown is the resultant summary which shows all communication which was held between both. It saves data on temporary basis which is in the memory.
- The Alchemi.NET Manager and Executor have started and they are further waiting for the heartbeat signal for establishing the connection with each other for further processing.
- The established connection at the Manager and further acknowledge should be displayed on the Executor site.
- The entry form of the particular executor is currently connected to the Manager. It has all the entries filled according to executor requirement. There are two buttons, one for submitting the entry in the table and other for the clearing entry if any field is filled incorrectly. After filling the entry in executor form all the data is stored in the executor database table in the Alchemi.NET database.
- The successful record added in the executor table. This is permanent storage of data entry.
- Messages sent between Manager and executor like First send the message from the input query and temporary store all data shown at output result.
- The message box yes for saving all the communication in permanent log file and No for truncating data between Manager and executor.
- The database summary forms in which we can see the entire log file data at Manager Site, which is permanently stored.
- The message box, yes for terminating the connection at the Manager site and No for further processing.
- The message box after clicking yes in previous form and after pressing OK on the message box, above connection is aborted.

Fig 1: Shows Four Executors running while executing application

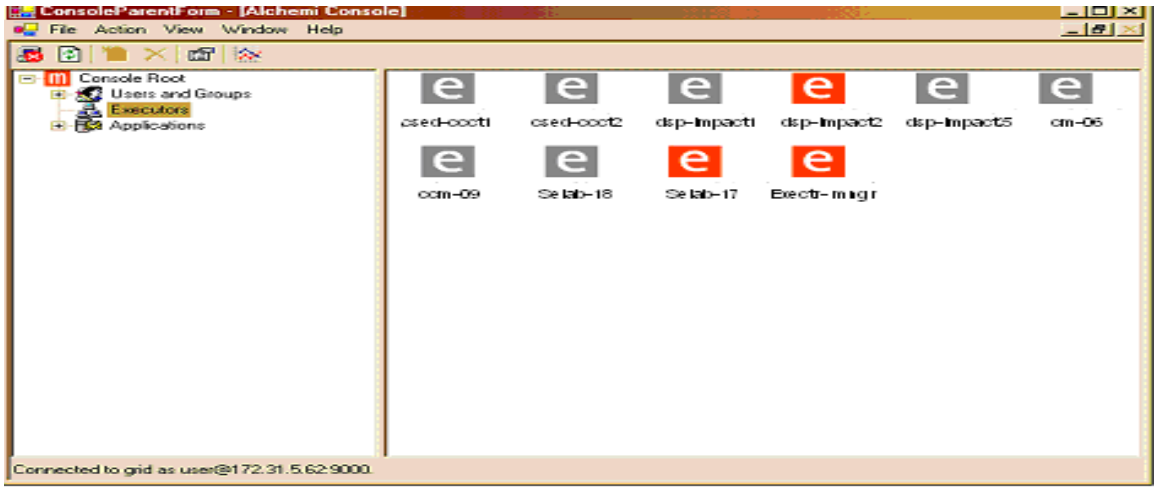
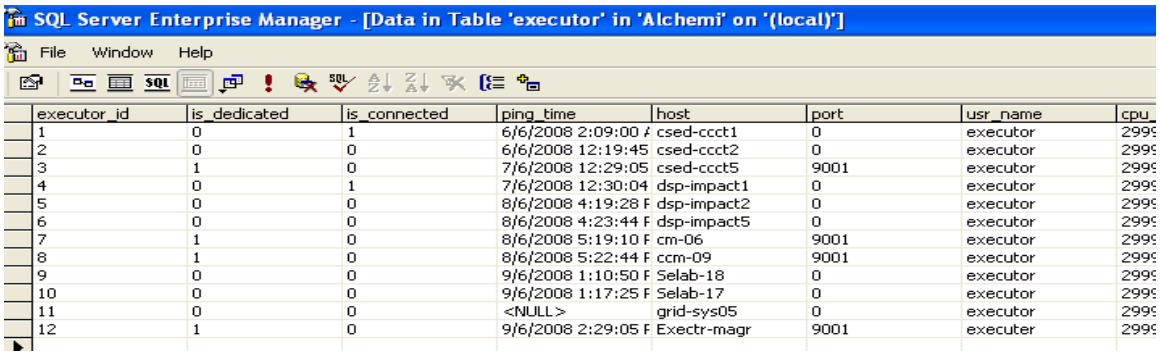


Fig 2: One-Executor stops while the application is running

Above Figures 1 and 2 shows the on line Executor who are connected to the Manager while running the application. Some of the Executor who is red in the figure is ideal and ready to execute the job/application and other gray in color Executor shows that they are not ideal at the time of any application running.

Table 6: Executor table after storing executor data



executor_id	is_dedicated	is_connected	ping_time	host	port	usr_name	cpu_
1	0	1	6/6/2008 2:09:00 A	csed-ccct1	0	executor	2999
2	0	0	6/6/2008 12:19:45	csed-ccct2	0	executor	2999
3	1	0	7/6/2008 12:29:05	csed-ccct5	9001	executor	2999
4	0	1	7/6/2008 12:30:04	dsp-impact1	0	executor	2999
5	0	0	8/6/2008 4:19:28 P	dsp-impact2	0	executor	2999
6	0	0	8/6/2008 4:23:44 P	dsp-impact5	0	executor	2999
7	1	0	8/6/2008 5:19:10 P	cm-06	9001	executor	2999
8	1	0	8/6/2008 5:22:44 P	ccm-09	9001	executor	2999
9	0	0	9/6/2008 1:10:50 P	Selab-18	0	executor	2999
10	0	0	9/6/2008 1:17:25 P	Selab-17	0	executor	2999
11	0	0	<NULL>	grid-sys05	0	executor	2999
12	1	0	9/6/2008 2:29:05 P	Exectr-magr	9001	executer	2999

Table 6 shows the entire executor list, which were connected to the Manager. The table has all the record about the executor with the entire field that is used by the executor.

This section deals with the implementation part of Paper work. After giving the basic requirements for implementation, we have proposed the system that deals with the problems found in Alchemi.NET.NET based computational grids has been proposed. The system is implemented using C# as programming language and MS SQL Server as Database. Considering different test cases the experimental evaluation was also done for the implemented system. In the next section concludes our Paper work and gives future scope of it.

5. Conclusion and future work

The sharing of computational resources is a main motivation for constructing Computational Grids in which multiple

computers are connected by a communication network. Due to the instability of grids, the fault detection and fault recovery is a very critical task that must be addressed. The need for fault tolerance increases as the number of processors and the duration of computation increases. In this Paper, we investigated the issues and challenges involved in dealing with faults in Alchemi.NET middleware have been investigated and the checkpoint algorithm in Alchemi.NET for dealing with various kinds of faults has been implemented. Further the efficiency of our proposed system under various conditions has been evaluated.

In a heterogeneous computing environment like Grid, a suite of different machines ranging from personal computer to supercomputer is loosely inter-connected to provide a variety of computational capabilities to execute collections of application tasks that have diverse requirements. These kinds of large scale high performance distributed services have recently received substantial interest from both research as well as industrial point of view. However, an important research problem for such systems is the lack of fault tolerance system. The heterogeneous and dynamic nature of grids makes them more prone to failures than traditional computational platforms. So, the system managing such infrastructures needs to be smart and efficient to overcome the challenge of fault detection and recovery.

5.1 Main Contributions

- Studied the Fault Tolerance in different grid middleware's and found most common kind of failures that can let the Alchemi.NET grid to work improperly.
- Alchemi.NET based grid has been set up in the departments.
- Various existing fault tolerance mechanisms already present in the setup grid are identified. After studying the internals of Alchemi.NET, different shortcomings that are there in Alchemi.NET for handling the faults dynamically has been identified.
- On the basis of identified shortcomings, proposed and updated fault tolerance in Alchemi.NET middleware with the help of checkpointing algorithm.
- Implemented checkpoint concept that will help in avoiding the grid to become inaccessible if the Manager or Executor fails.

6. References

1. N Jain, N Rathore, A Mishra. An Efficient image forgery detection using orthogonal wavelet transform and improved relevance vector machine, wireless personal communication, springer Publication-New-York (USA) IF-1.200. 2018; 101(4):1983-2008.
2. N Rathore. Performance of hybrid load balancing algorithm in distributed web server system, wireless personal communication, springer Publication-New-York (USA) IF-1.200. 2018; 101(4):1233-1246.
3. N Jain, N Rathore, A Mishra. An efficient image forgery detection using biorthogonal wavelet transform and improved relevance vector machine with some attacks, Interciencia Journal. 2017; 42(11):95-120.
4. N Rathore. Dynamic threshold based load balancing algorithms, wireless personal communication, springer Publication-New-York (USA), 2016; 91(1):151-185.
5. N Rathore, I Chana. Job migration policies for grid environment, wireless personal communication, springer Publication-New-York (USA) IF -0.979. 2016; 89(1):241-269.
6. Rathore, Neeraj, Inderveer Chana. Variable threshold-based hierarchical load balancing technique in Grid, engineering with computers. 2015; 31.3:597-615.
7. Vishal Sharma, Rajesh Kumar, Neeraj Kumar Rathore. Topological broadcasting using parameter sensitivity based logical proximity graphs in coordinated ground-flying ad hoc networks, Journal of wireless mobile networks ubiquitous computing and dependable applications (JoWUA), Scopus indexed. 2015; 6(3):54-72.
8. Rathore, Neeraj, Inderveer Chana. Load balancing and job migration techniques in grid: a survey of recent trends, wireless personal communications. 2014; 79.3:2089-2125.
9. Rathore, Neeraj, Inderveer Chana. Job migration with fault tolerance based QoS scheduling using hash table functionality in social Grid computing, Journal of intelligent & fuzzy systems. 2014; 27.6:2821-2833.
10. Neeraj Kumar Rathore, Farah Khan. Internet of things: A review article, Journal of cloud computing (JCC), ISSN Print: 2349-6835, ISSN Online: 2350-1308, IF=0.333. 2018; 5(1):20-25.
11. Neeraj Kumar Rathore, Farah Khan. Survey of IoT, Journal of cloud computing (JCC), ManTech Publication. 2018; 1(1):1-13.
12. Neeraj Kumar Rathore, Pramod Kumar Singh. A comparative analysis of fuzzy based load balancing algorithm, Journal of computer science (JCS). 2017; 5(2):23-33.
13. Neeraj Kumar Rathore, Harikesh Singh. Analysis of grid simulators architecture, Journal on mobile applications and technologies (JMT). 2017; 4(2):32-41.
14. Neeraj Kumar Rathore. Checkpointing: fault tolerance mechanism, Journal on cloud computing (JCC). 2016; 3(4):27-34.
15. Neeraj Kumar Rathore. A review towards: Load balancing techniques, Journal on power systems engineering (JPS). 2017; 4(4):47-60.
16. Neeraj Kumar Rathore. Faults in Grid, International Journal of software and computer science engineering, Mantech Publiations. 2016; 1(1):1-19.
17. Neeraj Kumar Rathore. Installation of Alchemi NET in computational grid, Journal on computer science (JCOM). 2016; 4(2):1-5.
18. Neeraj Kumar Rathore. Ethical hacking & security against cybercrime, Journal on information technology (JIT). 2016; 5(1):7-11.
19. Neeraj Kumar Rathore. Efficient agent based priority scheduling and load balancing using fuzzy logic in grid computing, Journal on computer science (JCOM). 2015; 3(3):11-22.
20. Neeraj Kumar Rathore. Map reduce architecture for grid, Journal on software engineering (JSE). 2015; 10(1):21-30.
21. Neeraj Kumar Rathore. GridSim installation and implementation process, Journal on cloud computing (JCC). 2015; 2(4):29-40.
22. Neeraj Kumar Rathore, Inderveer Chana. Report on hierarchal load balancing technique in grid environment, Journal on information technology (JIT), ISSN Print: 2277-5110. 2013; 2(4):21-35.
23. Neeraj Kumar Rathore, Inderveer Chana.

Checkpointing algorithm in alchemi.NET, Pragyaan: Journal of information technology, IMS Dehradun, ISSN No: 0974-5513, IEEE, CSI and MPCET Dehradun. 2010; 8(1):32-38.
24. Neelesh Jain, Neeraj Kumar Rathore, Amit Mishra. An Efficient image forgery detection using biorthogonal wavelet transform and singular value decomposition in 5th International conference on advance research applied science, environment, agriculture & entrepreneurship development (ARASEAED), Bhopal organized & sponsored by Janparishad, JMBVSS & International council of people at Bhopal (M.P.) India, held on 04-06, 2017, 274-281.
25. Neeraj Kumar Rathore, I Chana. A sender initiate based hierarchical load balancing technique for grid using variable threshold value in International conference IEEE-ISPC, ISBN- 978-1-4673-6188-0. 2013; 1-6:26-28.
26. Neeraj Kumar Rathore, I Chana. A Cognitative analysis of load balancing technique with job migration in grid environment, world congress on information and communication technology (WICT), Mumbai, IEEE proceedings paper, ISBN-978-1-4673-0127-5, 2011, 77-82.
27. Neeraj Kumar Rathore. Efficient load balancing algorithm in grid in 30th M.P young scientist congress, Bhopal, M.P, 2015, 56.
28. Neeraj Kumar Rathore. Efficient hierarchical load balancing technique based on grid in 29th M.P young scientist congress, Bhopal, M.P, 2014, 55, Feb 28.
29. Rohini Chouhan, Neeraj Kumar Rathore. Comparision of load balancing technique in grid, 17th Annual conference of Gwalior Acadmy of mathematical science and Natonal symposium on computational Mathamatics & Information Technology, JUET, Guna, M.P, 2012, 7-9.
30. Neeraj Kumar Rathore, Inderveer Chana. Fault tolerance algorithm in alchemi.NET Middleware, national conference on education & research (ConFR10), Third CSI national conference of CSI Division V, Bhopal Chapter, IEEE Bombay, and MPCST Bhopal, organized by JUIT, India, 6-7th, 2010.
31. Neeraj Kumar Rathore. Inderveer Chana. Checkpointing algorithm in Alchemi.NET, Annual conference of Vijnana Parishad of India and national symposium recent development in applied mathematics & information technology, JUET, Guna, M.P, 2009.
32. Neeraj Kumar Rathore, Inderveer Chana. Comparative analysis of checkpointing, PIMR third National IT conference, IT enabled practices and emerging management paradigm book and category is communication technologies and security issues, Topic No/Name-46, prestige management and research, Indore, (MP) India, 2008; 32-35:46.
33. Neeraj Kumar Rathore, Inderveer Chana. An Efficient load balancing technique for grid in Scholar's Press, Mauritius, Project id: 6621, ISBN: 978-3-330-65134-0, 2018.
34. Neeraj Kumar Rathore and Pramod Singh. An Efficient Load Balancing Algorithm in Distributed Networks Lambert Academic Publication House (LBA), Germany, ISBN: 978-3-659-78892-5, 2016.
35. Neeraj Kumar Rathore, Rohini Chohan. An Enhancement of GRIDSIM architecture with load balancing in Scholar's Press, Project Id: 4900, ISBN: 978-3-639-76989-0, 2016.
36. Neeraj Kumar Rathore, Anuradha Sharma. Efficient dynamic distributed load balancing technique in Lambert Academic Publication House, Germany, Project ID: 127478, ISBN No: 978-3-659-78288-6, 2015.
37. Neeraj Kumar Rathore, Inderveer Chana. Checkpointing algorithm in Alchemi.NET in Lambert Academic Publication House (LBA), Germany ISBN-10:3843361371, ISBN-13:978-3843361378, 2010.
38. Neelesh Jain, Neeraj Kumar Rathore, Amit Mishra. An Efficient image forgery detection using Biorthogonal wavelet transform and singular value decomposition in 5th International conference on advance research applied science, environment, agriculture & entrepreneurship development (ARASEAED), Bhopal organized & sponsored by Janparishad, JMBVSS & International council of people at Bhopal (M.P) India, held on 04-06, 2017, 274-281.
39. Neeraj Kumar Rathore, Inderveer Chana. A Cogitative analysis of load balancing technique with job migration in grid environment, World congress on information and communication technology (WICT), Mumbai, IEEE, book ISBN: 978-1-4673-0127-5, book e-ISBN: 978-1-4673-0126-8, 978-1-4673-0125-1, 2011, 77-82. DOI10.1109/WICT.2011.6141221.
40. Neeraj Kumar Rathore, Inderveer Chana. Checkpointing algorithm in Alchemi.NET, Pragyaan: Journal of information technology, IMS Dehradun. ISSN No: 0974-5513, IEEE, CSI, MPCET Dehradun, 2010; 8(1):32-38.
41. Neeraj Kumar Rathore, Inderveer Chana. Comparative analysis of checkpointing, PIMR Third National IT conference, IT enabled practices and emerging management paradigm book and category is Communication technologies and security issues, Topic No/Name-46, prestige management and research, Indore, (MP) India, 2008, 32-35.
42. Rathore NK, I Chana. A cogitative analysis of load balancing technique with job migration in grid environment. IEEE proceedings, 2011, 77-82.