On a Virtual Shared Memory Cluster System with Virtual Machines

Minakshi Tripathy and C.R. Tripathy

Abstract—**In this paper, an architecture with a load balancing model and a fault tolerant model for virtual shared memory clusters is proposed. The centralized dynamic load balancing model uses manager worker concept with a virtualized compute server. The virtual server can be expanded "on the fly" for load balancing by adding virtual machines temporarily. The fault tolerant model controlled by a virtual machine monitor describes the checkpointing based recovery methods. The performance evaluation results show that the proposed system achieves significant speedup in terms of execution time and checkpoint time.**

Index Terms—**Virtual cluster, virtual machine, virtual server, load balancing, fault tolerance, check pointing and recovery.**

I. Introduction

Modern cluster computing increasingly execute on clusters that rely upon virtual machines. Virtual shared memory cluster system supports the creation and management of virtual clusters in seamless way. Virtual clusters map and dynamically remap them onto the nodes of the physical cluster. The clusters of high end workstations and PCs have a number of advantages over traditional multiprocessor systems. They exhibit shorter design cycles and tower costs than the tightly coupled microprocessors. Such system can also benefit from heterogeneity. One of the most important aspects of the modern clusters, especially in the context of commercial applications, is the potential for providing highly available system since component replication is not as costly as in other architecture. Moreover, due to the success of shared address space abstraction, many new applications are being developed for this abstraction [1-2].

The virtual shared memory cluster system described in this paper provides a virtual address space i.e. shared among all processors in a loosely coupled distributed shared memory cluster system [3-4]. This cluster system also provides a simple abstraction of logical cluster of virtual machines or virtual clusters. It can create any number of virtual clusters of arbitrary size, while multiplexing individual virtual machines on the available physical machine hardware [5-7]. Virtual shared memory clusters are composed of several processing nodes connected by a commodity network. Since the nodes are physically distributed, independent and heterogeneous, it offers great flexibility when maintaining and upgrading processors [8-10].

Crash and omission failures are common in service providers. The probability of a node failure increases with the number of nodes. Apart from reducing the providers computation power, this can also lead to wastage of computation time when the crash occurs before finishing the task execution. In order to avoid this problem, checkpoint based recovery mechanisms have been proposed [11-13]. These mechanisms record the system state periodically to establish recovery points. Upon a node crash, the last checkpoint is restored and task execution is resumed from that point. However, in virtual shared memory systems where the checkpoint mechanism has to deal with huge virtual machines, it must be saved and restored. For this reason, these environments require efficient checkpoint based fault tolerant models [14-16].

The virtual shared memory can serve as a platform independent repository, centralized dynamic load balancing virtual model and can transfer data among heterogeneous processors. The virtual clusters run a guest operating system of either a linux or Microsoft windows, which may differ, from the host operating system that handles the physical cluster activities [17-20]. In this paper, we discuss the centralized dynamic load balancing with virtual shared memory clusters. The proposed centralized dynamic virtual model supports load balancing dynamically on the fly with heterogeneous processor. It provides high performance with ease of use including small number of simple operations. This paper also proposes a checkpoint and recovery based fault tolerant model for virtual shared memory clusters. The checkpoints are stored in a distributed file system. Using this technique, the checkpoints are distributed and replicated among all the nodes of the service provider and eliminates any single point of failure. There is no performance bottleneck in the proposed checkpoint mechanisms because the checkpoint can be concurrently recovered from different nodes. This allows resuming faster task execution after a node crash under contention. Our system is able to recover and reconfigure itself after any number of successive single node failures under a fail stop model.

The rest of the paper is organized as follows. The various notations used in the paper are presented in Section 2. The Section 3 presents the proposed architecture for the virtual shared memory cluster system. The Section 4 describes the proposed centralized dynamic virtual model for load balancing with "on the fly" concept. The Section 5 describes the proposed fault tolerant virtual model for fault tolerance with check pointing and recovery mechanisms. The Section 6 discusses the performance evaluation and the results are compared with previous work [7], [10], [16]. Finally, we

Manuscript received January 20, 2011; revised march 8, 2011.

Minakshi Tripathy is wuth Sambalpur University, Jyotivihar, Burla, Sambalpur, Orissa, India (e-mail: minakshiom@ gmail.com).

C.R. Tripathy is with the Department of Computer Science and Engineering, V.S.S. University of Technology, Burla, Sambalpur, Orissa, India (e-mail: minakshiom@yahoo.co.in).

draw our conclusion in the Section 7.

II. Notation and Assumptions

The following notations and assumptions are used throughout this paper to describe a number of different system parameters.

Notation

VC	Virtual Cluster
VM	Virtual Machine
N	Total number of clusters and virtual clusters
N	Total number of processors and virtual machines
M	Total memory of virtual machine
B	Network bandwidth to migrate task
P_i	Migrated data rate
t_i	Copy time to transfer memory
T_c	Total copy time
F	Number of instructions executed by CPU
I	Number of instructions executed by VM
U	Average CPU usage of VM
T	Total migration time
R	Transaction rate
T_{comp}	Computation Time
T_o	Checkpointing overhead
$Trans_o$	Checkpointing overhead
R_o	Relative overhead
N_{chk}	Number of checkpoints
T_{chk}	Checkpoint time
T_{upload}	Time to upload Checkpoint in DFS
w_i	Processing power of i^{th} VM
w_o	Execution speed of migrated VM
w	Processing power of VC
S	Execution time of a cluster
E	Efficiency of cluster

Assumptions

i. All physical processors are heterogeneous in nature.
ii. System consists of several VCs including several VMs.
iii. VMs are homogeneous in nature.
iv. Centralized server is virtual in nature.

III. Proposed Virtual Shared Memory Cluster System (VSMC) Architecture

Modern clusters are turning to the approach of a manager worker model. The proposed virtual compute server works as manager on a number of co-operating workstations or PCs working as workers. Centralized management of the virtual server controls the entire system. Virtual shared memory cluster system use data pull semantics; rather than data push semantics of other middleware technologies. This makes it extremely easy to load balance among processors, since each processor can take just as much or as little data as it can handle. Due to pull semantics, hardware enhancements and upgrades are simple. Newly added processors starts to interact immediately after they are connected. The replacement can be accomplished by shutting down a particular processor and installing a new one in its place.

This section proposes the architecture of the virtual shared memory cluster (VSMC) system. The proposed system is designed as a flexible platform for constructing virtual clusters (VC) with virtual machine (VM) management. The Fig. 1 depicts the proposed VSMC system with its core components.

The system supports dynamic creation and management of VCs on a physical cluster. It aggregates many VMs in a VC. Each node of a VC is connected to a private virtual network and to the underlying physical network. The nodes of a VC are homogeneous in terms of virtualized resources (e.g. memory size, number of CPUs, private disk size etc) and operating system. Different clusters may exploit different configurations of virtual resources and different operating systems. The running VC shares the physical resources according to a creation time mapping onto the physical cluster. The VCs may be reallocated by means of the run time migration of the VM between physical nodes. One local node manager (LNM) runs on each physical node and interacts directly with the virtual machine monitor (VMM) to perform management operations such as creating, deleting and migrating VMs on behalf of the controller. The LNM also collect resource usage data from the VMM and monitors local events. Next, the LNM reports resource usage updates and events back to the controller. The controller is the central component of the VSMC system. It communicates with the LNMs to collect usage data and manage VMs running on each physical node. The controller provides event notification to the users to receive a notification for a particular event such as: VM has started, been destroyed or changed etc.

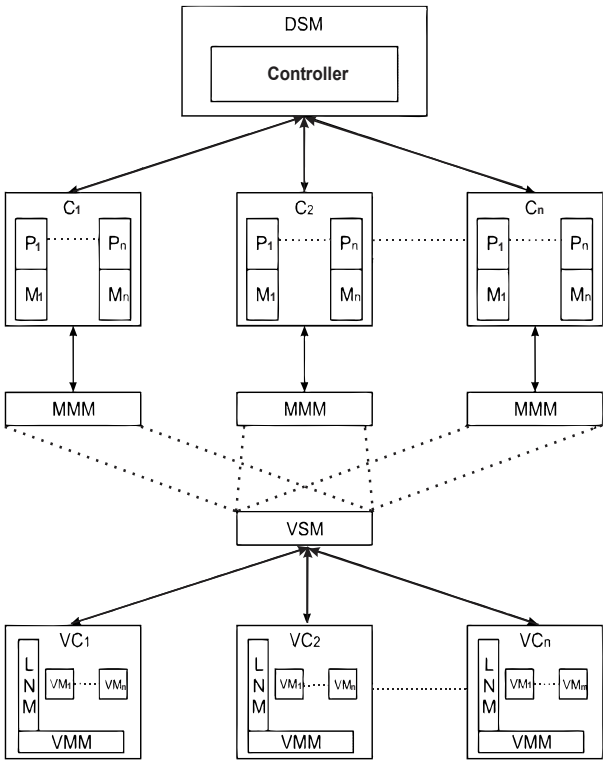

Fig 1. Architecture of the virtual shared memory cluster system

A. The System Components

The proposed VSMC system consists of the following elements.

1) Virtual Shared Memory (VSM)

A Virtual Shared Memory (VSM) is a single address space shared by a number of processors. Any processor can

access any memory location in the address space directly. Memory mapping managers (MMM) implement the mapping between local memories and the shared virtual memory address space. In addition, their chief responsibility is to keep the address space coherent at all times i.e. the value returned by a read operation is always the same as the value written by the most recent write operation to the same address. A part of each computer's memory is remarked for shared space and the remainder is a private memory. To provide the programmers the illusion of a single shared address space, a memory mapping management layer is required to manage the VSM space.

2) Virtual Cluster (VC)

A Virtual Cluster (VC) is a collection of Virtual Machines (VM) that are running onto one or more physical nodes of a cluster and that are connected by a virtual private network. All the VMs are homogeneous i.e. each VM may access a private virtual disk. All the VMs of a VC run the same operating system and may access a shared disk space. Different VCs may coexist on the same physical cluster, but no direct relationship exists among them apart from their concurrent access to the same resources. The virtualization is used to deploy multiple VCs, each exploiting a collection of VMs running on the operating system and associated services and applications. Therefore, the VMs of different VCs may be targeted to exploit a different operating system and applications to meet different user needs.

3) Virtual Machine

A classic virtual machine is an efficient isolated duplicate of the underlying machine. The VM enables the sharing of a single physical computer in terms of CPU, memory and I/O devices by multiple VCs, each independently configured with their own OS and application. A system consisting of VMs includes the development and testing of new OSs, simultaneously running distinct OS on the same hardware and server consolidation. A VM environment is created by VMM also called an "OS for OSs". The monitor creates one or more VMs on a single real machine. Each VM provides facilities for an application or a "guest OS" that believes to be executing on a standard hardware environment.

4) Controller

The controller is the center of the VSMC system. It can either be bootstrapped into a VM running in the system, or run on a separate server. The controller has the following functions.

i) User Authentication- All users of VSMC system must authenticate before making requests to the system.
ii) VM operation- The controller invokes processes for VM management requests such as checking resource availability and making placement decision.
iii) Global state maintenance- The controller maintains the lists which constitute the global state for running VM and VMM. It also instantiates VCs which may contain an arbitrary set of VMs.
iv) Monitoring data- The controller is responsible for consolidating monitoring data sent by the LNMs into a format accessible by the system. Users use this data to describe the state of the system. Modules use this data to restrict user resource requests based on the current system load or make VM scheduling decisions to determine where VMs should run.
v) Event notification- The VSMC system often needs to alert users when various events in the system occur. Events typically fall into one of the three categories: VM operation requests, VM state changes and Errors or unexpected events.

5) Local Node Manager (LNM)

The Local Node Manager (LNM) of the VSMC system presents modules to the controller for manipulating VMs on its node. The LNM translates the operation into equivalent VM operation by the VMM running on the node. As the VSMC system runs, VM and VMM resource usage fluctuates considerably. The LNM on each node monitors these fluctuations and reports them back to the controller. It reports such as resource usage of CPU utilization, network receive and transmit load, disk I/O activity and memory usage.

6) Virtual Machine Monitor (VMM)

In contrast to a non virtualized system, full virtualization of all system resources makes it possible to run multiple operating systems on a single physical platform. A virtualized system includes a new layer of software called a Virtual Machine Monitor (VMM). The principal role of the VMM is to arbitrate access to the underlying physical host platform resources so that these resources can be shared among multiple operating systems that are the guests of VMM. The VMM presents to each guest operating system a set of virtual platform interfaces that constitute a VM. The VMMs build some properties that are useful in system security like isolation, inspection and interposition. The isolation means software running in a VM can not access or modify the monitor or other VM, inspection means the monitor can access the entire VM state and interposition means the monitor can intercept and modify operation issued by a VM.

B. The System Modules

In this section, the process modules of VSMC system are described with the interfaces that each component supports. The module1 queries for VM state information and VM resource usage details by LNM and passed to the controller. The module2 lists the operations used by the controller to the VSMC system users. Since the system is designed to manage clusters, the common case is to invoke these operations on lists of VMs rather than on a single VM at a time. The Table1 shows the list of valid operations to perform on VM. The Table2 shows the operations, which are used by the VSMC system users.

1) LNM Module:

This module is managed by LNM and results are passed to the controller. The responsibility of this module is to make the VM ready and to operate on the existing VM.

Procedure LNMprocess
Get VM state information
Get VM resource usage statistics
Get hardwae characteristics of the physical machine
Operate on existing VM
End procedure

2) Controller Module:

This module is managed by the controller to start list of

VMs on LNMs.

Procedure Controller process
List state and resource usage information for VMs
List LNMs and resource usage information for VMMs
Operate on existing VMs
Migrate VMs to LNMs
End procedure

TABLE I: VM OPERATIONS

Operation	Description
Pause	Pause VM execution
Resume	Resume execution of a paused VM
Shutdown	Nicely halt a VM
Reboot	Shutdown and restart VM
Store	Save VM's memory image to persistent storage
Restore	Restore VM to run state
Destroy	Hard shutdown a VM
Cycle	Destroy and restart a VM

TABLE II: USER OPERATIONS

Method Name	Description
Connect	Authenticate and connect with controller
Reconnect	Reconnect to the controller upon an unexpected disconnect

IV. PROPOSED CENTRALIZED DYNAMIC VIRTUAL MODEL (CDVM)

This section proposes a centralized dynamic virtual manager worker model (CDVM). The conceptual view of the same is depicted in the Fig. 2. The centralized server is virtual in nature and acts as the manager. The processors of different clusters act as workers. To use the CDVM, the incoming jobs are listed and submitted to the server or manager accessing the VSM. Similarly, all the processors or workers, which are physically, distributed at different locations access a common DSM. When the jobs are submitted to the manager, the workers work to complete the job as quickly as possible. Since, the computational power of the server is derived from an aggregation of processors, it is possible to expand the system by adding new processors without affecting the ongoing activities. The server can be expanded "on the fly" to provide services for the job requests in demand by adding processors temporarily.

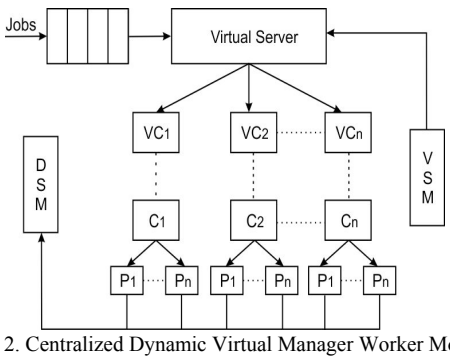

Fig. 2. Centralized Dynamic Virtual Manager Worker Model

A. Load Balancing

This subsection describes the load balancing aspect of the VSMC systems. The load balancing in VSMC system is an intellectually challenging activity. Load balancing in other words is to adapt easily to changes in the processor or job pool that demand dynamic reassignment of resources to meet the evolving system requirements. This can be also termed as dynamic scheduling.

The virtual manager using address spaces, job list and worker list controls the entire CDVM. These are the spaces occupied in VSM that are globally shared. The job list contains the following describing jobs.

i) An unique job identification number for each job.
ii) Jobs pending to indicate whether there are any active jobs in the system.
iii) Number of active jobs currently in the system.
iv) One pointer pointing to the active job.

The workers list contains the followings describing workers.

i) An unique worker identification number for each worker.
ii) Number of active workers currently in the VSMC system.
iii) One pointer pointing to the active worker.

The manager is virtualized and used for initialization, control and management of the VSMC system. Our proposed CDVM deals with load balancing with the variations in task difficulty or worker speed which may cause some tasks to take longer than others. It is possible to add processors or workers "on the fly" dynamically without changing or reconfiguring the VSMC system. The VSMC system can be up and run full time, with hardware maintenance and with a little or no upgradation to few machines or even without changing the system.

B. Load Balancing Modules

This subsection proposes the load balancing modules for the proposed CDVM.

1) Manager Module:

The purpose of the manager module is to provide services to the users. It divides its job into a number of tasks that can be distributed to individual worker nodes in the virtual server.

Procedure manager process
Enter jobs to job list
Divide the job into tasks
For Each task in a job
Distribute tasks to workers of Virtual Server
Create new job address space in VSM to hold data that Are globally shared
Insert tasks to job address space
Execute and receive results of task
End For
Delete the job from job address space
End procedure

2) Worker Module:

The worker module on each worker node in the virtual server executes tasks of job in the job address space.

Procedure Worker process
For Each worker in the Virtual Server
Select a job from job list
For Each job in job address space
Select a new task
Perform task computation and receive results
End For

End for
End procedure

3) Selectjob Module:

The purpose of selectjob module is to find a suitable valid job for the worker.

Procedure Selectjob
For Each worker in the worker list
For Each job in the job list
Select a new job
If the status of the job is valid
Update job list and worker list
End If
Commit the transaction
End For
End For
End procedure

4) Addjob Module:

The addjob module is responsible for modifying the job list and taking other steps to submit the new job to the system.

Procedure addjob
Obtain the job list
Assign an unique id to the new job
Create a new job address space to hold task and results
Update job list
Commit transaction
End procedure

5) Remove Job Module:

The remove job module retrieves the job list to delete the desired job.

Procedure remove job
Retrieves the job list
Select the job to remove
Delete the job and job address space
Update job list
Commit transaction
End procedure

6) Add Worker Module:

The add worker module is structured in much the same way as add job module. It retrieves the worker list to check whether the system can handle additional active worker or not.

Procedure add worker
Retrieves the worker list
If the system capacity supports additional worker
Assign an unique worker identifier
Update the worker list
Commit transaction
Else
Cancel the transaction
End if
End procedure

V. Proposed Fault Tolerant Model (FTVM)

This section presents the details of the proposed fault tolerant model. Our fault tolerant model as shown in the Fig. 3 is built on top of a virtual server (VS) that uses VMs created on demand.

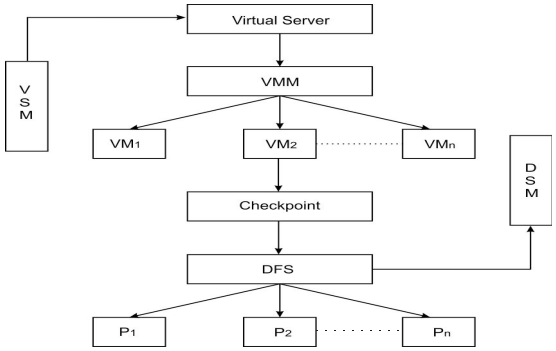

Fig. 3. Fault tolerant virtual model

The VMM takes complete control of the machine hardware and creates VMs, each of which behaves like a complete physical machine. The VMM virtualizes as a shared memory multiprocessor machine on a commodity cluster. The VMM also hides the dynamic changes of physical hardware configurations. It allows a VM to provide a fixed number of processors even if available; the physical machines are added or removed dynamically. An application running inside a VM seems to have N processors all the time even if the number of available physical machines decreases and becomes less than N.

Making a checkpoint of task running within a VM imply moving of data, since it must include all the information needed to resume the task execution in another node. In order to make a checkpoint, the task status i.e. memory and disk contents needs to be saved. A task should be stopped while doing a checkpoint. This is because, if the task changes its status while the checkpoint is being done, checkpoint inconsistency may occur due to the concurrent accesses. Once the checkpoint is ready and the VM is stopped, it could be directly uploaded to the DFS.

The checkpoints can not be stored in the node where the task is running, because if the node crashes, the checkpoint will not be accessible. It would be a single point of failure resulting in a non fault-tolerant solution. To overcome this problem, the proposed model uploads the checkpoints to a DFS. This system splits the stored data in blocks that are distributed among the different nodes. Every node in the VS is a part of the DFS and stores some of the blocks of the checkpoints. The checkpoints are replicated and distributed among the different nodes. In this way, if one node crashes, the checkpoints can be restored from the other nodes. As the checkpoint is replicated, it can be concurrently obtained from several nodes.

C. Failure Recovery Modules

This subsection presents recovery modules when a failure is detected from the normal event errors in job or memory address spaces. The recovery modules are controlled by the VMM. If an error occurs in an operation on VSM address space, the user assumes that either server has failed or VSMC system has been shut down and destroyed address space. In either event, one VM is removed and if an error occurs in an operation on a job address space, the server remains still operational.

1) Check pointing Module:

The checkpoint module running on each VM responds to

the requests to checkpoint tasks on that VM. This receives a request to checkpoint a list of tasks running on a node. Then it checks for whether the task is in a safe state and initiate check pointing.

Procedure Check pointing
For Each task in a fixed interval of time
Commit transaction on all VMs.
Save checkpoint states to VM memory.
Load the checkpoints made to DFS of processor disk
End For
End procedure

2) Recovery Module:

The completion of a task is indicated to VM by failure of a operation following the destruction of job address space from the virtual server. This indicates normal completion of the job and the user is supposed to continue selecting a new job.

Procedure recovery
For a failure of a task
Retrieve the replicas from DFS
Download all the checkpoints
Merge processor disks
Select a new job or new VM
Resume the execution
End For
End procedure

VI. Performance Analysis

In this section, the performance analysis of the VSMC system architecture based on the theoretical analysis has been made.

A. Theoretical Analysis

This subsection makes an analysis for virtualization abstracts with resources such as CPU and memory through generating VMs to support resource assignment. In particular the VM migration has been applied for flexible resource allocation or reallocation by moving applications fom one physical machine to another for stronger computation power, larger memory, fast communication capability and energy savings. The original host machine copies the memory data of the VM that is to be migrated and sends the data to the receiving destination host machine. In the mean time, the host records the changed bits of memory during the migrated phase. These migrated data bits are generated by running application in the VM and delivered to the destination in next rounds.

We define the parameter 'M' as the total memory of the VM and 'B' as network bandwidth assigned to migrate task. To describe the speed of memory changing in iteration i, Pi is introduced as the migrated data rate of the round i. Now, the time for transferring memory in round i can be calculated from every migrated data rate of previous iteration according to the following equation [10, 20].

$$t_i = \frac{P_{i-1} \times t_{i-1}}{B} = \frac{M \times \pi_{k=1}^{i-1} P_k}{B^i} \tag{1}$$

$$Theorem \quad 1 : T_c = M \times \sum_{i=1}^{n} \frac{\pi_{k=1}^{i-1} P_k}{B^i} \tag{2}$$

Proof: When the migrated data of iteration is small enough or too many rounds have been done, the migration process transfers the remaining changed memory for the last time. In the final round, VM is stopped and its memory is not writable to ensure that no more data are to be sent. To analyze the relation between each parameters and the performance of the whole process, the transfer time of all rounds is summed up and hence the result for copy time.

$$Theorem \quad 2 : R = \sum_{i=1}^{n} \frac{\pi_{k=1}^{i-1} P_k}{B^{i-1}} \tag{3}$$

Proof: It is clear from equation (2) that the total copy time Tc would be longer while the migrated VM has more total memory and/or less availability network bandwidth. For a given Pk, when it is increased, the copy time of all rounds after the current one will be extended. We use R as the extended transferred memory while copying migrated data as below. So, R can be given as follows and hence the result.

$$R = \frac{T_c \times B}{M} \tag{4}$$

To quantitatively analyze, we introduce parameters F representing the number of instructions executed by CPU per second, it is a variable only relative to the main frequency of processor. The VM's executed instructions during migration are represented as follows.

$$I = FxUxT \tag{5}$$

where U is VM's average CPU usage and T is the total time of migration.

$$Theorem \quad 3 : w = \sum_{i=1}^{N} w_i \tag{6}$$

In [10], the authors assign each VM a separate weight representing the proportion of the CPU share. The weight is a relative value which means the actual CPU allocated to a VM is decided by its weight compared to that of the other VMs. The controller allocates more CPU time to the one with higher weight. So the migrated VM's executing speed is given by

$$w_o = \frac{e}{1 - e} x \sum_{i=1}^{n} w_i \tag{7}$$

where e is the expected percentage of CPU time allocated to the VM and $w_1, w_2 \ldots . w_n$ are the weights of VMs running on host except the one being migrated. The weight or processing power of a VM_i, can be defined as the inverse of the time per operation on this VM.

$$w_i = \frac{1}{t_{op_i}} \tag{8}$$

Hence the result for the cluster execution time or the processing power 'w' of a virtual cluster with P processors.

For fault tolerance, the checkpoint time in virtual shared memory clusters is very small as this system has high memory bandwidth or higher network bandwidth.

$$Theorem \quad 4 : R_o = T_o . N_{chk} \left(\frac{R}{1 + T_o . N_{chk}} \right) \tag{9}$$

Proof: If R is transaction rate then the computation time

for each transaction is defined as

$$T_{comp} = \frac{1}{R} \tag{10}$$

If each transaction modifies for N_{chk} number of checkpoints then the system checkpoints at every t seconds. So for the check pointing overhead T_o, transaction overhead is as follows.

$$Trans_o = T_o.N_{chk} \tag{11}$$

Then the relative overhead with this checkpoint frequency is defined as follows and hence the result.

$$R_o = \left(\frac{T_o . N_{chk}}{\frac{1}{R} + T_o . N_{chk}} \right) \tag{12}$$

A fault tolerant cluster system should satisfy the following condition.

$$T_o \leq R_o \tag{13}$$

$$Theorem\ 5:\ T_{chk} = T_{upload} + Trans_o \tag{14}$$

Proof: Now the checkpoint time (T_{chk}) is the summation of checkpoint transaction overhead time and checkpoint uploaded time (T_{upload}) to distributed file system and hence the result.

Now, for a VSMC system with N virtual machines and P processors, the speedup S (N,P) can hold the relation as below.

$$S(N, P) = \sum_{i=1}^{N} w_i \tag{15}$$

This implies that S (N,P)<P for 1≤i≤P (16)

Finally with size N and a VM with P processors, The efficiency E (N,P) is defined as

$$E(N, P) = \frac{S(N, P)}{P} \tag{17}$$

D. Results and Discussions

This subsection discusses on the results. For the experimentation the core java programming codes are developed. Several experiments are conducted on 16 node clusters and the average results are presented. Experimental data has been obtained based on the SQL server, database and programming. The global states are kept in a SQL backing database. The SQL server initializes the file system to be mounted prior to starting up a new VM. The Table 3 shows the directories that it creates for each VM.

In all the experiments, the base system disk has been already uploaded to the checkpoint storage, thus only regular checkpoints must be uploaded. We computed execution times on various window sizes and then the results are compared with that of WMU [7]. The Fig. 4 shows a comparative picture of the execution times of the proposed VSMC and WMU systems. Apart from the significant performance improvements, the execution time is observed to be better then the WMU system [7]. The execution time versus number of processors is also compared with existing load balancing model [10]. The MemX in [10] consists of multiple VMs with a server while the proposed CDVM model consists of a virtual server with multiple processors. The MemX codes were developed using C programming code and its execution time was extracted from quicksort method. The Fig. 5 shows that the execution time in CDVM is almost about two times smaller than that of MemX. Further, it provides a clear high performance edge in terms of efficiency. The proposed FTVM is compared with an existing fault tolerant architecture CFTI [16]. The checkpointing time is an average of the checkpointing time measured on 16 VMs with a processor disk of 100MB that contains the information required to boot the VM. The Fig. 6 shows the time to make a series of checkpoints within an interval of 100 seconds in the VMs with varying memory sizes. The time to make a checkpoint mainly depends on two aspects: creating the checkpoint and uploading to the checkpoint storage. The time needed to make a checkpoint increases with the memory size of VM as well as processor disk. It is basically due to the time required to save the memory and restart the VM execution after the checkpoint has been done. Therefore, the time to upload the checkpoint of the memory is proportional to the memory size. As shown in the Fig. 6, the first checkpoint is found to be more expensive then others, as all the processor disks must be uploaded to the checkpoint storage, while in rest of the checkpoints only the modifications on the processor disk and memory are uploaded. As shown in the same figures, the checkpoint time of the proposed FTVM is less than that of CFTI. Obviously the lower checkpoint time benefits from having lower task execution time.

TABLE III: SQL SERVER DIRECTORIES

Directory	Description
Global	Stores data that is globally accessible by all clusters with all VMs on the VSMC system.
Cluster	Stores files accessible by the current cluster including its VMs in the same cluster.
Local	Sets up services and configuration files specific to particular VM.
Home	Automatically takes care of mounting these directories upon login.

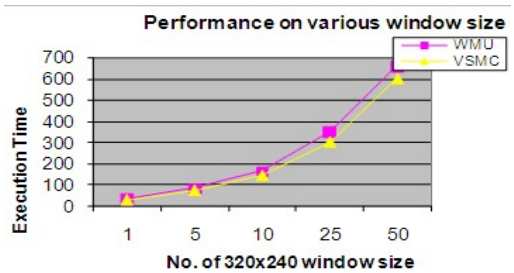

Fig. 4. Number of window sizes (324x240) vs execution time

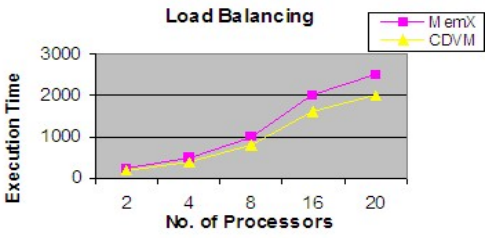

Fig. 5. Number of processors vs execution time

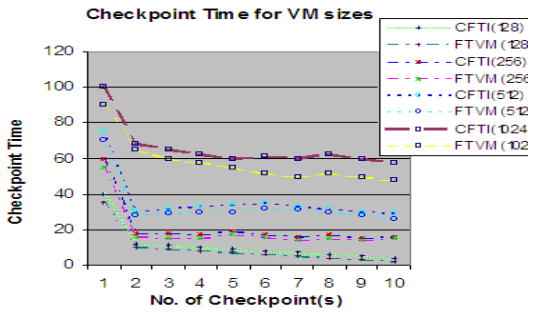

Fig. 6. Number of checkpoints VS checkpoint time for different VM sizes

VII. Conclusion

In this paper a new virtual shared memory cluster system (VSMC) is proposed. The proposed VSMC architecture is an extensible, event driven management system for clusters of virtual machines. It supports the basic virtual machine and virtual cluster management techniques such as creating, destroying, manipulating and migrating virtual machines. It also supports customizable modules for flexibility. The proposed architecture has the mechanism of load balancing and manageability with central control and management from virtual server. It handles dynamic scheduling very efficiently with the proposed centralized dynamic virtual model (CDVM) model. The CDVM model with the concept of "on the fly" can add workers any time to the proposed system with its ongoing activities. The proposed fault-tolerant virtual model (FTVM) reduces the checkpoint time and as a consequence, the interference on task execution. The checkpoints stored in a distributed file system, are distributed and replicated on all the virtual machines of virtual server to guarantee system recovery when a failure occurs. The experimental results obtained, clearly shows that the efficiency, performance, robustness and ease of use of the proposed system is better than that of the existing system.

References

[1] D. J. Sorin, M. M. K. Martin, M. D. Hill and D. A. Wood, "Safety Net: Improving the Availability of Shared Memory Multiprocessors with Global Checkpoint/Recovery," *In the proceedings of the 29th Annual International Symposium on Computer Architecture*, 25-29 May 2002, pp 123-134.2002.

[2] M. T and C. R.Tripathy, "Design and analysis of a Dynamically Reconfigurable Shared Memory Cluster," *International Journal of Computer Science and network Security*, 10 (9), Sept 2010, pp 145-158.2010.

[3] K. L. Wu and W. K. Fuchs, "Recoverable distributed shared virtual memory: Memory coherence and storage structures," *Nineteenth International Symposium on Fault- Tolerant Computing*, 21-23 Jun 1989, pp 520-527.1989.

[4] K. L.Wu and W. K. Fuchs, "Recoverable Distributed Shared Virtual Memory," *IEEE Transactions on Computers*, 39 (4), Apr 1990, pp 460-469.1990.

[5] K. Li and P. Hudak, "Memory Coherence in Shared Virtual Memory Systems," *ACM Transactiona on Computer Systems*, 7 (4), Nov. 1989, pp 321-359. 1989.

[6] M. McNett, D. Gupta, A. Vahdat and G. M. Voelker, "Usher: An Extensible Framework for Managing clusters of Virtual machines,"*In the proceedings of 21st large installation System Administration conference*, 11-16 Nov 2007, pp 167-181.2007.

[7] M. Vuletic, P. Ienne, C. Claus and W. Slechele, "Multithreaded Virtual Memory Enabled Reconfigurable Hardware Accelerators,"*IEEE International Conference on Field Programmable Technology,* Dec 2006, pp 197-204.2006.

[8] C. Clark and K. Fraser et.al. "Live Migration of Virtual Machines," *In Proceedings of the 2nd ACM/USENIX symposium on Networked Systems Design and Implementations*, May 2005, pp 273-286.2005.

[9] J. T. Pfenming, A. Bachem and R. Munnich, "Virtual Shared Memory Programming on Workstations Clusters," *International Journal of Future Generation Computer Systems*, 11(4-5), Nov 1995, pp 387-399.1995.

[10] M. R. Hines and K. Gopalan, "MemX: Supporting Large Memory Workloads in Xen Virtual Machines," 2nd International *Workshop on Virtualization technology in Distributed Computing*, Nov 2007, pp 1-8.2007.

[11] Y. Zhou, P. M. Chen and K. Li, "Fast Cluster Failover Using Virtual memory- Mapped Communication," In the proceedings *of 13th International Conference on Supercomputing*, 20-25 June 1999, pp 373-382.1999.

[12] M. Tripathy, C. R. Tripathy, and B. D. Sahoo, "Job admission controls in cluster networks with queuing methods,", *Proceedings of the International Conference on Computing, Communication and Information Technology Applications*, 21–23 Jan 2010, pp 235-241.2010.

[13] Minakshi Tripathy and C.R. Tripathy, "Centralized Dynamic Load Balancing Model for Shared Memory Clusters," *Proceedings of the International Conference on Control, Communication and Computing,* 18-20 Feb 2010, pp 173-176.2010.

[14] Paul Pop and Viacheslav Izosimov et.al, "Design Optimization of Time and Cost constrained Fault-Tolerant Embedded Systems with Checkpointing and Replication," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems,* 17 (3), Mar 2009, pp 389-402.2009.

[15] Y. Zhang and K. Chakraborty, "Fault recovery Based on Checkpointing for Hard Real-Time Embedded Systems," *Proceedings of the 18th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, 3-5 Nov 2003, pp 320-325.2003.

[16] I. Goiri and F. Julia, "Checkpoint-based Fault Tolerant Infrastructure for Virtualized Service Providers," *12th IEEE/IFIP Network Operations and Management Symposium,* 19-23 Apr 2010, pp 455-462.2010.

[17] R. Iyer and R. Illikkal, "VM3: Measuring, modeling and managing VM shared resources," *Computer Networks Journal,* 53 (17), 3 Dec 2009, pp 2873-2887.2009.

[18] M. Chapman and G. Heiser, "Implementing transparent shared memory on clusters using virtual machines," *In the proceedings of USENIX,* Apr 2005, pp 383-386.2005.

[19] E. J. H. Yero and M. A. A. Henriques, "Speedup and scalability analysis of Master–Slave applications on large heterogeneous clusters," *Journal of Parallel and Distributed Computing*, 67 (11), Nov 2007, pp 1155-1167.2007.

[20] S. Adabala and V. Chadha, "From virtualized resources to virtual computing grids: the In- VIGO system," *Journal of Future Generation Computer Systems - Special section: Complex problem-solving environments for grid computing*, 21 (6), June 2005, pp 896-909.2005.

Ms. Minakshi Tripathy received the degree of B.Sc. (PCM), M.Sc. (Statistics) and MCA from Sambalpur University. She has done 'A' level course from DOEACC, New Delhi. She is currently a Ph.D. (Computer Science) student at Sambalpur University, Burla, Orissa. She has publications in four different international conferences and two international journals. Her research interest includes shared memory, cluster computing, load balancing and fault tolerance.

Prof. (Dr.) C.R. Tripathy received the B.Sc. (Engg.) in Electrical Engineering from Sambalpur University and M. Tech. degree in Instrumentation Engineering from I.I.T., Kharagpur respectively. He got his Ph.D. in the field of Computer Science and Engineering from I.I.T., Kharagpur. He has more than 50 publications in different national and international Journals and Conferences. His research interest includes Dependability, Reliability and Fault–tolerance of Parallel and Distributed system. He was recipient of "Sir Thomas Ward Gold Medal" for research in Parallel Processing. He is a fellow of Institution of Engineers, India. He has been listed as leading scientist of World 2010 by International Biographical Center, Cambridge, England, and Great Britain.