Journal of Theoretical and Applied Information Technology
31st March 2014. Vol. 61 No.3
© 2005 - 2014 JATIT & LLS. All rights reserved·
ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

MDP: A PARADIGM FOR COMMUNICATION AND COMPUTATION IN MOBILE CLUSTER

[1]AGHILA RAJAGOPAL, [2]M.A. MALUK MOHAMED
[1,2]Software System Group, MAM College Of Engineering, Affiliated To Anna University Chennai.
E-mail: [1]ssg_akila@mamce.org , [2]ssg_malukmd@mamce.org

ABSTRACT

The proposed work explores the interprocess communication across communicating parallel tasks in a mobile cluster. The process which is linked with specific mobile or static nodes will not be resilient to the changing conditions of the mobile cluster. The proposed Mobile Distributed Pipes (MDP) model enables the location independent intertask communication among the processes executing in static and mobile nodes. This novel approach enables the migration of communicating parallel tasks during runtime, which occurs according to the context and location requirements. A transparent programming model for a parallel solution to Iterative Mobile Grid Computations (IMGC) using MDP is also proposed. The proposed model is resilient to the heterogeneity of nodes such as static or mobile and the changing conditions in the mobile cluster because of mobility. The design of runtime and functional library support for the proposed model is also presented.

Keywords: *Location-independent(LI)communication, Mobile Grid Computation Problem (MGCP), Mobile Grid Computation Task (MGCT), Iterative Mobile Grid Computation (IMGC), Iterative Mobile Grid Module(IMGM), and Mobile Distributed Pipes (MDP).*

1. INTRODUCTION

An effective communication of parallel tasks on a distributed computing system requires the selection of the nodes during the runtime. The number of nodes to be selected at real time in a mobile cluster is not static. The nodes and their communication links are prone to failure, which results in complexity of the system during communication processes of the nodes. The nodes could also be heterogeneous in terms of processing power, operating systems, and architecture and these nodes need to coordinate with each other. The uneven load in the network due to this heterogeneity in system properties needs to be balanced.

The load balancing is prioritized with respect to the granularity of the individual subtasks. When the programing of the intertask communication is transparent, it enables numerous application domains to employ the parallel computing power of the cluster of nodes. Some of the application domains are Iterative Mobile Grid Computations (IMGC), suboptimal algorithms, and network of filters.

The computing power in mobile systems is made ubiquitous through parallel and distributed computing. The mobile systems must ideally offer seamless computing, flexibility in communication, and higher availability of the distributed information. The computing resources of a system are integrated to work in a common system, forming clusters. Clusters are preferred as they yield better fault tolerance and price-performance ratio compared to conventional mainframe architectures.

Parallel computing has been performed using methods like ARC [13], ADM [16], EMPS [17], Sprite [19], Piranha [18], Condor [12], (Network of Workstations) NOW [15], and Batrun [14]. Of these techniques Condor, NOW, and Batrun are employed for wired networks and the remaining techniques do not involve intertask communication. So, a scheme is proposed for mobile cluster involving inter-process communication using Mobile Distributed Pipes (MDP).

An application of parallel and distributed computing is mobile telemedicine. A general telemedicine system comprises of a small group of hospitals which provides remote healthcare services [3]. But, in developing nations the majority of the population is in the rural areas which require larger Internet-based telemedicine systems. A variable Internet-based P2P architecture is used for telemedicine networks. This system is based on a store and forward model, involving a distributed context-aware scheduler.

A transparent programming model is used for the communication of the parallel tasks in a wide area grid [7]. The grid model involves Distributed Pipes with grid abstraction (GDP), which performs the location independent inter-process communication between machines. This technique allows the anonymous migration of parallel task communication with respect to the grid dynamics. The model can support both parallel load and sequential load. But this model does not provide support for mobile systems.

The physical mobility of the mobile nodes (MNs) may result in message loss in a distributed mobile computing environment [2]. The battery power can be conserved to a great extent when the message delivery is guaranteed exactly once. The limited characteristic of the mobile nodes involving in the mobile cluster requires an efficient communication scheme for exactly-once message delivery. So, an exactly once multicast protocol (EOMP) is used to increase the power efficiency. An unreliable wireless MAC layer multicast delivers the messages to the MNs. The EOMP tolerates the failure at the mobile support station (MSS) by turning the MSS stateless.

Computational mobile grid is regarded as an integration of mobile clusters [4]. Mobile cluster computing can also be implemented using IPv6 [10]. An Anonymous Remote Mobile Cluster Computing (ARMCC) method is introduced to use the stationary computing power of the nodes, to attain parallel programming in a distributed mobile system. The cluster model is extended to an environment of mobile grid that combines the computational, service, and data grids. The participating mobile nodes are referenced using surrogate objects [5] which are implemented as a shared distributed object space. An environment for the iterative grid-based applications involves a distributed solution [9]. The environment is constructed using a mobile agent system.

A Mobile Distributed Pipes (MDP) model involving mobile cluster computing is proposed in the mobile cluster architecture. The mobile nodes are combined with the static nodes to form a mobile cluster. The parallel computing is implemented in mobile clusters by utilizing the idle processing power of the nodes in the mobile clusters. The number of mobile nodes is higher than the number of the static nodes in the mobile cluster. The motivation for this work attributes to the extensive potential value of mobile cluster computing in real-time environments.

A mobile cluster is defined as a group of interconnected mobile nodes by wireless networks. The mobile cluster coordinates with a set of mobile nodes to execute a specific task. The mobile cluster provides flexibility in terms of mobility, cluster security, and extendibility. When the domain of an IMGC is classified, their subdomains require exchanging their boundary values. The mobile grid computations are applied in the solution of elliptical partial differential equations. The problem classification in the suboptimal algorithms and network of filters requires the subtasks to exchange their intermediate values. The MDP model performs better in terms of throughput, synchronization time, speedup, task time, memory requirements, packet drop fraction, and packet delivery ratio (PDR) with respect to the existing models DP [11], and Moset [6]. The MDP model works in the environment of a mobile cluster which composes a large coverage and multiple clusters, compared to the previous parallel computing methods.

Mobile Distributed Pipes (MDP) is proposed to handle the issues of mobility, bandwidth, and fault tolerance relative to the existing Distributed Pipes (DP) model [11], operating in the mobile cluster. The communication of the parallel tasks created at runtime is connected using the MDP. The MDP model defines the transparent programmability for the parallel task communication. MDP support data flow between tasks independent of their location. This supports the anonymous migration of the parallel task communication.

The communication channels between the mobile nodes are regarded as global entities. The information about these global entities is stored globally in a designated mobile node. The communication channels are created or destroyed only during the runtime. MDP handles heterogeneity by the use of external data representation.

The remaining part of the paper is organized as follows: Section II involves the MDP model of IMGC. Section III involves the design and implementation of MDP. Section IV involves the implementation of the MDP model in an image rendering application. Section V involves the background work related to parallel computing and distributed computing. Section VI involves the performance analysis and comparison of the proposed MDP model and existing models DP [11], and Moset [6]. The paper is concluded in Section VII.

2. MDP MODEL OF ITERATIVE MOBILE GRID COMPUTATIONS

The overall mobile cluster architecture is shown in Fig. 1. A general Iterative Mobile Grid

Computation (IMGC) consists of repetitive computations in time and space dimensions [11]. The general program structure of the problem in the sequential execution in IMGC is given in Pseudo Code 1. The outer loop of pseudo code 1 iterates in time and the inner loop in each space dimension.

Pseudo Code 1: Program Format for general IMGC

```
FOR Time = InitialTime TO FinishTime
    FOR SpaceX = InitialSpaceX TO FinishSpaceY
        FOR SpaceY = InitialSpaceY TO FinishSpaceY
UserCustomizedFunction ()
        ENDFOR
    ENDFOR
ENDFOR
```

A. Proposed Model

The model involves a master-worker computation model. There are numerous *master processes* and *worker processes*. The master process is the *MobileGrid Computation Problem* (MGCP) which initiates or requires the computation [11]. The worker processes are created on the mobile nodes which involve in the parallel computation. The worker process is the *MobileGrid Computation Task* (MGCT) which performs the parallel computation for the MGCP. The computation is performed through *Iterative Mobile Grid Modules* (IMGM), also known as the worker process. The communication of the parallel tasks is achieved through the process of *domain decomposition*. The several IMGMs are allotted a subdomain of the computation. The role of the MDP is to enable the exchange of the boundary values between the IMGM's of multiple mobile nodes. The computation results of all the IMGMs are then transferred to the Master Process.In the proposed model, the system handles selection of the least loaded mobile nodes, anonymous migration of IMGMs, features related to fault tolerance, load balanced classification of tasks, and results collection. The programs in this model can be accommodated to a changing mobile cluster. There is no limit on the number of IMGMs. Hence, the number of mobile nodes utilized will be optimized during runtime. The programs based on this model are tolerant to a heterogeneous group of unevenly loaded mobile nodes.

B. Initialization

InitializeProblem () is used to register the master process with the system and the process is terminated using *EndProblem* ().*StartIMGM* () is used to register the IMGM with the system and it is terminated using *EndIMGM* ().

C. Domain Decomposition

The mobile grid details are transferred to the system from the master process. The granularity of computation to be assigned to individual MGCTs, mobile nodes to be allotted for each worker process, and optimum number of worker processes is decided by the system [11]. The master process forms the initial data for the individual worker processes to transfer the worker processes and to establish channels that collect the output results from each IMGM.

The master process initiates the data for the individual worker processes based on the information from the system. The initial data enable the transfer of IMGMs and creation of channels to collect the output from each IMGM.

The master process sends the mobile grid details to the system using the function call *SendMobGridDetails* (). Similarly, the number of worker process to be used is obtained using the function call *EstimateSplitsCount* (). The granularity of each worker process is estimated using the function call *GetSplitDetails* (). A brief description of the function calls used is given in TABLE I.

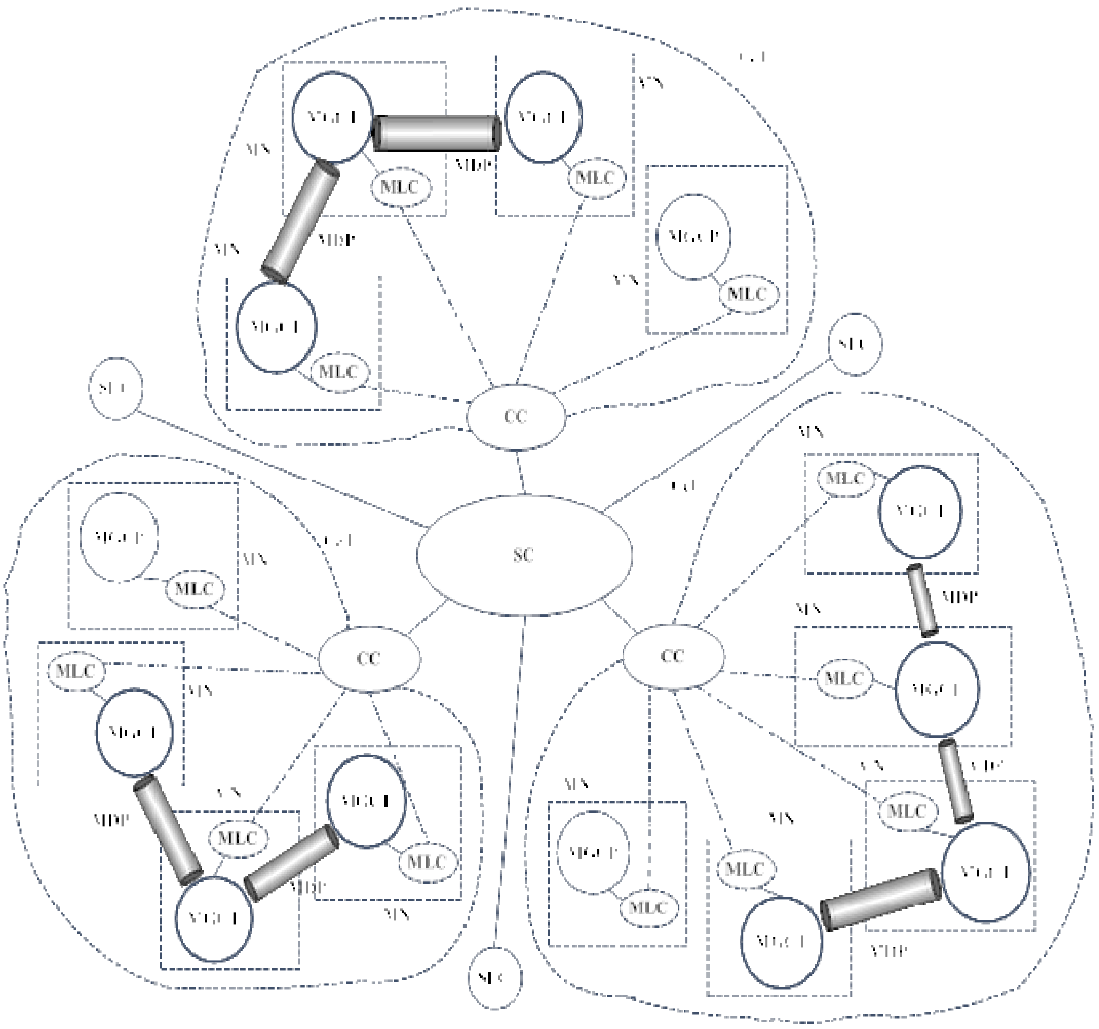

Fig. 1. Overview of mobile cluster architecture.

D. Anonymous Transfer of Subtasks

The anonymous transfer of a worker process is started when the MGCP invokes the *Transfer*() function call. But, the master process does not provide any machine specific parameters of the *Transfer* () function call. The syntax of the function call is given as follows:

int Transfer (*int ProblemIndex*, *int SplitIndex*, *char* TransferFile*, *int InfoType*, *void* Info*, *int SpaceX*, *int SpaceY*, *int SpaceZ*, *int Store*, *char* OutputPipe*)

Transfer () transfers the program code for a worker process and furnishes it with initial data. The information consists of the number of mobile grid points in the three dimensional space. *OutputPipe* is the name of the *Mobile Distributed Pipe* to which the worker process writes its results.

E. Information Gathering by Worker Processes

The anonymously migrated worker processes are collected by the mobile local coordinator *mlc* and the program code is compiled, which spawns the worker processes. A worker process initializes its data structures to hold and collect the initial data. *GetTaskMobGridDetails* () is used to determine the size of the initial data and for the initialization of the data structures. The worker process collects the initial data by invoking the function call

TABLE I
FUNCTION CALLS IN DOMAIN DECOMPOSITION

Data Type	Function	Parameters	Description
int	SendMobGridDetails ()	int ProblemIndex, int SpaceX, int SpaceY, int SpaceZ, int Store, int SplitOrientation	It sends the mobile grid details of a MGCP to the system. Problem Index is the index by which the system identifies a MGCP. SpaceX, SpaceY, and SpaceZ are the number of mobile grids in the three dimensional space. Store represents the number of former time slices stored.
int	EstimateSplitsCount ()	int ProblemIndex	It collects the number of worker processes for the MGCP denoted by Problem Index.
int	GetSplitDetails ()	int ProblemIndex, int SplitIndex, int* StartMobGrid, int* TotalMobGrid	It gathers the details regarding the split of the problem. The starting mobile grid of the subdomain and the total number of mobile grids in the subdomain are stored at the addresses pointed by StartMobGrid and TotalMobGrid respectively.

TABLE II
FUNCTION CALLS IN INFORMATION GATHERING BY WORKER PROCESSES

Data Type	Function	Parameters	Description
int	GetTaskMobGridDetails ()	int* SpaceX, int* SpaceY, int* SpaceZ, int* Store	It gives the number of mobile grid points in the three dimensional space.
int	GetTaskInfo ()	void* Info, int SpaceX, int* SpaceY, int* SpaceZ, int* Store	It saves the first data matrix at the address pointed by Info.
int	GetTaskMachineDetails ()	int* MachineName, char* OutputPipeName	It saves the location of the worker process in the MGCP at the address pointed by MachineName.

GetTaskInfo (). The function call *GetTaskMachineDetails* () determines the name of the *OutputPipe* to be opened and the position of the worker process. During runtime, the system provides information corresponding to individual worker processes. The syntax and semantics of the function calls are given in TABLE II.

F. Transparent Communication

Each worker process has to communicate with its surrounding worker process to exchange boundary values. Since the worker processes are transferred to anonymous nodes, a worker process will not know the neighboring worker process' location [11]. The number of neighbors of a worker process depends on the position of the worker process. Hence, the number of MDP to be opened by theworker process cannot be estimated until runtime. A worker process collects the information about its neighbors and the number of MDP to be opened at runtime and it is accomplished by the function calls given in TABLE III.

G. Design Issues

The performance of the MDP scheme depends on the mobility of the nodes and the node density. The number of nodes entering a mobile cell simultaneously should be limited else nodes might not be captured correctly by the *cc*. The connectivity of the mobile nodes with the *cc* is limited by a threshold radius, beyond which the mobile node leaves the mobile cell. The *mlc* and master/worker process are connected by Unix domain socket connections, and the connections between *sc*, *cc*, *and mlc* are TCP connections.

The parallel computing on mobile clusters is subject to several key issues such as, mobility of nodes, connectivity of mobile nodes, transmission time, uneven nodal distribution, uneven load in the network, and difference in performances of node.

1) Handoff Process

When a mobile node moves from one mobile cell to another, the transition process is termed as *handoff*. During handoff, the channel resources should be managed to preserve the connectivity of the network. The channel used in the former cell may not be reusable in the new cell because of low signal strength, or co-channel or adjacent channel interference. So the transiting mobile gets isolated from the rest of the mobile cluster. When the new channel has not be allocated to the mobile node within a short time span, the messages transmitted in the network will get delay, leading to retransmission of data.

The transmission time can be managed and the retransmission of data can be avoided by using a topology based management. Two types of topology of nodes are used such as, *ring topology* and *tree topology*, to solve the handoff issue. The handoff issue was not properly addressed in the existing Moset [6] model, but it is efficiently handled by the topology scheme introduced in the proposed MDP model.

2) Load Balancing

The details regarding the load and availability of nodes in the network are gathered by the system, so that the optimum number of worker processes to be implemented and their individual granularities can be estimated. The processing power of individual elements is also used for the load balancing in a heterogeneous group of nodes.

3) Disconnectivity of mobile nodes

Timers are maintained for detecting the disconnectivity of the mobile nodes from its associated cell. A node when it does not return within the stipulated time set in the timer, then the sub-process is re-submitted to some other node. In the case of mobile nodes, the co-coordinator (*cc*) acts as the timer. When there is a failure in the resubmission of the sub-process to some other mobile node, the *cc* itself will execute the sub-process. The mobile nodes may get disconnected from the cell after the execution of the sub-process and return to the cell before the timer timeouts. In this case, the *cc* would be able to decide the failure of mobile nodes under the stipulated time of the timer. This ensures the fault tolerance of the network.

TABLE III
FUNCTION CALLS IN TRANSPARENT COMMUNICATION

Data Type	Function	Parameters		Description
int	GetTaskOpenPipeIdentities ()	char** PipeIdentities, ModeOfAccess, NoOfInitiatedPipes	int* int	It furnishes the names, mode of access and number of MDP to be initiated.
int	GetTaskNoOfPipesToInitiate ()	int* NoOfInitiatedPipes		It saves the number of MDP to be initiated at the address pointed by NoOfInitiatedPipes

4) Bandwidth

The mobile nodes are characterized by high fluctuations in the network bandwidth, depending on whether it is a static node or a mobile node, and on the type of connection in the present cell. The MDP model differentiates the type of connectivity and provides flexibility in terms of task size and network bandwidth.

3. DESIGN AND IMPLEMENTATION OF MOBILE DISTRIBUTED PIPES

A. Runtime Support

The runtime support comprises a mobile local coordinator (*mlc*) daemon executing on each node involving in parallel computation, a co-coordinator (*cc*) daemon, and a system coordinator (*sc*) daemon on a designated node.

B. Mobile Local Coordinator

The *mlc* executes on each mobile node that involves in parallel computation, the static local coordinator (*slc*) executes on each static node that involves in parallel computation. The *mlc* services requests from the user processes (*up*). The *mlc* maintains two tables to support the MDP services, namely, the *User Processes Blocked for Write Table* (UPBWT) and *User Process Information Table* (UPIT) [11]. The UPIT maintains information relative to the *up* registered with the *mlc* on its node. The UPBWT tracks the processes which have opened a MDP in write mode alone. The writing process is blocked until the MDP is opened by some other process in read mode. The *mlc* uses the table to inform the blocked processes when any other process opens the MDP in read mode.

The *mlc* uses a *MobileGrid Computation Task Submitted Table* (MGCTST) for supporting mobile grid computations. The *mlc* services the task requests using the MGCTST. The table is referenced by the process id of the task. The table is updated under two conditions, i.e. when an already submitted task terminates or when a new task is submitted. The *mlc* forwards the information to the *cc* when the service needs additional parameters.

The FSM of the *mlc* is given in Fig. 2. In the **INIT** state, *mlc* cleans the secondary system files and initializes its data structures. The *mlc* establishes a TCP connection with the *cc* and registers with the *cc*. In the **LISTEN** state, the *mlc* waits for the messages from the *cc* or any user process. When a message is received from the *cc*, it changes its state to **SC Msg RXD** and services the message. When a message is received from a user process, it changes its state to **UP Msg RXD** and services the request.

The initial communication the m*lc* and a process established through a known common channel, which is necessary for a user process to register with the *mlc*. User processes which register with the *mlc* are specified with unique communication channels for subsequent communication.

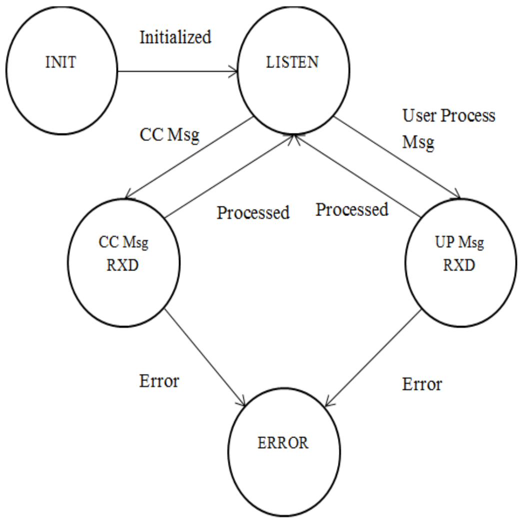

Fig. 2. FSM of Mobile Local Coordinator (MLC).

C. Co-Coordinator

The co-coordinator represents a system coordinator relative to mobile nodes (MN) which are within the specified cell [6]. Any MNwithin the coverage area of the mobile cell registers a group of computing elements to the *cc* executing on that mobile cell. The co-coordinator collects the group of computing elements and registers with the *sc*. The multicasting of the dataset and the history of execution are maintained by the co-coordinator. The MLCT (*Mobile Local Coordinator Table*) tracks the *mlc* in the system. The table is updated when there is a change of *mlc* in the mobile cell.

The mobility of the MN is also monitored by the *cc*. When a MN moves out of the mobile cell and enters another mobile cell, then the new mobile cell informs the *cc* of the old mobile cell through handoff. The information among the *cc* is exchanged when the new mobile cell already has the *cc* daemon executing. When the new mobile cell does not execute the *cc* daemon, the *cc* gets registered with the *sc* and executes the *cc*.

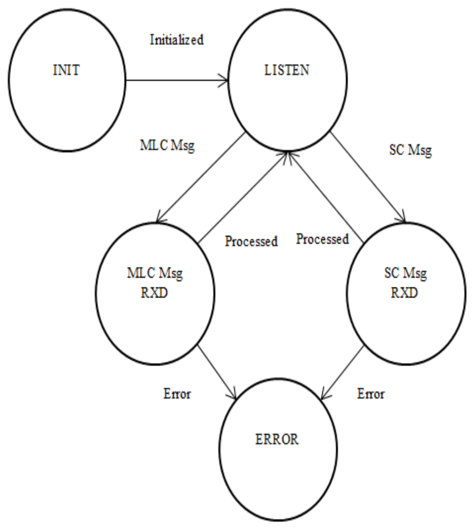

Fig. 3. FSM of Co-Coordinator (CC).

The FSM of the *cc* is given in Fig. 3. In the **INIT** state, *cc* cleans the secondary system files and initializes its data structures. In the **LISTEN** state, the *cc* waits for the messages from the *mlc* or *sc*. When a message is received from the *mlc*, it changes its state to **MLC Msg RXD** and services the message. When a message is received from the *sc*, it changes its state to **SC Msg RXD** and services the request.

D. *System Coordinator*

The sc coordinates the group of co-coordinators from the various mobile cells. The sc senses the failure of the mobile nodes when the mlc fails. The sc tracks the individual co-coordinators and establishes the communication between them. TCP sockets are used to connect sc with the individual by including static node in the distributed processing. The static node establishes the computing elements based on its capabilities through slc. The static node also includes the cc which establishes a group of computing elements represents the mobile nodes which are within its cell. The sc groups these computing elements into cluster-subgroups and tracks the total number of computing elements under each subgroup. The sc allocates each computing element a unique ID affiliated with each group subscribed by it.

The *sc* comprises of three tables, namely, the *Mobile Distributed Pipes Table* (MDPT), the *MobileGrid Computation Task Table* (MGCTT), and the *MobileGrid Computation Problem Table* (MGCPT) [11]. The MDPT tracks the MDP channels and is updated whenever a MDP is created, enabled, disabled, or deleted. When a process enables a MDP before the pipe is enabled for reading, the relative *mlc* information is stored in the table. The MGCPT maintains the information relative to the MGCP that is submitted to the *sc*. The MGCPT is updated when a work is completed or when a work is submitted to the *sc*. The MGCTT maintains information relative to individual tasks constituting the MGCP. The MGCTT is updated when a task begins execution, when the problem is subdivided into tasks, or when a task terminates.

The FSM of *sc* is given in Fig. 4. In the **INIT** state, the *sc* cleans the secondary system files and initializes its data structures. In the **LISTEN** state, the *sc* selects the connection requests from the co-coordinators. The *sc* registers the *cc* with the system when a connection request from a *cc* is received, and a TCP socket connection is established between them. The *sc* listens for messages from the registered *mlcs* on unique channels, and continues to listen to new requests for connection. When a message from a *cc* is received, it changes its state to **CC Msg RXD** and executes the message. The *sc* returns to the **LISTEN** state when the message is processed.

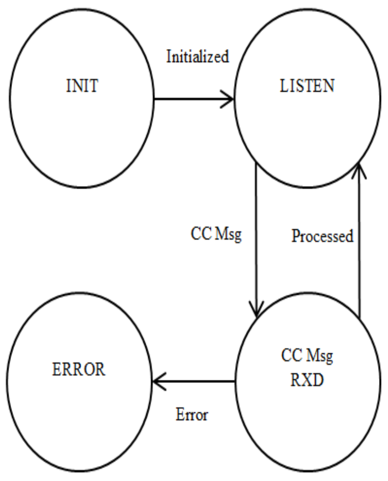

Fig. 4. FSM of System Coordinator (SC).

E. *Function Library*

The library of functions comprises various services to support the MDP model of IMGC and the location transparent communication via MDP [11]. The variants of the function calls support communication across different architectures by external data representation. The function library is constructed over UNIX and TCP domain stream protocol.

A TCP socket is created and a new message sequence is initiated by giving a message to *sc* through *cc*, for an open request in *WriteMode*. The message comprises the TCP socket descriptor, mode of access, and process index of the requesting process. The MDPT is updated with this information by the *sc*. When the MDP is already opened by another process in *Read Mode*, the *sc* returns the information to the caller. When the MDP is not opened for reading, the updating of MDPT and UPBWT are performed at *sc* and *mlc* respectively.

A TCP socket is created and combined with a local port for an open request in *Read Mode*. A message is given to the *sc* to update the MDPT with the port number, process index, mode of access, and TCP socket descriptor. The library of functions along with their description is given in TABLE IV.

TABLE IV
BASIC SERVICES OF MDP

Data Type	Function	Parameters	Description
int	EstablishMDP ()	char* PipeIdentity	It sends a message to the sc through the cc and mlc executing on its machine. The sc establishes the MDP if there is no other channel with the same identity and makes an entry in MDPT
int	InitiateMDP ()	char* PipeIdentity, int ModeOfAccess	It sends a message to the sc through the cc and mlc executing on its machine with the name of MDP as a parameter. The sc completes the message sequence by communicating with the user process about the creation of the MDP.
int	GetMDP ()	int MDPDescriptor, char* Memory, int MemorySize	It gets the information from the socket descriptor for the MDPDescriptor. The MDPDescriptor returned is the real socket descriptor. This function call executes message sizes above the sizes of TCP messages.
int	PutMDP ()	int MDPDescriptor, char* Memory, int MemorySize	It puts the information to the socket descriptor for the MDPDescriptor. The MDPDescriptor returned is the real socket descriptor. This function call executes message sizes above the sizes of TCP messages.
int	EndMDP ()	int MDPDescriptor	It ends the socket descriptor for the MDPDescriptor. This function call sends a message to the cc through mlc with the name of MDP and ProcessIndex as parameters. This updates the MDPT at sc.
int	RemoveMDP ()	char* PipeIdentity	It sends a message to the sc through the cc and mlc with the name of the MDP as an argument. The sc removes the relative element in MDPT and returns the removal status to the function call.

TABLE V
EXTENDED SERVICES OF IMGC

Data Type	Function	Parameters	Description
int	EstablishMobGridComputationProblem ()		It sends a message to sc through cc and mlc. The sc produces an element for the problem in MGCPT and returns ProblemIndex
int	SendMobGridDetails ()	int ProblemIndex, int SpaceX, int SpaceY, int SpaceZ, int Store, int SplitOrientation	It sends a message to sc through cc and mlc. The message contains the parameters of the function call. The update of MGCPT is performed by sc and updated status is returned.
int	EstimateSplitsCount ()	int ProblemIndex	It sends a message to sc through cc and mlc. The load details of all machines are collected from relative mlcs. This information is used to split the problem by the sc. The sc produces a new element in MGCTT to save the information about the division and returns the splits count.
int	GetSplitDetails ()	int ProblemIndex, int SplitIndex, int *StartMobGrid, int *TotalMobGrid	It sends a message to sc through cc and mlc. The values from the MGCTT are gathered by the sc and the initial and total number of mobile grids is returned.
int	Transfer ()	int ProblemIndex, int SplitIndex, char *TransferFile, int InfoType, void *Info, int SpaceX, int SpaceY, int SpaceZ, int Store, char *OutputPipe	It sends a message to sc through cc and mlc with its ProblemIndex and SplitIndex. The values of mlcs from MGCPT and MGCTT are gathered by the sc and messages are sent to the mlcs. Each mlc establishes a TCP socket, joins the TCP socket with a local port, and listens to it. The port numbers are given to the sc and it passes the gathered information to the mlc which

			started the transfer. The mlc which started the transfer establishes a TCP connection and transfers the data, code, and output MDP name to the other mlc. The MGCTST of the transferred mlc is updated this information. The TCP connection is disabled when the transfer is finished.
int	EndMobGridComputationProblem ()	int ProblemIndex	It sends a message to sc through cc and mlc with ProblemIndex. The sc removes the corresponding element from the MGCPT.
int	EstablishMobGridComputationTask ()		It sends a message to sc through cc and mlc. The new task element is updated into the MGCTT and TaskIndex is returned.
int	GetTaskMobGridDetails ()	int *SpaceX, int *SpaceY, int *Store	It sends a message to sc through cc and mlc. The message comprises of the process index of the task. The mlc collects the corresponding information from the MGCTST.
int	GetTaskMachineDetails ()	int *MachineName, char *OutputPipeName	It sends a message to sc through cc and mlc. The subdomain for which the task depends on the mlc is observed. The position of this subdomain is returned by the sc. The mlc returns the subdomain information and the OutputPipeName to the task.
int	GetTaskInfo ()	void *Info, int SpaceX, int SpaceY, int Store	It sends a message to mlc. The mlc observes the information from MGCTST and the information is returned to the task.
int	GetTaskNoOfPipesToInitiate ()	int *NoOfInitiatedPipes	It sends a message to sc through cc and mlc. The process index is appended with the message. The sc observes the information from the MGCPT and the number of MDP to be initiated by the task is returned.
int	GetTaskInitiatedPipeIdentities ()	char **PipeIdentities, int *ModeOfAccess, int NoOfInitiatedPipes	It sends a message to sc through cc and mlc. The process index is appended with the message. The identities of the MDP to be initiated and their mode of access are returned.
int	EndMobGridComputationTask	int TaskIndex	It sends a message to mlc. The mlc removes the corresponding element from the MGCTST and the message is forwarded to the sc through cc. The sc updates the values of MGCPT in response to the message.

F. Overhead of Interfaces and Extended Services for IMGC

The overhead of a function call is due to the message sequences initiated by the function call [11]. The function calls *EstablishMDP* (), *EndMDP* (), and *RemoveMDP* () forms a message from the *mlc* to the *sc*, a message to the *mlc* on the mobile node, a message from the *lc* back to the *up*, and a reply from the *sc* to the *lc*. The normal size of exchanged data packets is approximately 100 bytes. These function calls are used only once during the lifespan of a MDP. The function calls *GetMDP* () and *PutMDP* () are changed to the current system call, which do not result in any overheads. These function calls are used multiple times during the lifespan of a MDP. The extended services for an IMGC along with their description are given in TABLE V.

G. Variations from DP, GDP and Moset

The following issues were not solved in DP, GDP and Moset: handoff, load balancing, disconnectivity of mobile nodes, and fluctuations in bandwidth. The ring topology involves each mobile node with exactly two neighbors, which forms a planar structure. This structure arranges the mobile nodes arranged in a mobile grid of rows and columns. The example ring topology architecture is shown in Fig. 5.

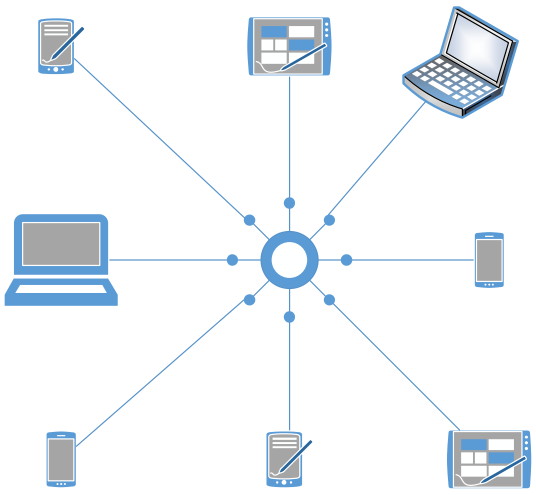
Fig. 5. Example Ring Topology

The tree topology composes of a hierarchical structure of node levels which forms a tree. The lower level of the tree contains more nodes than the previous level of the tree. An example tree topology is shown in Fig. 6. The fragmented topology is reconstructed prior to the destruction of the pre-defined topology during the roaming of a mobile node. The reconstruction of the fragmented topology is performed by choosing a new node as an option to the migrated node. The time spent on the reconstruction should be less for the satisfactory performance of the mobile cluster computing.

The various machines compose of divided sub-domains of computations. The load ratio on each machine determines the granularity of each subdomain. The load can be balanced by ignoring the machines for which the load indices cross a designated value. DP is not able to account for disconnection during handoff process, which is solved in MDP model using the reconstruction of fragmented topologies.

The existing GDP [7] and Moset [6] models cannot handle a heterogeneous and varying load conditions, which is overcome in the proposed MDP model by issuing threshold values (load indices) and neglecting the machines which cross the threshold value. The existing GDP and Moset [6] models does not support flexible wireless bandwidth, which is overcome in the proposed MDP model by distinguishing the type of connectivity in a cell of the mobile cluster.

The disconnectivity issue was not properly addressed in the existing Moset [6] model, but it is efficiently handled by the timer scheme introduced in the proposed MDP model.

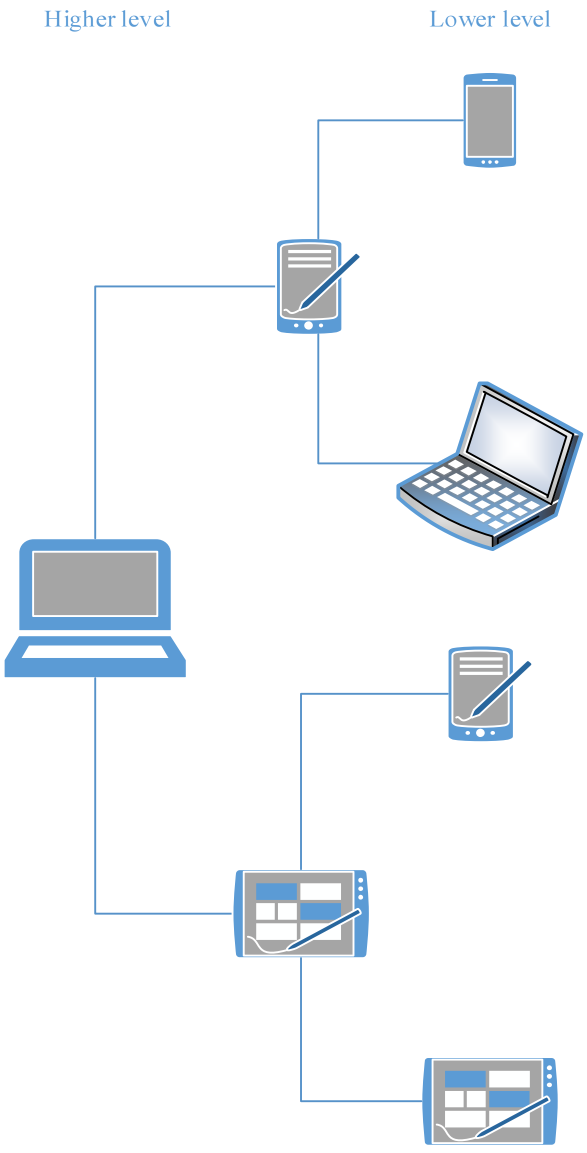

Fig. 6. Example Tree Topology

3. CASE STUDY

A variety of engineering applications are based on the principle of iterative mobile grid computation (IMGC). These applications possess a pattern in their process interaction. A distributed image rendering application is considered for the case study.

H. *Distributed Image Rendering*

Each mobile node (MN) comprises a client and a daemon. The clients submit the tasks to the MN for the distributed processing, while the daemons the computing elements at the MNs, that process a part of the submitted task along with other daemons [6].

The image rendering application is based on images obtained by CT scan. A CT scan image contains information on a transverse plane only. The CT scanners generate 3-D parallel plane stack images. Each stack consists of a sequence of X-ray absorption coefficients. The data sets can be viewed as a 3-D field due to the availability of parallel plane image stacks. The stack is converted into a single image by ray-casting and volume rendering. A ray-casting algorithm is employed to cast the parallel rays from the observer into the volume. The progressive attenuation at each point along the ray due to particle fields is computed. Simultaneously, the light scattered in the direction of the eye from the light source is also estimated at each point. The CT scan of a human head with the skull partially removed is taken as the input data.

I. *MDP Model of Computation*

The MDP model consists of a master process for each mobile cell and several worker processes. The master process is the initiator for the mobile grid computation. The master process coordinates the parallel computation by simultaneous communication with the system. A worker process involves calculations over a subdomain. The grain size of the subdomain for a worker process depends on the load ratio.

J. *Master Process*

The master process initializes the MGCP with the system and initiates the parallel computation [11]. The system decides the number of worker process, their associated granularities, and the machines on which they process. The master process gathers the information regarding each split from the system, which is used to create data packets for individual worker processes. *Output Pipe* for each worker process is created and transferred. The master process opens the *Output Pipes* and waits for the outputs. The *Output Pipes* are closed and removed after the outputs are collected. A sample code of the master process is given in Pseudo Code 2.

Pseudo Code 2: Sample code of the master process

```
int main ()
{
  …
  ProblemIndex =
EstablishMobGridComputationProblem ();
…
SendMobGridDetails (ProblemIndex,
  RowsCount, ColsCount, 1,
  DIVIDE_COLUMNS);
…
SplitsCount = EstimateSplitsCount
  (ProblemIndex);
…
for (SplitIndex =0; SplitIndex < SplitsCount;
  SplitIndex ++)
{
   GetSplitDetails (ProblemIndex, SplitIndex,
     &StartMobGrid, &TotalMobGrid);
   …
   CreatePacket (Matrix, RowsCount,
     StartMobGrid, TotalMobGrid, 1,
     DIVIDE_COLUMN, Data);
   …
   OutputPipe =
     HostName.“Output”.ProblemIndex.SplitNo;
   EstablishMDP (OutputPipe);
   Transfer (ProblemIndex, SplitNo,
     TransferFile, FLOAT_TYPE, Data,
     OutputPipe);
}
…
for (SplitIndex = 0; SplitIndex < SplitsCount;
  SplitIndex ++)
{
  OutputPipe =
  HostName.”Output”.ProblemIndex.SplitID;
  PipeFd[SplitIndex] = InitiateMDP
  (OutputPipe, GET_MODE);
}
for (SplitIndex = 0; SplitIndex < SplitsCount;
  SplitIndex ++)
{
   GetSignChar (PipeFd[SplitIndex],
        &ExtraOutput, sizeof (ExtraOutput),
        CONVERT_XDR);
      while (ExtraOutput)
      {
        GetFloat (PipeFd[SplitIndex], &Output,
          sizeof (Output), CONVERT_XDR);
        GetSignChar (PipeFd[SplitIndex],
          &ExtraOutput, sizeof (ExtraOutput),
          CONVERT_XDR);
      }
    }
    for (SplitIndex = 0; SplitIndex < SplitsCount;
      SplitIndex ++)
    {
      EndMDP (PipeFd[SplitIndex]);
      OutputPipe =
      HostName."Output".ProblemIndex.SplitIndex;
      RemoveMDP(OutputPipe);
    }
    EndMobGridComputationProblem
      (ProblemIndex);
    return 1;
}
```

K. Worker Process

The worker processes are transferred to anonymous remote nodes for processing [11]. A worker process initializes itself with its local *mlc*. The worker processes collect the information corresponding to the allotted subdomain. A sample code of the master process is given in Pseudo Code 3.

Pseudo Code 3: Sample code of the worker process

```
  int main ()
  {
    …
    TaskIndex =
EstablishMobGridComputationTask ();
    …
    GetTaskMobGridDetails (&RowsCount,
      &ColsCount, &Depth, &SplitOrientation);
    GetTaskMachineDetails (&MachineName,
      OutputPipeName);
    …
    InitiateMDP (OutputPipeName,
    PIPE_READ);
    …
    GetTaskInfo (Info);
    …
GetTaskNoOfPipesToBeEstablishedAndInitiat
  ed (&NoOfEstablishPipe,
  &NoOfInitiatedPipes);
…
GetTaskEstablishPipeIdentities
  (EstablishPipes, NoOfEstablishPipe);
for (PipeIndex = 0; PipeIndex <
  NoOfEstablishPipes; PipeIndex++)
{
  EstablishMDP (EstablishPipes[PipeIndex]);
}
…
GetTaskInitiatedPipeIdentities (InitiatedPipes,
  ModeOfAccess, NoOfInitiatedPipes);
for (PipeIndex = 0; PipeIndex
  <NoOfInitiatedPipes; PipeIndex ++)
{
  PipeFd[PipeIndex] = InitiateMDP
    (InitiatedPipes[PipeIndex],
    ModeOfAccess[PipeIndex]);
}
…
if (MachineName != FINAL_MGCTM)
{
  PutFloat(SucceedingMachineWd,
    &FormerTimeFormerMobGrid, sizeof
    (float), 1, CONVERT_XDR);
}
if (MachineName != INITIAL_MGCTM)
{
  PutFloat (SucceedingMachineWd,
    &FormerTimeSucceedingMobGrid,
    sizeof (float), 1, CONVERT_XDR);

}
for (Time = 0; Time<TMax; Time ++)
{
  if (MachineName != INITIAL_MGCTM)
  {
    GetFloat (FormerMachineRd,
      &FormerTimeFormerMobGrid, sizeof
      (float), 1, CONVERT_XDR);
    PutFloat (FormerMachineWd,
      &FormerTimeSucceedingMobGrid,
      sizeof (float), 1, CONVERT_XDR);
  }
  else
  {
    …
  }
  for (MobGrid = StartMobGrid; MobGrid <
  TotalMobGrid -1; MobGrid ++)
  {
        …
      }
      if (MachineName != LAST_MGCTM)
      {
        GetFloat (SucceedingMachineRd,
          &FormerTimeSucceedingMobGrid,
          sizeof (float), 1, CONVERT_XDR);
        …
        PutFloat (SucceedingMachineWd,
          &FormerTimeFormerMobGrid, sizeof
          (float), 1, CONVERT_XDR);
      }
      else
      {
        …
      }
    }
    for (PipeIndex = 0; PipeIndex
      <NoOfInitiatedPipes; SplitIndex ++)
    {
      EndMDP (PipeFd[PipeIndex]);
    }
    for (PipeIndex = 0; PipeIndex <
      NoOfEstablishPipes; SplitIndex ++)
    {
      RemoveMDP (EstablishPipes[PipeIndex]);
    }
    EndMobGridComputationTask (TaskIndex);
    return 1;
  }
```

4. Background Work

Some of the techniques related to parallel computing and distributed computing are discussed in this section.

A. Integration of Mobile Hosts into the Mobile Grid

A mobile grid is constructed by the integration of the grid computing paradigm under service constitution technology and mobile computing paradigm [1]. It combines the powerful features of the grid computing capability and omnipresent accessibility of mobile distributed system. This technique consists of a distributed system model where the resources are organized as P2P (Peer to Peer) model. The resources of the hosts are visualized as services. This system is implemented as a shared distributed object space and in0creases the information processing capability and service sharing.

B. Tool for Distributed Computing

OptimalGrid is a new pattern of middleware for computation of larger problems in a distributed computing environment [8]. OptimalGrid automates the problem partitioning, dynamic redeployment, runtime management, and deployment of problem.

C. Process Interaction in Distributed Computations

The distributed computations are programs which communicate with the passing of the messages [20]. These programs generally process on network architectures such as NOW or distributed parallel machines. Some of the models for process interaction in distributed computations are network of filters, heartbeat algorithms, broadcast algorithms, decentralized servers, token-passing algorithms, bag of tasks, and probe/echo algorithms. These models involve parallel sorting, computing network topology, and termination detection.

D. Location-independent communication methods

The proposed MDP is also compared with other location-independent (LI) communication methods between mobile agents [21, 22]. Nomadic Pict language is a distributed infrastructure for mobile computations employing LI intertask communication [21]. Low-level Nomadic Pict enables agent formation, transfer of agents between machines, communication of asynchronous messages between agents, and fine cooperation. High-level Nomadic Pict enables the LI communication.

The low-level translation is user-defined via an arbitrary infrastructure. This language is constructed relative to asynchronous messaging. The TCP connections are formed on demand, but the program can also use a layer which enables authentic communication on the top of UDP. The messages are transparently delivered regardless of machine disconnection and agent migration. The infrastructure encoding consists of three sections, namely, a primary level component, an auxiliary compositional translation, and a phrase-by-phrase definition.

LI techniques permit communication with a mobile agent irrespective of the migrations [22]. The implementation of these communication methods requires soft distributed infrastructure algorithms. LI communication enables the modules to communicate without explicitly monitoring each other's movements.

Nomadic Pict [21] has a few limitations in terms of space and agent migration. The agent migration process is not transparent and a migration process is blocked until the machine is back in the network. Due to lack of space, a machine disconnection would block the agent migrations and communications via the query server. The buffering in the query server due to machine disconnection is an impractical option. The proposed MDP model is transparent and allows real-time migration of the mobile agents. The agent migration process is never blocked in the MDP model.

E. *Recent parallel computing methods*

Some of the other recent parallel and distributed computing systems include real-time aggregation system [23], language virtualization [24], and agent based parallel computing [25].The real-time aggregation system is used for large-scale parallel and distributed systems. A recent load balancing technique in randomly partitioned cluster services involve requests across a cluster of backend servers [26]. This was used to avoid the performance bottlenecks in large-scale cloud computing services. The advantage over existing communication technologies like Mobile IP (MIP) and Wi-Fi would be the stability in handoff when handling large clusters of data, where the existing methods have a smaller limited range than MDP model.

4. PERFORMANCE ANALYSIS

The round trip time (RTT) of communication over the network is around 0.3ms to a few milliseconds. Generally, the average RTT is less than 0.15ms. The performance analysis displays the speedup attained by the parallel execution of the problem and decreased memory requirements. The communication overhead can be prohibited to attain a linear to super-linear speedup. The analysis involves memory requirements, consequences of parallel execution, synchronization delay among sub-processes, and effects of load on speedup and execution time. *Speedup* is the ratio of sequential execution time to the parallel execution time. The distributed image rendering application is considered for the performance analysis. The proposed method is analyzed and compared with existing methods Moset [6] and DP [11].

A. *Memory Requirements*

The task completion time with respect to the memory requirements is analyzed and compared with DP [11]. The comparative results are shown in Fig. 7. The results show that the task completion time for the given models increase gradually, with the two models almost following a similar pattern. But, the completion time for the MDP model is about 40% lesser than the DP model as the number of grid computations approach 70.

The effect of process memory size on execution time is analyzed for the MDP and Moset [6], considering three mobile nodes. The comparative results are shown in Fig. 8. The results show that the task execution time for the given models is decreasing gradually with little constancy in between. But, the task execution time for the MDP model is about 35% lesser than the DP model as the memory size approach 225 MB.

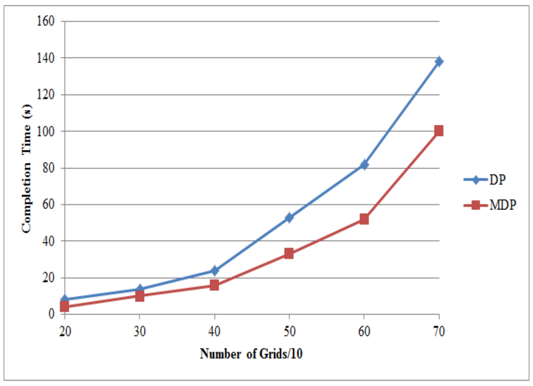

Fig. 7. Memory requirements of MDP and DP.

The memory required in the existing DP [11] and Moset [6] models are large due to the higher task completion time and execution time. These parameters are increased in the proposed MDP model by increasing the resolution of the sub-tasks division and thus decreasing the task completion time and execution time.

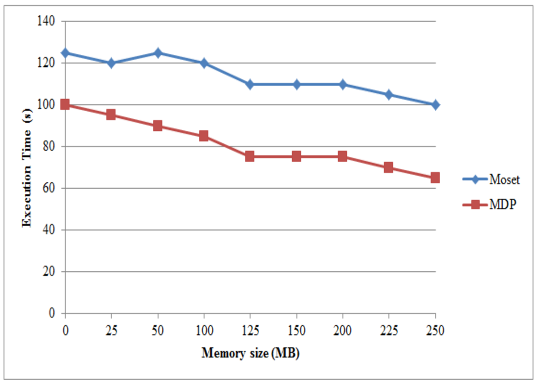

Fig. 8. Effect of process memory size for MDP and Moset.

The memory usage of a framework for mobile systems (Misco) [27] and MDP is analyzed and compared in Fig. 9. It is observed that MDP occupies lesser memory than Misco framework. Misco occupies average memory of 40MB whereas MDP occupies only average memory of 32MB.

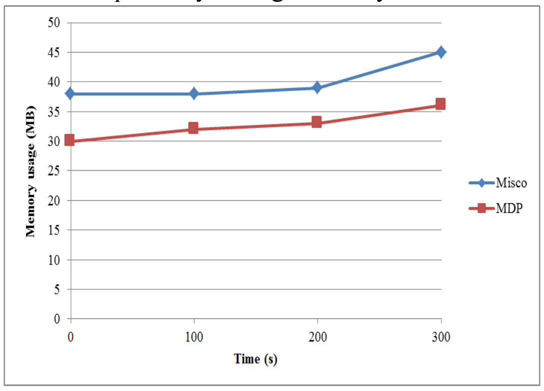

Fig. 9. Memory usage for MDP and Misco.

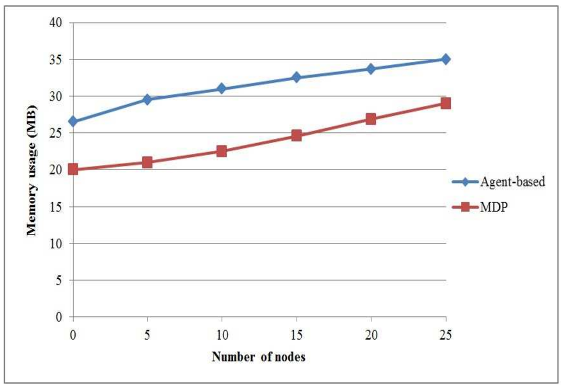

Fig. 10. Memory consumption for MDP and agent-based parallel computing.

The memory consumption of agent based parallel computing [25] is analyzed and compared with that of MDP relative to number of nodes in Fig. 10. It is observed that MDP occupies lesser memory than agent-based parallel computing. The former occupies average memory of 32MB whereas the latter occupies only average memory of 24.5MB.

B. Consequences of Parallel Execution

The grain size of a subtask is the count of the mobile grid points for the subtask. The task time is the sum of the synchronization delay by the subtask and the actual CPU time. Synchronization time is the total time the subtask takes to receive data from the surrounding subtasks and is given by the sum of the individual synchronization delay. The comparative results for MDP and DP [11], with equal load division for five nodes are given in Fig. 11, in terms of task time, synchronization time, and speedup. The synchronization time, task time for the MDP model is only half of that of the DP model.

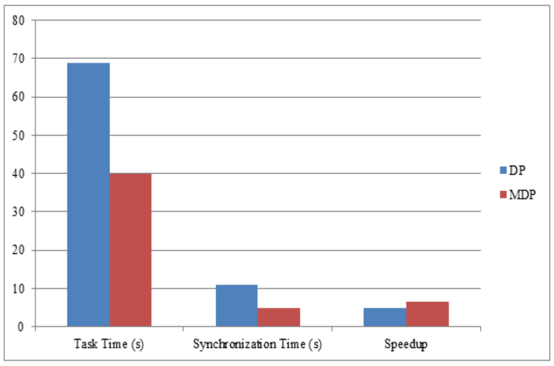

Fig. 11. Consequence of parallel execution for 5 nodes using DP and MDP.

The comparative results for MDP and DP [11], with larger tasks for 20 nodes are given in Fig. 12, in terms of task time and speedup. In the case of 20 nodes, the task time for the MDP model is about 77% of the DP model. The rate of speedup achieved due to parallel execution is higher in MDP model (25.451) compared to the DP model (18.791).

Journal of Theoretical and Applied Information Technology
31st March 2014. Vol. 61 No.3
© 2005 - 2014 JATIT & LLS. All rights reserved·
ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

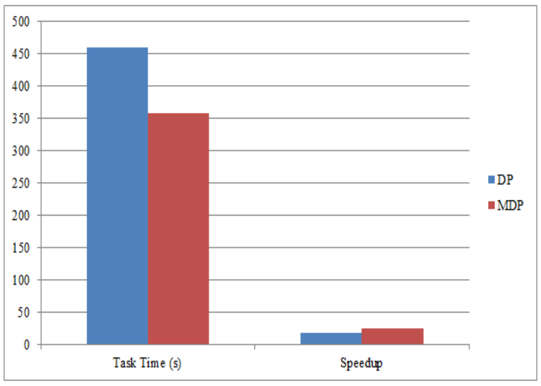

Fig. 12. Consequence of parallel execution for 20 nodes using DP and MDP.

The speedup in the existing DP [11] and Moset [6] models are less due to the higher parallel execution time. This is increased in the proposed MDP model by increasing the number of parallel executions per machine and thus increasing the speedup.

The speedup of agent based parallel computing [25] is analyzed and compared with that of MDP relative to number of nodes for a fixed runtime of 60s in Fig. 13.

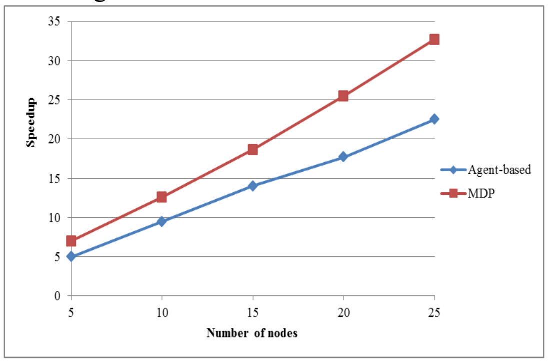

Fig. 13. Speedup for MDP and agent-based parallel computing.

C. Synchronization Delay Relative to Number of Machines

The total synchronization delay of individual subtasks determines the performance saturation and load imbalance. A higher value of synchronization delay for a subtask signifies that the subtask accomplishes its task faster than the neighboring subtasks. When the number of machines participating in the parallel computation increases, the mean grain size decreases. This results in performance saturation beyond which no machine would contribute significantly to the speedup.

The data for a subtask is generated by its neighbor in its last beat of calculation, and so the synchronization time is considerably low for the initial samples. The prohibition of communication overhead attains linear-like speedup and the decreased memory requirements further increase the performance to super-linear speedup.

The progress in total synchronization delay of individual subtask relative to the subtask count is given in Fig. 14. The average synchronization delay varies randomly initially and then after N = 3 there is a linear increase. The average synchronization delay for the MDP model is about 80% of the DP model. The result shows the average of total synchronization delay by the separate subtasks. When the number of subtasks increases, their associated granularity decreases. This increases the synchronization delay for each subtask.

The average synchronization delay in the existing DP [11] model is higher due to the inefficient load balancing. This is overcome in the proposed MDP model by an efficient load balancing scheme which issues threshold values (load indices) and neglects the machines which cross the threshold value.

D. Performance Saturation

Granularity increases when the number of mobile grid points is increased. The computational domain is divided into more subdomains when there is an increase in the number of nodes. Hence, there is a decrease of granularity and increase of synchronization delay of individual subtask.

A saturation point is defined as the value of absolute granularity at which the average synchronization delay exceeds 10% of the average task time. The absolute granularity is anonymous to the performance resilience. The value of absolute granularity with 10 tasks is computed to be around 35ms. The observed values of RTT among the nodes vary from 0.2ms to 3ms with an average of 0.24ms.

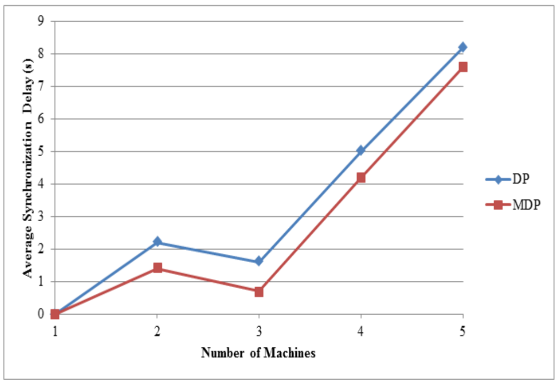

Fig. 14. Average synchronization delay vs. number of machines.

E. Effect of Load on Execution Time

The effect of load on the execution time for various numbers of mobile nodes is analyzed and compared for MDP and Moset [6] models, considering three mobile nodes. The comparative results are given in Fig. 15. The execution time increases initially as the load on the machines is increased for both the models. Later, the execution time decreases on a small scale for the MDP model, but it remains almost for the Moset model. The execution time of the MDP model is about 75% of the Moset model. This shows the efficiency of the MDP model over the Moset model.

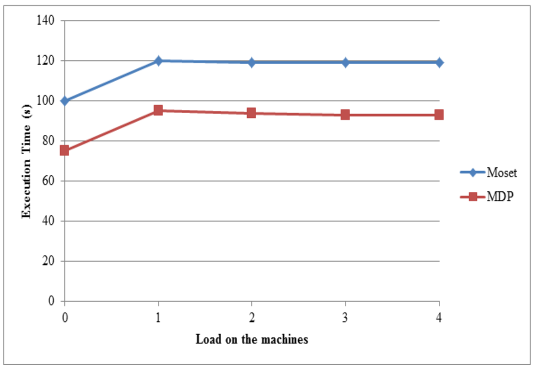

Fig. 15. Effect of load on execution time for MDP and Moset.

The initial increase in the execution time is due to the lag in the adaptability of the models. Once attaining the attaining the adaptability over the varying load on the machines, the execution time remains almost constant. The high execution time initially in the existing Moset [6] model is attributed to the initial value of the granularity of subtasks. This is overcome in the proposed MDP model by increasing the initial value of the granularity of subtasks.

F. Effect of handoff and load balancing

The effects of handoff on various parameters are analyzed. The various parameters are:

1) PDR

PDR is analyzed with respect to different mobility speed of the mobile nodes. The comparative analysis is given in Fig. 16. It is observed MDP possesses higher PDR than Dynamic Clustering-Based VANET [28].

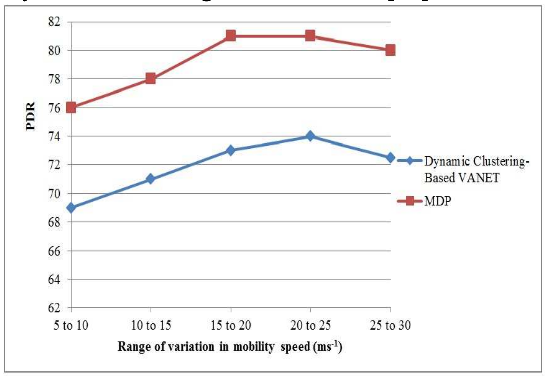

Fig. 16. Effect of handoff on PDR in MDP and Dynamic Clustering-Based VANET.

2) Throughput

Throughput is analyzed with respect to number of clusters. The comparative analysis is given in Fig. 17. It is observed MDP possesses higher throughput than Dynamic Clustering-Based VANET [28] due to effective load balancing.

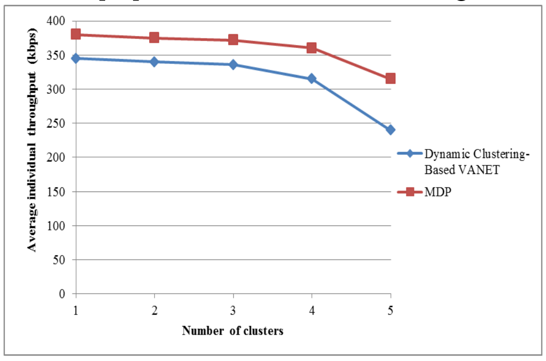

Fig. 17. Effect of load balancing on throughput in MDP and Dynamic Clustering-Based VANET.

3) Packet drop fraction

Packet drop fraction is analyzed with respect to number of clusters. The comparative analysis is given in Fig. 18. It is observed MDP possesses lesser packet drop fraction than Dynamic Clustering-Based VANET [28] due to effective handoff.

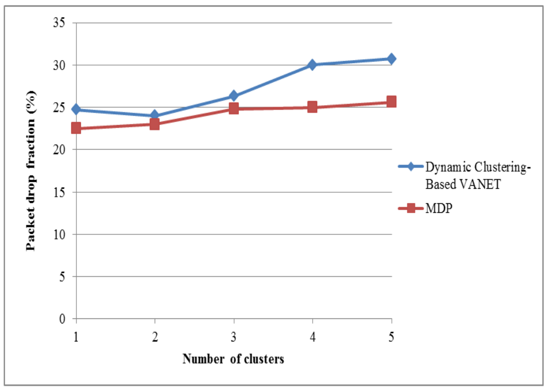

Fig. 18. Effect of handoff on packet drop fraction in MDP and Dynamic Clustering-Based VANET.

5. CONCLUSION

An effective methodology for the computation and communication in a mobile cluster is proposed using mobile distributed pipes (MDP). This technique shows better performance compared to the present techniques like DP [11], Dynamic Clustering-Based VANET [28], and Moset [6] in terms of synchronization time, speedup, task time, memory requirements, throughput, packet drop fraction, and packet delivery ratio (PDR). The future work of parallel computing in mobile cluster involves extending distributed computing to a larger span of mobile grids.

REFERENCES

[1] M. Mohamed, "An object based paradigm for integration of mobile hosts into grid," *International Journal of Next-Generation Computing*, 2011, 2, pp. 1-23.

[2] M. A. M. Mohamed, *et al.*, "EOMP: an exactly once multicast protocol for distributed mobile systems," *International Journal of Parallel, Emergent and Distributed Systems*, 2010, 25(3), pp. 183-207.

[3] S. Kailasam, *et al.*, "Arogyasree: an enhanced grid-based approach to mobile telemedicine," *Int. J. Telemedicine Appl.*, 2010, 2010, pp. 1-11.

[4] M. Mohamed, "Communication and Computing Paradigm for Distributed Mobile Systems," *Journal on Information Sciences and Computing*, 2007, 1(1), pp. 33-41.

[5] M. M. Mohamed, *et al.*, "Surrogate Object Model: A New Paradigm for Distributed Mobile Systems," in *Proceedings of the 4th International Conference on Information Systems Technology and its Applications (ISTA'2005), May*, 2005, pp. 124-138.

[6] M. A. M. Mohamed, *et al.*, "Moset: An anonymous remote mobile cluster computing paradigm," *Journal of Parallel and Distributed Computing*, 2005, 65(10), pp. 1212-1222.

[7] D. Janakiram, *et al.*, "GDP: A Paradigm for Intertask Communication in Grid Computing Through Distributed Pipes," in *Distributed Computing and Internet Technology*. vol. 3816, G. Chakraborty, Ed., ed: Springer Berlin Heidelberg, 2005, pp. 235-241.

[8] T. J. Lehman and J. H. Kaufman, "OptimalGrid: middleware for automatic deployment of distributed FEM problems on an Internet-based computing grid," in *Cluster Computing, 2003. Proceedings. 2003 IEEE International Conference on*, 2003, pp. 164-171.

[9] K. Hairong, *et al.*, "Iterative grid-based computing using mobile agents," in *Parallel Processing, 2002. Proceedings. International Conference on*, 2002, pp. 109-117.

[10] A. Basit and C.-C. Chang, "Mobile cluster computing using IPV6," in *Ottawa Linux Symposium*, 2002, pp. 31-39.

[11] B. K. Johnson, *et al.*, "DP: a paradigm for anonymous remote, computation and communication for cluster computing," *Parallel and Distributed Systems, IEEE Transactions on*, 2001, 12(10), pp. 1052-1065.

[12] M. Litzkow and M. Solomon, "Supporting checkpointing and process migration outside the UNIX kernel," in *Mobility*, M. Dejan, cacute, D. Frederick, and W. Richard, Eds., ed: ACM Press/Addison-Wesley Publishing Co., 1999, pp. 154-162.

[13] R. K. Joshi and D. J. Ram, "Anonymous remote computing: a paradigm for parallel programming on interconnected workstations," *Software Engineering, IEEE Transactions on*, 1999, 25(1), pp. 75-90.

[14] F. Tandiary, *et al.*, "Batrun: utilizing idle workstations for large scale computing," *Parallel & Distributed Technology: Systems & Applications, IEEE*, 1996, 4(2), pp. 41-48.

[15] T. E. Anderson, *et al.*, "A case for NOW (Networks of Workstations)," *Micro, IEEE*, 1995, 15(1), pp. 54-64.

[16] J. Casas, *et al.*, "Adaptive load migration systems for PVM," presented at the Proceedings of the 1994 ACM/IEEE conference on Supercomputing, Washington, D.C., 1994.

[17] G. J. W. van Dijk and M. J. van Gils, "Efficient process migration in the EMPS multiprocessor system," in *Parallel Processing Symposium, 1992. Proceedings., Sixth International*, 1992, pp. 58-66.

[18] D. Gelernter and D. Kaminsky, "Supercomputing out of recycled garbage: preliminary experience with Piranha," presented at the Proceedings of the 6th international conference on Supercomputing, Washington, D. C., USA, 1992.

[19] F. Douglis and J. Ousterhout, "Transparent process migration: Design alternatives and the sprite implementation," *Software: Practice and Experience*, 1991, 21(8), pp. 757-785.

[20] G. R. Andrews, "Paradigms for process interaction in distributed programs," *ACM Comput. Surv.*, 1991, 23(1), pp. 49-90.

[21] P. T. Wojciechowski, "Algorithms for location-independent communication between mobile agents," in Proceedings of AISB'01 Symposium on Software Mobility and Adaptive Behaviour (York, UK), 2001.

[22] A. Unyapoth and P. Sewell, "Nomadic pict: correct communication infrastructure for mobile computation," SIGPLAN Not., 2001, vol. 36, no. 3, pp. 116-127.

[23] S. Bohm, *et al.*, "Aggregation of Real-Time System Monitoring Data for Analyzing Large-Scale Parallel and Distributed Computing Environments," in *High Performance Computing and Communications (HPCC), 2010 12th IEEE International Conference on*, 2010, pp. 72-78.

[24] H. Chafi, *et al.*, "Language virtualization for heterogeneous parallel computing," *SIGPLAN Not.*, 2010,vol. 45,no. 10, pp. 835-847.

[25] D. Sánchez, *et al.*, "Agent-based platform to support the execution of parallel tasks," *Expert Systems with Applications*, 2011,vol. 38,no. 6, pp. 6644-6656.

[26] B. Fan, *et al.*, "Small cache, big effect: provable load balancing for randomly partitioned cluster services," presented at the *Proceedings of the 2nd ACM Symposium on Cloud Computing*, Cascais, Portugal, 2011.

[27] A. Dou, *et al.*, "Misco: a MapReduce framework for mobile systems," presented at the *Proceedings of the 3rd International Conference on PErvasive Technologies Related to Assistive Environments*, Samos, Greece, 2010.

[28] A. Benslimane, *et al.*, "Dynamic Clustering-Based Adaptive Mobile Gateway Management in Integrated VANET — 3G Heterogeneous Wireless Networks," *Selected Areas in Communications, IEEE Journal on*, 2011,vol. 29,no. 3, pp. 559-570.