A Model for Coordinating Jobs on Mobile Wireless Computational Grids

Adekunle Adeyelu
Maths and Computer Sc. Dept.
Benue State University
Makurdi, Nigeria

Emmanuel Olajubu
Computer Sc. and Eng. Dept.
Obafemi Awolowo University
Ile-Ife, Nigeria

Adesola Aderounmu
Computer Sc. and Eng. Dept.
Obafemi Awolowo University
Ile-Ife, Nigeria

Tivlumun Ge
Maths and Computer Sc. Dept.
Benue State University
Makurdi, Nigeria

ABSTRACT

This paper presents a report of a study carried out to develop a job coordination model for the active resources on a mobile wireless computational grid. This was with a view to addressing the problem of frequent disruptions in connections and high computation time for a job resulting from the mobility of the mobile resources actively processing tasks. The proposed framework is a load-balancing three tier hierarchical system configuration and scheduling policies employing mobile Agents coordinating messengers carrying data and instructions among the hierarchical structured nodes. The model achieves a remarkable performance as compared with theoretical values in that there were reduction in response times and latencies when simulated with various workloads. The proposed migration and checkpointing approach ensures that currently executing processes are not always migrated because of loss of signal only, but only with reduction in battery power of the mobile hosts within the allocated time for processing a task.

Keywords

Mobile wireless computational Grid, Mobile Agents, load-balancing, hierarchical, scheduling

1. INTRODUCTION

Wireless Grids extend the capability of Grid computing to wireless devices. It expands the wired grid thereby simplifying the interchange of information and the collaboration between heterogeneous wireless devices [1]. It is growing because of the fast developments in wireless technology and Grid computing technology. The number of users of laptops, PDAs (Personal Digital Assistants), cell phones, and other wireless devices is continuously increasing leading to more networked wireless devices, and creating an infinite united potential of unexploited resources [2]. Wireless grid computing supports sharing of these resources by mobile, and fixed wireless devices within the virtual organizations.

Computational grids represent a new and rapidly evolving research area, which has gained a lot of attention in the past several years. A computational grid can be viewed as a transparent accumulation of many computing devices on a network that facilitates sharing of distributed resources [3]. They are typically considered in the context of sharing processor power of many computers interconnected by a wired network. Most of the current grid applications focus on high performance computing mostly supporting applications of scientific research. Consequently, computing devices employed for such grid implementations usually consist of homogeneous collections of computers which are rich in different resources up to the class of servers [4].

Mobile wireless grids are challenged by dynamic topology due to the mobile nature of the limited computational and battery powered active resources; resulting in frequent disruptions in connections and high computation time for jobs. Since the purpose of the Grid is to provide a highly available, consistent and ubiquitous environment, integrating mobile devices into the Grid, through wireless links, could undermine all relevant quality of service requirements. Architecture to support large numbers of mobile devices in a computational grid for example must address the issues of device heterogeneity, low-bandwidth, high-latency connectivity; possibly extended periods of disconnectionon; device power consumption; and software interoperability. This work attempts to solve the problem of instability that results from disruptions in accessing the active wireless resources in a mobile wireless computational grid computing environment by developing a coordination framework with self- configuring and self-administering capability that allows dynamic changes for a mobile computational grid architecture and deliver results even in events of failures and instabilities of some of the scheduled mobile nodes.

2. RELATED WORKS

This section presents relevant works done on job coordination on wireless networks and mobile computational grids. In one of the works, the multi-agent system approach was employed using a cellular network-based grid architecture in which all agents have a high degree of independence [5]. The computing resources of the existing mobile devices in a given cell were aggregated to solve a resource-intensive task thereby realizing ubiquitous computing. In this work cellular networks are divided into geographical areas of service called cells. Each cell has a base station that uses wireless transmission technologies to provide services to mobile users in its area. The mobile computational resources providing services to these mobile users must reside within the non-interacting cell networks. The system is in such a way that only mobile nodes can initiate a task. The completion of assigned task cannot be guaranteed as it could be disrupted if a subordinate agent suddenly becomes an initiator. This architecture is not tolerant to temporal mobility of the subordinate out of the wireless range. Also migration of sub-tasks to another agent without checkpointing makes it unsuitable for cooperating dependent computation. This wastage of computational resources is imminent as the Initiator can anytime abort the task, leading to termination of tasks on the subordinate.

In another research effort, there was an improvement in this architecture through the modification of the function of the Keep-alive server which collects real-time information about the progress of each Subordinate towards completing the partial computational task delegated to it by the grid infrastructure [6]. If one of the Subordinates should time-out or abort a partial computational task, the Keep-Alive server will inform the Brokering Service, which will reallocate the corresponding partial task to a different Subordinate. On the other hand, if the Initiator of a task moves to a new area, the Brokering Service is currently programmed to abort all partial tasks associated with the Initiator.

It also attempted to hide the complexity of the underlying architecture from clients of the resource-sharing services. As the Subordinates or the Initiator moves out of range of the wireless cell containing the original Brokering Service, the handoff mechanisms in place ensure a smooth, transparent transition to the next cell, thus making the user unaware of the change of control.

The research attempt proposed a collaborative problem-solving Framework for Mobile devices. It addressed only the problem of handoff resulting from the movement of either the Initiator or the subordinate migrating to another cell by proposing load balancing techniques employed in wired networks to resolve such imbalances and restore network stability [6].

Further improvement led to another architecture that embraces mobile devices within a wireless cell to share their computational power by joining a framework modeled after a computational grid [7]. The paradigm of a multi-agent system was used with each mobile device viewed as an autonomous intelligent agent [8], [9] with a significant degree of autonomy, capable of performing independent tasks, sharing resources with other agents, and communicating with other agents in the grid.

These attempts were operational within a cell network. None of them addressed issues of mobility of these mobile nodes out of their cells, even if their battery power is still high enough to continue with computation, a condition that is highly probable in a mobile wireless grid. There was no consideration for interactions among cells. The research effort seeks to tackle these issues with a view to developing a recovery protocol that deals with the peculiar nature of mobile wireless computational grid, such that the system could coordinate the scheduled resources and the computational tasks to completion even in the event of failures and instability of mobile resources across heterogeneous platforms and cells.

3. MATERIALS AND METHODS

The architecture of the model is as shown in Figure 1. It is a three-tier paradigm consisting of the Chief coordinator at the base level, level coordinator at the middle and mobile hosts at the last level. The scheduling policy employed is hierarchical. The chief coordinator breaks down the submitted job into tasks and distribute to the level coordinators. The level coordinators split the tasks into processes for the mobile hosts to execute. The mobile agents move the instructions, data and results from one level of the architecture to the other. They also monitor the power and signal strength status of the mobile hosts. Migration and checkpointing operations are also carried out by these agents.

The wireless mobile computational grid system generally consists of **n** mobile worker nodes called Mobile Hosts (MHs) and **m** Level coordinators called Mobile Support Nodes (MSNs), n>>m. MHs are connected through wireless networks and MSNs are connected through wired network. Communication links connecting MHs and MSNs are assumed to be First In-First-Out (FIFO). Jobs take arbitrary but finite amount of time for processing. There are no synchronized clocks or shared memory among mobile hosts and level coordinators except at the chief coordinator level. Two types of messages are assumed; execution messages generated based on computational work processes and Coordination Messages generated to coordinate the checkpointing and migration activities. Two types of checkpoints are saved; Migration Checkpoint which is saved before planned disconnection of MHs and permanent checkpoints which is saved by the level and chief coordinators. One or more tasks on the level coordinators may try to initiate checkpointing but only one task can have the privilege over the others depending on if the task is a cooperating dependent one. The algorithm is shown in Figure 2.

The mobile agents adapt the MESSENGER coordination model. They send the migration checkpoints to the level coordinators before planned disconnection and migration. They also use Mobile Internet Protocol (Mobile IP) to relay the migration checkpoints or results of completely executed processes to the level coordinators within the time assigned for them. Only the most recent checkpoints are saved by the mobile agents on the local storage of the mobile hosts. If any mobile host fails as a result of hardware or software failure, the level coordinator reschedules the program to another available mobile host.

The mobile agent also monitors the battery power and signal strength status of the worker nodes. For this architecture, battery power status of the mobile host is taken as a higher priority condition for migration than signal strength. This is because battery power availability is a critical factor for executing a program. The mobile host can continue to run a program even if it is out of network range provided that its battery power is not lower than a preset minimum value. The agent performs migration at a

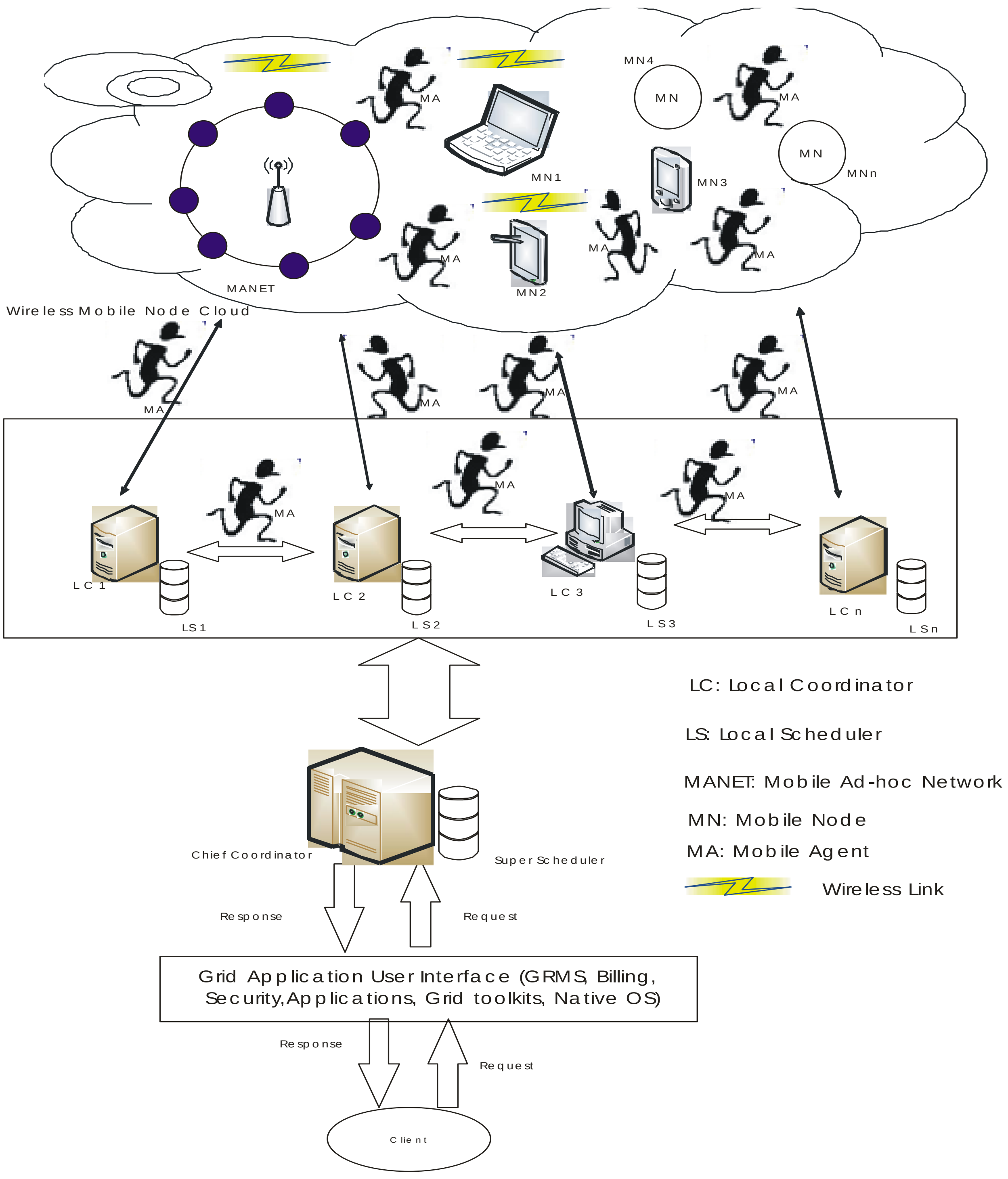

Figure 1: Job Coordination Model Architecture

```
input: λ:job in form of computing cycles
output: result: result of  completed job
12.1    The function receives job and data from a grid user submits result after completion of computation.
12.2    User submits job in form of computing cycles λ
12.3    Chief coordinator receives job, create tasks and submits to scheduler create( )
12.4    scheduler assigns tasks(1,................., n) to coordinators 1,.................,n
12.5    each level coordinator $Al_i$ receives tasks $P_i\lambda$, i=1,......, n; $\sum_{i=1}^{n} P_i = 1$
12.6    local scheduler on level coordinator breaks tasks $P_i\lambda$, i=1,......, n; $\sum_{i=1}^{n} P_i = 1,$    into processes
P(i,1)$P_i\lambda$, ........................., P(i,j)$P_i\lambda$; j≤m, m= maximum number of  mobile hosts scheduled per level coordinator and
assigns them to mobile hosts mhost(i,j) through the mobile agent.
12.7    mobile hosts receive programs and data for execution receive( )
12.8    mhost(i,j) starts executing the program
        12.8.1     while mhost(i,j) not timed out
        12.8.2              {
        12.8.3              monitor ← SignalStrength ( )
        12.8.4              update ← UpdateIp
        12.8.5              }end do
        12.8.6              if power ← BatteryPower ( ) ≤ preset minimum
        12.8.7              {        Update Checkpoint chkpt( )
        12.8.8                       migrate and update registry
        12,8.9              } else
        12.8.10   update chkpt ( )
        12.8.11   if (mhost(i,j)  timed out and SignalStrength (i,j) =0 then  { terminate mhost(i,j)
        12.8.12   reschedule         process on mhost(i,j) to another mobile host
        12.8.13   update registry of  level coordinator
        12.8.14                      }
        12.9      for level = 1 to number of mobile hosts {
        12.9.1    level collector receives partial results result(i,j) from mobile hosts mhost(i,j)}
        12.9.2    level collector receives results of executed task(k), k≤n from other Al(k) for cooperating dependent
                  computation
        12.9.3    level scheduler takes checkpoints
        12.9.4    if (level coordinator(k) timed out) then {
        12.9.5    reschedule $P_k\lambda$ to another level coordinator Al(w) w > n
        12.9.6    Chief coordinator updates its registry
12.10   level coordinator sends results of completed task task(i) to chief coordinator
12.11   Chief coordinator accepts results from other level coordinators Al(1,........,n)
12.12   Chief coordinator performs final computation
12.13   chief coordinator submits result to grid job owner
12.14   end
```

Figure 2: Overall algorithm of Job coordination

threshold level in the event of constant reduction of power. It takes checkpoints at increasing intervals and sends the last updated checkpoint to the level coordinator when the available battery powers is at or lower than the permissible minimum. In this work, a worker node could move from one network to another while it is still running its scheduled programs. It is also assumed that the mobile host is lost only if couldn't return results of a scheduled program within the time frame allocated and couldn't communicate with the host; the level coordinator.

When this happens, the level coordinator reschedules the remaining part of same program to available mobile host to continue from the last checkpoints received from the lost host. On the other hand, if the mobile worker node is on another network, its agent sends its checkpoint, and address of the new mobile hosts to the host coordinator using Mobile IP. This happens only when it has finished execution of the program or the allocated time for the program has elapsed. It does not send checkpoints to the level coordinator after every hop from one wireless node to the other. The mobile agents employ uncoordinated checkpointing by recording individual checkpoint independently. The mobile agent has the full autonomy in deciding when to take checkpoint. This eliminates coordination overhead all together and forms a consistent global state on recovery

The level coordinators employ the bus-based shared memory model to coordinate the tasks employing Objective Linda data structure. They are used as the intermediate servers that handle processes distributed on the virtual network set up to solve a specific problem on the grid. They connect with the chief coordinator.

A level coordinator keeps records of checkpoints of all mobile hosts on its cell and has the capability to perform migration operations within its cell network. It performs increasing interval backup and update of checkpoints from all active hosts and executes the task after receiving results of completed programs from all the mobile hosts as well as from the other level coordinators if necessary. It also keeps records of:

i. Scheduled processes stored as a database.
ii. active mobile hosts assigned to execute each process stored as an array of numbers.
iii. Variables (input, output and intermediate variables) of processes and task allocated stored as an array of characters,
iv. Expected completion time of each scheduled process as array of numbers and characters.
v. Available mobile hosts as an array of numbers
vi. Current checkpoints of each mobile host as a database
vii. routing information of the network as a linked list

It consists of the registry, local scheduler, level checkpointer, level collector, level shared memory and migration unit.

a. The Registry

This is made up of:

i. List of all the mobile hosts and their properties stored as a database; and
ii. database of available mobile hosts in events of the need for migration with their current status (processor capacity available, IP address, physical memory available, current battery power, signal strength)

b. Local Scheduler

This is the module that outlines plan of work to be done, showing the order in which processes are to be carried out and the amounts of time allocated to each of them. It schedules processes to available mobile hosts by obtaining records of available resources from the registry and allocate processes to them depending on the ones that meet the requirements for the process.

c. Level Checkpointer

This segment keeps records of the mobile nodes' checkpoints within its cell. When a checkpoint file arrives from a mobile node through its agent, it is downloaded and saved on the level coordinator storage. Checkpoints can potentially be very large, so a check is made for the entire checkpoint file and only the most current checkpoint is kept.

d. Level Collector

This module gathers and keeps the results returned by the agents from different mobile hosts, performs the final computations (if necessary), then sends results to the chief coordinator through level schedulers. It also monitors the status of the processes and resources in each node, checks for the conditions for migration before initiating migration procedures.

e. Level Shared–memory

The shared memory is a common, content-addressable data structure. It contains tuples which are sequences of typed fields consisting of basic data items like numbers and character strings. Level coordinators communicate with the chief coordinator by writing or reading tuples into or out of the shared memory. The basic communication acts are: creating a tuple, and removing or reading tuple from the shared memory. Synchronization is performed by letting level coordinators tasks wait until a tuple to be read has been inserted into the shared memory. Tuples are sent into the shared memory by means of templates and retrieved by providing templates which matches certain tuples (associative pattern matching). The template itself has a tuple structure and hence determines arity, types of the elements, and optionally constant values for the elements of a matching tuple. These elements are specified by type and value (actuals) and/or typed placeholders (without a value, called formals).

For a tuple to be read, the requesting worker node specifies a template for a tuple it wishes to obtain. The shared memory performs a matching operation in order to find an appropriate tuple. A tuple matches a given template if the arities of both correspond and if each actual field matches one of the same type and value or a formal field of the corresponding type.

The shared memory keeps dependent tuples for other level coordinators to access. Each level coordinator can observe the state of the shared memory and check for tuple requests of the shared memory. When one node makes reference to a tuple that is contained in another node's memory, the other nodes can see the request and respond with the required data. The tuple is only placed in the shared memory only if it will be used by more than one node or if it is ready before any request is made of it. The **in** statement removes the matching tuples from the shared memory. This is applied when the tuple is no longer needed by any worker node. The **rd** statement makes a copy of the tuple, having the original in the shared memory.

The chief coordinator dynamically constructs an application-specific logical network over a network of available grid resources. Multiple logical coordinators can be mapped on to the same chief coordinator. Each link of the logical network has a grid determined name and several (optional) weights. These integer variables are what the chief coordinator uses for navigation. Each allocated node has a grid- assigned name and a system-wide address. It also contains node variable area, which can be accessed by other coordinators for the purpose of coordinating among themselves. The chief coordinator supports three types of variables referred to as checkpoint values of the level coordinators, node-values on the grid architecture (node id), and network-variables (node name, node address, link-names, link–weights). It employs the registry, scheduler, checkpointer, collector and shared memory, all of these performing similar functions as in the level coordinator. It also performs migration operations in event of failures of scheduled level coordinators. Figure 2 shows the overall algorithm of the job coordination model.

The simulation program was written in MATLAB 7.10.0. The workload was simulated as a number of computing cycles required to complete a job using random numbers. Ten workloads were used to test the simulation program. Different instructions were also incorporated into the program to measure the response time and latency performance metrics for the workload. The processing powers, in form of computing cycles were set between1MHZ and 3GHZ for mobile nodes and between 3GHZ and 100GHZ for the level coordinators using random numbers. The simulation programs were run for different number of level coordinators **n** = (10, 25, 50, 75, 100), and mobile nodes **m**= (1000, 2500, 5000, 7500, 10000). The workload for each run was increased gradually from 10^{20} to 10^{21} computing cycles with a step of 2. Random numbers using the linear multiplicative congruential pseudo random number generator were generated to simulate all the system parameters and workload parameters. The simulation experiments were carried out eight times (arbitrary value) for the same set of jobs. The geometric means of these results were taken because a single extreme value has less of an impact on the geometric mean of a series than on the arithmetic mean [10]. Using the geometric mean makes it harder for a system to achieve a high score on the benchmark suite by achieving good performance on just one of the programs in the suite, making the system's overall score a better indicator of its performance on most programs. The geometric mean of **n** values is calculated by multiplying the **n** values together and taking the **n***th* root of the product. The results were plotted as graphs. The performances of the model were tested by varying a set of conditions while keeping the others constant. This was done in order to analyze the sensitivity of the model [11] with respect to response time and latency performances as the number of mobile nodes per level coordinator and number of level coordinators vary for different workloads.

4. DISCUSSION OF RESULTS

The performance metrics of response time and latency were used to specify the performance of the model formulated for coordinating jobs on the wireless computational grid. The metrics were measured relative to the first and lowest workload; 10^{20} cycles, **n**=10 and **m**=1000 as the cases may be. **n** stands for number of level coordinators and **m** implies number of mobile nodes per level coordinator. Workload and job are used interchangeably. The results are as presented in the following section.

4.1 Response Time versus Latency Performances at Fixed Points

This section examined the response time and latency performances of the system when operated at arbitrarily chosen levels for all the workloads. The graphs of response time and latency for fixed values of **n** and **m** were plotted together for easy assessment. Figure 3 shows variation of response time and latency for the different jobs taken at arbitrary points **n**=10, **m**=5000. From the graph there is a reduction by 18.75% in response time for workload $2x10^{20}$ cycles. Workload $3x10^{20}$ cycles recorded 59.38% reduction in processing time over the first. The third workload's response time obviously the highest is about 59.38% lower than the expected value relative to the first job. The latency performance showed that latency reduced by 12.5% when workload increased from 10^{20} to $2x10^{20}$ cycles. A further reduction by 87.5% was observed when the workload was increased to $3x10^{20}$ cycles. The system performed optimally on workload 10^{21} cycles with reduction in latency by 97.22%. By observation, latency reduced as the workload increased, though at some points, for example workload $7x10^{20}$ cycles for possible reasons of checkpointing and migration due to failure of mobile nodes, it relatively increased compared to the heaviest workload.

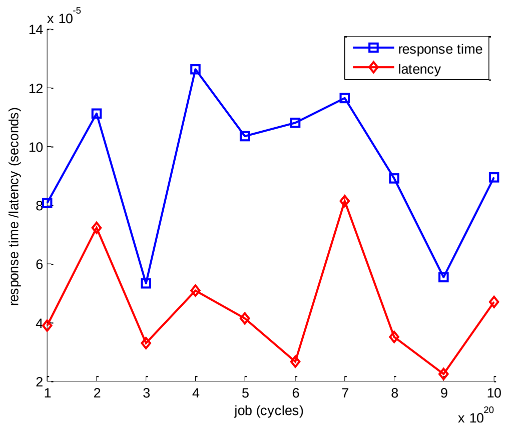

Figure 3: Graph of response time and latency against workloads for n=10, m=5000

Figure 4 shows the graphs of response time and latency taken at **n**=10 for workload 10^{20} cycles varying **m**. The purpose is to investigate the variation of response time and latency with various number of mobile nodes per level coordinator **m** and fixed number of level coordinator **n** for an arbitrarily chosen workload. From the results, the response time increased from 1000 to 2500, by 200% at **m**=5000, by 400% at **m**=7500 and by 500% at **m**=10000. This showed that the response time increased as the number of mobile nodes per level coordinator increases for fixed number of level coordinator while processing a job. This implies that the system responded faster when operated at a lower number of mobile nodes per coordinator.

From the same figure, the results showed that latency increased by 150% from **m**=1000 to 2500, by 300% at **m**=5000, by 400% at **m**=7500 and 420% at **m**=10000. There was a significant drop in the percentage of increase of latency form m=5000. For example the increase in latency from m= 5000 to m= 7500 was 100%, while it was 20% between m=7500 and 10000. The results showed that the latency increased as the number of mobile nodes per level coordinator increases for fixed number of level coordinator while processing a job. This implies that the system performed best in latency when operated with fewer mobile nodes per coordinator.

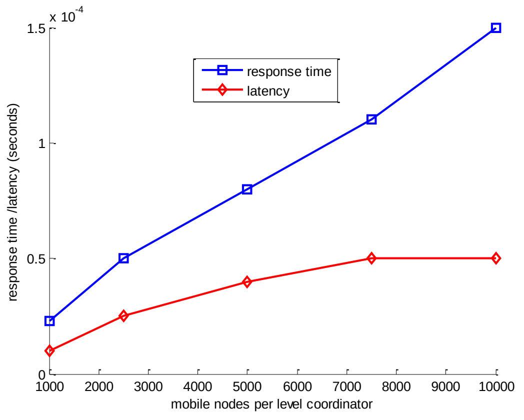

Figure 4: Graph of response and latency for 10^{20} cycles workload at n=10

Figure 5 shows the graph of response times for all the workloads with **m**=1000. From the results, for workload 10^{20}cycles, response time increased with increase in number of **n**; for example there was an increase in response time by 108.70%, 160.87%, 421.74% and 856.52% respectively as **n** increased from 10 to 25, 50, 75 and 100. Similarly, for workload $8x10^{20}$cycles, there was an increase in response time by 571.53%, 542.86%, 1471.43% and 937.71% respectively when **n** was increased from 10 to 25, 50, 75 and 100. For **n**= 10, the response time for workloads $3x10^{20}$, $4x10^{20}$ and $7x10^{20}$ cycles reduced by 120%, 441.18% and 762.50% respectively. Similar trends were observed for n=25, 50,75 and 100, where for instance response time reduced by 71.43%, 620%, and 620% correspondingly for workloads $3x10^{20}$, $6x10^{20}$ and 9 $x10^{20}$ with **n**=75. The results suggested that response time increases as the number of level coordinators on similar job increases and decreases as the workload increases irrespective of the number of level coordinators.

Figure 6 shows the graph of latency of workloads operated at **n**=10, 25, 5, 75 and 100 for **m**=2500. Table 5.6 and figure 5.9 show similar results for **m**= 5000. The values **m**= 2500 and **m**= 5000 were chosen arbitrarily in order to investigate the impact of changing **n** on each workload with fixed **m**. The results showed that the latency at **n**=10, **m**=2500 for example, latency reduced by 16.67% from workload 10^{20} to $2x10^{20}$ cycles, increased by 8.33% on workload $5x10^{20}$ cycles, and reduced by 16.67% at 10^{21}cycles workload. Also at **n**=50, latency reduced by 62.5%, 50%, 60% and 67.5% respectively on workloads $4x10^{20}$, $5x10^{20}$, $6x10^{20}$ and 10^{21}cycles and reduced by 40% only on workload $2x10^{20}$cycles. For same workload 10^{20} cycles, latency increased by 3325%, 8166.67%, 629.17% and 3566.67% when **n** increased from 10 to 25, 50, 75 and 100 respectively. For workload $9x10^{20}$ cycles, it increased by 2150%, 100%, 650% and 550% respectively as **n** increased from 10 to 25, 50, 75 and 100 accordingly. These results suggested that latency reduces as the workload increases for fixed number of level coordinator and increases as the number of level coordinator increases for a fixed workload at **m**=2500.

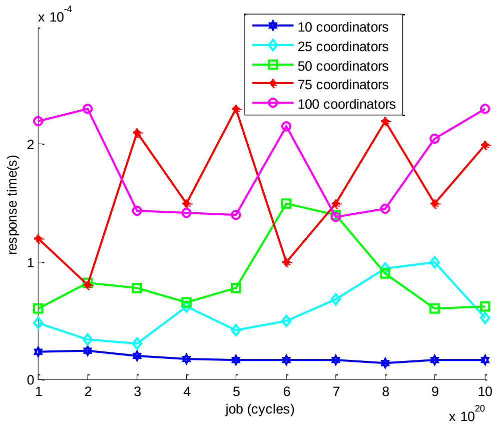

Figure 5: Graph of response time for jobs at m=1000

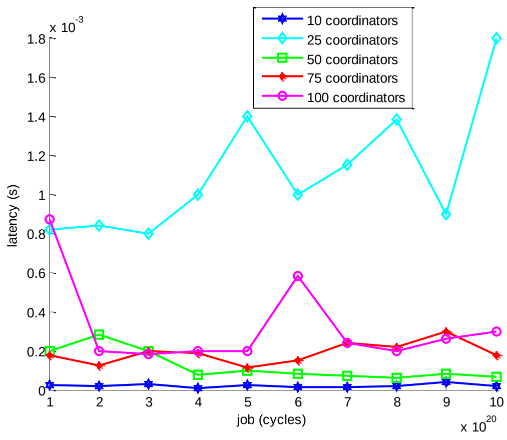

Figure 6: Graph of latency for workloads at m=2500

Generally it was observed that response time increases with increasing number of level coordinators on same job and decreases with increasing number of workloads irrespective of the number of level coordinators and number of mobile nodes per level coordinator. Also latency reduces as the workload increases for a fixed number of level coordinator and increases as the number of level coordinator increases irrespective of the workload and number of mobile nodes per level coordinator.

5. CONCLUSION

In this study, a hierarchical load balancing paradigm facilitating the reliability of mobile wireless computational grid was developed. The proposed architecture suggests solutions to resource and network failure by application of modified checkpointing and migration protocol. On receiving a job, the chief coordinator splits in into tasks for level coordinators which also split them into processes to be executed by mobile hosts. The mobile agents were the carriers of data and instructions over these hierarchically structured nodes. From the results of the simulation it could be shown that the proposed load balancing approach enhances the reliability and efficiency of the mobile wireless computational grid system in that the response and delay times are predictably lower compared with the expected theoretical values.

6. REFERENCES

[1] Agarwal, A., Norman, D. O. and Gupta, A. 2004 Wireless Grids: Approaches, Architectures, and Technical Challenges. A working paper 4459-04, MIT Sloan School of Management.

[2] Manvi, S. and Birje, M. 2010. A Review on Wireless Grid Computing. International Journal of Computer and Electrical Engineering, **2**(3): 469-473.

[3] Kurkovsky, S., Bhagyavati and Arris, R. 2004 Emerging Issues in Wireless Computational Grids for Mobile devices. In Proceedings of the 8th Multiconference on Systemic, Cybernetics, and Informatics, Seattle, WA

[4] McKnight, L.W. and Howison, J. 2003. Towards a Sharing Protocol for Wireless Grids.Proceedings of International Conference on Computer Communication and Control Technologies (CCCT '03), Orlando FL.

[5] Kurkovsky, S. and Bhagyavati 2003a. Modelling a Computational Grid of Mobile Devices as a Multi-Agent

System. In proceedings of the 2003 International Conference on Artificial Intelligence (IC-AI'03), Las Vegas, NV.

[6] Kurkovsky, S., Bhagyavati and Arris, R. 2004a. Modelling a Grid-Based Problem-Solving Environment for Mobile Devices. Journal of Digital Information Management, **2** (2): 109-114.

[7] Kurkovsky, S., Bhagyavati and Arris, R. 2004b. A Collaborative Problem Solving Framework for Mobile Device. In Proceedings of ACMSE '04, Huntsville, Alabama USA.

[8] Fukuda M., Tanaka Y., Suzuki N., Bic L.F. and Kobayashi S. 2003. A Mobile-Agent based PC Grid. Proceedings of the 5th Annual International Workshop on Active middleware services (AMS2003), Seattle, WA.

[9] Kuang, H., Bic, L. and Dillencourt, M. 2002. Iterative Grid-Based Computing Using Mobile Agents. In Proceedings of the 2002 International Conference on Parallel Processing, Vancouver, B.C. Canada.

[10] Carter, N. 2002. Computer Architecture. Schaums Outline Series, Tata McGraw- Hill, New Delhi :279-287.

[11] Banks, J. 1998. Handbook of Simulation: Principles, Methodology, Advances, Applications and Practice, Engineering and Management Press, USA: 23-25.