Collaborative Autonomic Resource Management System for Mobile Cloud Computing

Ahmed Khalifa[1,2], Mohamed Eltoweissy[1]
[1] The Bradley Department of Electrical and Computer Engineering, Virginia Tech, Blacksburg, Virginia, USA
[2] Switching Department, National Telecommunication Institute, Cairo, Egypt
e-mail: {akhalifa, toweissy}@vt.edu

Abstract— **Mobile cloud computing promises more effective and efficient utilization of the ever-increasing pool of computing resources available on modern mobile devices. To support mobile cloud computing, we propose a Collaborative Autonomic Resource Management System (CARMS), which automatically manages task scheduling and resource allocation to realize efficient cloud formation and computing in a dynamic mobile environment. CARMS utilizes our previously proposed Global Resource Positioning System (GRPS) to track current and future availability of mobile resources. In this paper, we present CARMS architecture and its associated Adaptive List-based Scheduling and Allocation AlgorithM (ALSALAM) for adaptive task scheduling and resource allocation. ALSALAM uses the continually updated data from the loosely federated GRPS to automatically select appropriate mobile nodes to participate informing clouds, and to adjust both task scheduling and resource allocation according to the changing conditions due to the dynamicity of resources and tasks in an existing cloud. Our simulation results show that CARMS offers effective and efficient support for mobile cloud computing that has not yet been adequately provided by prior research.**

Keywords- Mobile cloud computing; Resource management; Dynamic resource maps;Autonomic computing; Collaborative computing.

I. INTRODUCTION

Cloud computing enables the delivery of computing resources as a utility. This utility concept is expected to drastically bring down computing costs. Moreover, the computation resources of mobile devises are increasing, for example quad-core platforms and significantly enhanced storage and memory capabilities. Recently, principles of cloud computing have been extended to the mobile computing domain, leading to the emergence of Mobile Cloud Computing (MCC). A MCC system (MCCS) has been defined from different views in the literature [1]. One of these perspectives defines a MCCS as a way of outsourcing the computing power and storage from mobile devices into an infrastructure cloud of fixed supercomputers. Here, a mobile device is simply a terminal which accesses services offered in the cloud. Another view defines a MCCS as an infrastructure-less cloud that is formed locally by a group of mobile devices, sharing their computing resources to run applications. This paper adopts and extends the latter definition as follows: A MCCS is a shared pool of configurable computing resources that are harvested from available or potentially available local or remote nodes that are either mobile or fixed over a network to provide on-demand computational services to users.

Mobile devices in MCCS are expected to have reasonably powerful capabilities, for example exploiting the virtually unlimited power supply in our vehicles making them good candidates for housing powerful on-board computers augmented with huge storage devices that may act as networked computing centers on wheels [15].

MCC has a dynamic nature as nodes, usually having heterogeneous capabilities, may join or leave the formed cloud varying its computing capabilities. Also, the number of reachable nodes may vary according to the mobility pattern of these nodes. Resource management systems for MCCS should support this dynamicity, hide the heterogeneity of resources, provide users with unified access, evaluate and predict the availability and performance of resources, and guarantee the quality of service to meet users' requests.

Research in resource management systems and algorithms for mobile cloud computing is still in its infancy. In [4], authors proposed a preliminary design for a framework to exploit resources of a collection of nearby mobile devices as a virtual ad hoc cloud computing provider. In [5], a mobile cloud computing framework was presented. Experiments for job sharing were conducted over an ad-hoc network linking a user group of mobile devices. The Hyrax platform [6] introduced the concept of using mobile devices as resource providers. The platform used a central server to coordinate data and jobs on connected mobile devices. Task scheduling and resource allocation algorithms were reported in [7-11]. These algorithms used cost, time, reliability and energy as criteria for selection.

Most of the existing resource management systems [4-6] for MCC were designed to select the available mobile resources in the same area or those follow the same movement pattern to overcome the instability of the mobile cloud environment. However, they did not consider more general scenarios of users' mobility where mobile resources should be automatically and dynamically discovered, scheduled, allocated in a distributed manner largely transparent to the users. Additionally, most current task scheduling and resource allocation algorithms [7-11] did not consider the prediction of resource availability or the connectivity among mobile nodes in the future, or the channel contention, which affects the performance of submitted applications. Consequently, there is a need for a solution that effectively and autonomically manages the high resource variations in a dynamic cloud environment. It should include autonomic components for resource

 ISBN: 978-1-61208-271-4

discovery, scheduling, allocation and monitoring to provide ubiquitously available resources to cloud users.

In an apparent departure from previous work, our Collaborative Autonomic Resource Management System (CARMS) provides a more general distributed solution to cloud formation and management based on dynamic calendars of available or potentially available resources. Our main contributions in this paper are:

(1) The CARMS architecture which provides system-managed cloud services such as configuration, adaptation and resilience through collaborative autonomic management of dynamic cloud resources, services and membership; and

(2) Adaptive task scheduling and resource allocation algorithm to map applications' requirements to the currently or potentially available mobile resources. This would support formed cloud stability in a dynamic resource environment.

The rest of the paper is organized as follows. Section II presents the architecture of CARMS. Section III presents ALSALAM; our proposed adaptive task scheduling and resource allocation algorithm. Section IV discusses the performance evaluation. Finally Section V concludes the paper and outlines future work.

II. COLLABORATIVE AUTONOMIC RESOURCE MANAGEMENT SYSTEM (CARMS)

In [12], we proposed the PlanetCloud concept to enable MCC to tap into the otherwise unreachable resources, which may be located on any opt-in reachable node, rather than being exclusively located on a static cloud service providers' side. A key PlanetCloud component was the Global Resource Positioning System (GRPS) that we presented in detail in [13]. GRPS adopts a spatiotemporal calendaring mechanism with real-time synchronization to support dynamic real-time recording and tracking of idle mobile or fixed resources. The calendar consists of records including data about time, location, and computing capabilities of GRPS participants. GRPS also forecasts the availability of resources, anytime and anywhere. GRPS makes use of the analysis of calendaring data coupled with data from other sources such as social networking to improve the prediction accuracy of resource availability. In addition, the GRPS provides hierarchical zone architecture with a synchronization protocol between different levels of zones to enable scalable resource-infinite computing.

In this paper, we describe and evaluate our CARMS integral to PlanetCloud. In PlanetCloud, a cloud application comprises a number of tasks. At the basic level, each task consists of a sequence of instructions that must be executed on the same node. Tasks of a submitted application are represented by nodes on a directed acyclic Graph (DAG) which is addressed in the next section. The set of communication edges among these nodes show the dependencies among the tasks. The edge $e_{i,j}$ joins nodes v_i and v_j, where v_i is called the immediate predecessor of v_j, and v_j is called the immediate successor of v_i. A task without any immediate predecessor is called an entry task, and a task without any immediate successors is called an exit task. Only after all immediate predecessors of a task finish, that task can start its execution.

CARMS manages clouds of mobile or hybrid resources (resources of mobile and fixed nodes). A CARMS-managed cloud consists of resources on virtual nodes that meet the cloud applications' requirements. Each virtual node is emulated by a subset of the real physical mobile nodes. The subset locally stores the state of the emulated virtual node. The real nodes perform the tasks assigned to their emulated virtual node. If a mobile node fails or leaves the cloud, it ceases to emulate the virtual node; a mobile node that joins the cloud attempts to participate in the emulation. CARMS attempts to provide each subset with a sufficient number of real mobile nodes, such that in case of failure, a redundant node can be ready to substitute the failed node.

A Cloud Agent, as a requester to form a cloud, manages the formed cloud by keeping track of all the resources joining its cloud using the updates received from the GRPS.

We design our CARMS architecture using the key features, concepts and principles of autonomic computing systems. As shown in Fig. 1, components of the CARMS and GRPS architectures interact with each other to automatically manage resource allocation and task scheduling to affect cloud computing in a dynamic mobile environment.

CARMS interacts with the information-base which maintains the necessary information about a requested cloud. The information-base includes user information, e.g., personal information and subscribed services, etc. Also, it contains information about the formed cloud, e.g., SLAs, types of resources needed, the amount of each resource type needed, and billing plan for the service.

CARMS comprises two primary types of nodes: Cloud Agent and participant nodes. CARMS performs all required management functions using the components detailed below.

1) Controller: In order to obtain a self-controlled operation, a controller is needed to automatically take appropriate actions. These actions are taken according to results of the evaluation received from the Performance Analyzer, described below, due to variations in the performance and workload in a cloud environment. The Controller manages interactions to form, maintain and disassemble a cloud. Besides, it makes decisions according to the applied policies. The Controller provides both policy and participant control functions. The policy control function prevents conflicts and inconsistency when policies are updated due to change inthe demands of a cloud. In addition, it distributes policies to other CARMS components. On the other hand, the participant control function manages the interaction between a cloud requester and resource providers, the cloud participants, to perform a Service Level Agreement (SLA) negotiation. Once the negotiation is successful, the participant control function updates the billing information and SLA of a participant in the information-base. Then, the Controller sends a cloud activation request to a Cloud Manager component.

 ISBN: 978-1-61208-271-4

2) Cloud Manager: decomposes the requested application, in a Cloud Agent, upon receiving a cloud activation request, to a set of tasks. This component can create some policies on the fly and assign a set of virtual resources to these tasks according to the received SLA information from the controller component. Then, the information of thevirtual resources is sent to the resource manager component for the appropriate real mobile resources allocation or de-allocation.

3) Resource Manager: Real mobile resources need to be allocated to the requested application. On the other hand, tasks of a requested application need to be scheduled. The Resource Manager component handles the resource allocation and task schedulingprocesseson real mobile nodes.The Resource Manager consists of two main units:

a) Resource Allocator: allocates local real resources for a task. Also, the resource allocator obtains the required information about the available real resources from(potential) participants by interacting with a GRCS of GRPS system. The Resource Allocator interacts with theregistry of Cloud Agentto store and retrieve the periodically updated data related to all participants within a cloud.

b) Task Scheduler: distributes tasks to the appropriate real mobile nodes and keeps a copy of these tasks in an image registry to retrieve them as needed such as in case of failure.

4) Monitoring Manager: consists of the following two units:

a) Performance Monitor: monitors the performance measured by monitoring agents at resource providers. Then, it provides the results of these measurements to the Performance Analyzer component.

b) Workload Monitor: The workload information of the incoming request is periodically collected by the Workload Monitor component.

*5) Performance Analyzer:*continually analyzes the measurements received from the Monitoring Manager to detect the status of tasks and operations, and evaluate both the performance and SLA. The results are then sent to both the Controller and the Account Manager.

6) Account Manager: In case of violation of SLA, adjustments are needed to the bill of a particular participant. These adjustments are performed by the Account Manager component depending on the billing policies negotiated by the requester of cloud formation.

III. ADAPTIVE TASK SCHEDULING AND RESOURCE ALLOCATION ALGORITHM

A. Application Model

For simplicity, we start with a basic application model. The load of submitted application is defined by the following

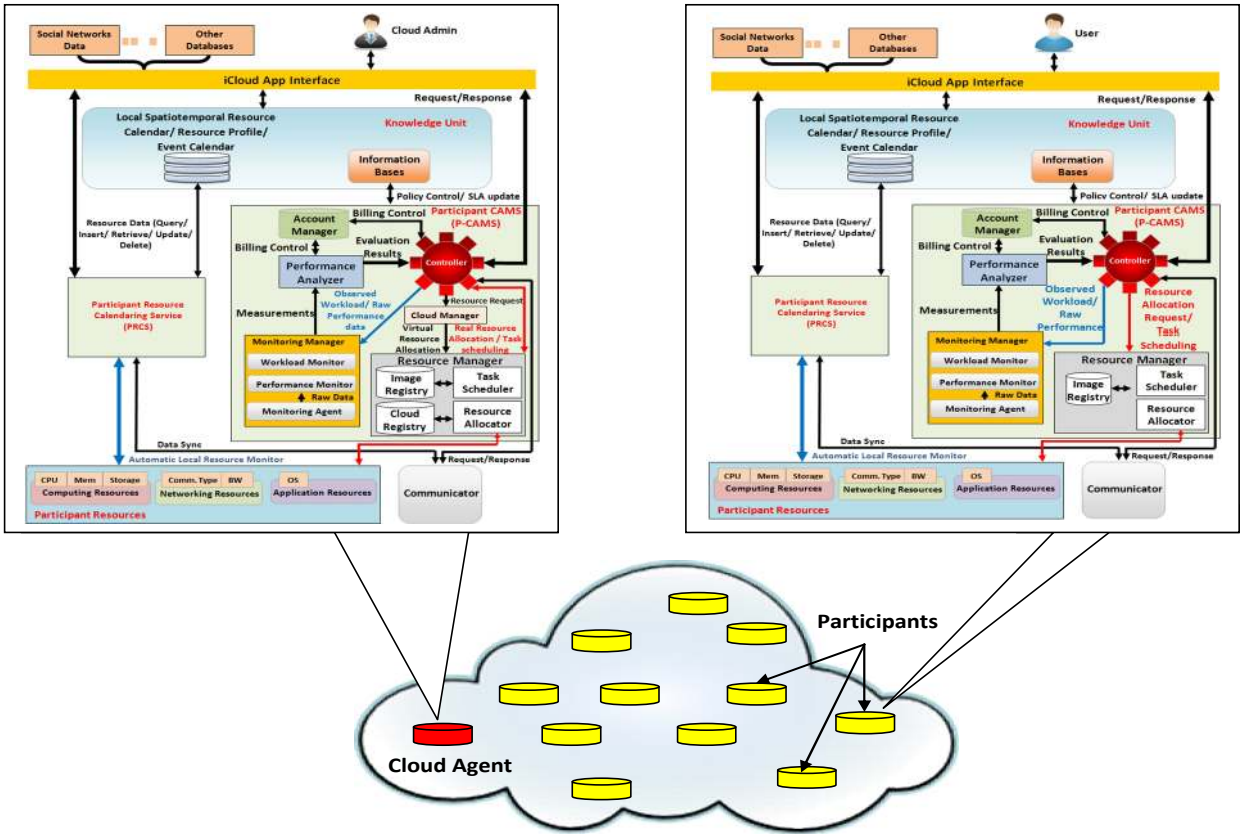

Figure 1. CARMS Architecture.

parameters: the number of submitted applications, the number of tasks per application, and the settings of each task. For example, the input and the output file size of a task before and after execution in bytes, the memory and the number of cores required to execute this task, and the execution time of a task.

Based on the criteria for selection, we mainly define two matrices: Criteria costs matrix, C, of size $v \times p$, i.e., $c_{i,j}$ gives the estimated time, cost, or energy consumption to execute task v_i on participant node p_j; and a R matrix, of size $p \times p$, which includes criteria costs per transferred byte between any two participant nodes. For Example, time or cost to transfer n bytes of data from task v_i, scheduled on p_k, to task v_j, scheduled on p_l.

As an example of time-based selection criteria, a set of unlisted parent-trees is defined from the graph where a critical-node (CN) represents the root of each parent-tree. A CN refers to the node that has zero difference between its earliest start time (EST) and latest start time (LST).The EST of a task v_i is shown in (1). It refers to the earliest time that all predecessor tasks can be completed. ET is the average execution time of a task.

$$EST(v_i) = \max_{v_m \in pred(v_i)} \{EST(v_m) + ET(v_m)\} \quad (1)$$

Where $ET(v_m)$ is the average execution time of a task v_m, and $pred(v_i)$ is the set of immediate predecessors of v_i. The LST of a task v_i is shown in (2).

$$LST(v_i) = \max_{v_i \in succ(v_i)} \{LST(v_m)\} - ET(v_i) \quad (2)$$

Where $succ(v_i)$ is the set of immediate successors of v_i.

B. Resource Model

Our cloud system represents a heterogeneous environment since the mobile nodes have different characteristics and capabilities, The total computing capability of the real mobile nodes, hosts, within a cloud is a function of the number of hosts within a cloud and the configuration of their resources, i.e., memory, storage, bandwidth, number of CPUs/Cores, and the number of instructions a core can process per second.

C. Proposed Algorithm

We propose a generic GRPS-driven algorithm for the task scheduling and resource allocation: Adaptive List-based Scheduling and Allocation AlgorithM (ALSALAM) for mobile cloud computing. ALSALAM supports the stability of a formed cloud in a dynamic resource environment. Where, a certain resource provider is selected to run a task based on resource discovery and forecasting information provided by the GRPS. The algorithm consists of two phases as follows.

1) Initial static scheduling and assignment phase

After, the information of virtual resources is sent to the Resource Manager for the appropriate real mobile nodes' resource allocation, the Resource Manager uses its Resource Allocator unit, which interacts with the GRPS to find the available resources of every possible node a Cloud Agent could reach. The information of location, time and the computing capabilities of these resources, which match the application requirements, are obtained from GRPS. This information affects matrices of criteria for node selection. Based on the next waypoint, a destination obtained from GRPS, of each mobile node and the updated location of the Cloud Agent, we can estimate which mobile nodes will pass through the transmission range of the Cloud Agent.

A priority is assigned to a node depending on the criteria of selection. For example, in a time-based approach, we may select a host such that the highest priority is given to the nodes which are located inside the transmission range of a Cloud Agent, followed by the nodes which are located outside this transmission range and will cross it, and finally to the rest of the nodes. Within each group, nodes are listed in descending order according to the available computing capabilities, e.g. their number of cores or central processing units (CPUs). Nodes, with the same computing capabilities, are listed in descending order according to the time they will spend in the transmission range of a Cloud Agent. This could minimize the overall execution and communication time.As a result, a host list, H, is formed based on the priorities as shown in Algorithm 1 presented in Appendix.

The Cloud Agent sends the cloud formation requests, through its Communicator unit, to all resource providers to in the list of hosts H. According to the (earliest) responses received about resource available time from all responders and the criteria of selection, the responders' IDs are pushed by the Resource Manager in increasing order of parameters which reduce their costs. For example, CPUs in use in time-based approach, i.e. the responding node, R_{min} , with maximum free CPUs is on the top of responders stack RS, top(RS). This could reduce the queuing delay and therefore enhance the overall execution time.

The Task Scheduler unit of the resource manager assigns and distributes the task at the top of the list of tasks L, top(L) to the host at the top of responders stack RS, top(RS).

2) Adaptive scheduling and reallocation phase

The actual measures, e.g., time, cost or energy, required to finish a task may differ from the estimated due to both the mobility of hosts and the resource contention. For example, the mobility of hosts affects the actual finish time of a task due to the delay a host takes to submit task results to other hosts in a MCCS.

The Estimated Finish Time of a task v_i on a node p_j, $EFT(v_i, p_j)$, is shown in (3), where ERAT is the earliest resource available time.

$$EFT(v_i, p_j) = ERAT(v_i, p_j) + ET(v_i, p_j) \quad (3)$$

We propose an adaptive task scheduling and resource allocation phase to adjust the resource allocation and reschedule the tasks dynamically based on both the updated measurements, provided by the Monitoring Manager, as well as the evaluation results performed by the Performance Analyzer. The Monitoring Manager aggregates the information about the current executed tasks periodically, as a pull mode. Due to the dynamic mobile environment, hosts of a cloud update the Monitoring Manager with any changes in the status of their tasks, as a push mode. Also, hosts periodically update the cloud registry of a Cloud Agent with any changes in the status of resources. Consequently, the

 ISBN: 978-1-61208-271-4

Performance Analyzer could re-calculate the estimated measures of the submitted tasks. As a result, tasks and resources could be rescheduled and reallocated according to the latest evaluation results and measurements.

IV. EVALUATION

To simulate the MCCS environment, we have extended the CloudSim simulator [14] to support the mobility of nodes by incorporating the Random Waypoint (RWP) model. A mobile node moves along a line from one waypoint W_i to the next W_{i+1}. These waypoints are uniformly distributed over a unit square area. At the start of each leg, a random velocity is drawn from a uniform velocity distribution.

In our evaluation model, an application is a set of tasks with one primary task. Each task, or cloudlet, runs in a single virtual machine (VM) which is deployed on a mobile node. VMs on mobile nodes could only communicate with the VM of the primary task node and only when a direct ad-hoc connection is established between them. For simplicity, a primary node collects the execution results from the other tasks which are executed on other mobile nodes in a cloud. There is only one cloud in this simulation. For scheduling any application on a VM, first-come, first-served (FCFS) is followed. We only considered the initial static scheduling and assignment phases through this part of the evaluation.

For calculating the collision delay, we consider the worst case scenario, a saturation condition, where each node has a packet to transmit in the transmission range.

A. Assumptions

- Communication between nodes is possible within a limited maximum communication range, *x* (km). Within this range, the communication is assumed to be error free and instantaneous.
- The distribution of speed is uniform.
- Every mobile node can always function well all the time with high reliability and does not fail.

B. Metrics and Parameters

Preliminarily, the evaluated metric is the average application execution time, which is the time elapsed from the application submission to the application completion.

We set parameters in the simulation according to the maximum and minimum values shown in Table I. The number of hosts represents the mobile nodes that provide their computing resources and participate in the cloud.

C. Experiments

We started our evaluation by studying the effect of collision delay due to channel contention on the performance of the submitted application. In this evaluation, all nodes have the same computing capabilities, i.e. homogeneous. Fig. 2 shows the average execution time of an application at a different number of nodes, ranging from 4 to 24 nodes, in a unit square area. The average speed of a mobile node equals 10 (m/sec). We set the transmission range to be 0.8 (km), which has been obtained from an evaluation not presented here due to space limitation. At this value, we can neglect the effect of the connectivity, i.e. a node is almost always connected with others. Fig. 2 shows that the worst performance is obtained when a host has a minimum number of cores, i.e. 1 core, and at a maximum number of tasks per application, i.e. 30. This is because at a small number of nodes, e.g. 4, most of the submitted tasks will be queued in a waiting list since just one core is available per task. The more the available nodes participate in the formed cloud, the more available cores to execute these tasks. Consequently, the average execution time of an application decreases with the increase of the number of mobile nodes. The collision delay should increase with node density, while results show that the collision delay is negligible if we compare it with the queuing delay. The results at 1 and 8 cores per host are very close to each other at a small number of tasks per application, at 4 tasks/application, since there is no effect of the queuing delay. Noticeable differences between these results and the others appear at a higher number of submitted tasks/application equals 15, at a number of cores/host equals 8, due to the significant effect of the mobility of hosts. The reason is that these tasks are assigned to more nodes in the formed cloud, and this leads to increase in the communication time until the primary node collects results from the other nodes. These results show that the collision delay is also negligible if we compare it with the communication delay. Conversely, the average execution time of an application decreases when the number of nodes

TABLE I. PARAMETERS

Parameters	Values	Parameters	Values
Density of nodes	4-40 (Nodes/Km²)	Communication range	0.1-1 (km)
Number of Hosts/Cloud	4-24	Application Arrival Rate (Poisson distribution)	7 (Applications/sec)
Number of hosts /Application	2-20	Expected execution time for a task	800 (Sec)
Number of tasks/Application	4 – 140	Number of CPUs/Cores per host (Uniform distribution)	1-8
Number of applications/Cloud	1 – 14	Average Node Speed (Uniform distribution)	1.389,10,20 (m/sec)

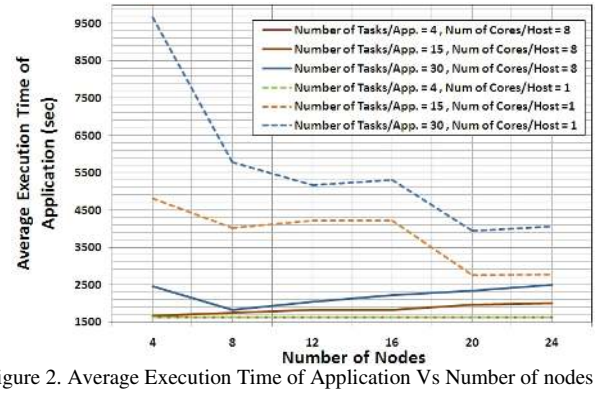

Figure 2. Average Execution Time of Application Vs Number of nodes at different number of submitted tasks/application and number of cores/host.

increases from 4 to 8 at a number of tasks per application equals 30, and at a number of cores/host equals 8. This is because the more the number of hosts, the more cores to execute these tasks. This reduces the queuing delay.

In the next experiments, we compare results of two cases: Using ALSALAM algorithm, which is based on the information obtained from GRPS, e.g. location and available processors, in resource scheduling and assignment and the random-based algorithm, which does not use this information, where a random mobile nodes are selected to execute the submitted application.

Let all 40 mobile nodes have a random number of cores, heterogeneous resources, ranging from 1 to 8 cores. Fig. 3 shows that the average execution time of an application when we consider one application is submitted to be executed. Each node has a transmission range equals 0.4 km, and its average speed equals 1.389 (m/sec). As expected, this evaluation provides significant differences between results of the two cases, with/without using the ALSALAM. The results of this figure show that executing the application on a smaller number of nodes, e.g. 8 hosts, has better performance in terms of average execution time of an application than in case of results at a larger number of hosts, i.e. 24 hosts. The higher number of submitted tasks per application leads to make some tasks waiting the previous ones in a waiting list to be executed. The total delay becomes higher if these tasks are distributed on a higher number of nodes, e.g. 24 hosts. This is because the communication delay is dominant.

We repeat our evaluation at a different number of hosts equals 4, 8 and 24 hosts, and at a different value of transmission ranges equals 0.4, and 1 (km). Fig. 4 shows that the average execution time of an application at a transmission range equals 1 (km) almost has a better performance than the case of a transmission range equals 0.4 (km) at the same number of hosts. Also, we can see that at a small transmission range, e.g. 0.4 (km), and a large number of hosts, e.g., 24 hosts, a worst performance is obtained. While, it has a better performance, at a number of hosts equals 8, than in case of a number of hosts equals 4. This observation is quite obvious because at this large number of tasks, greater than the total computing capabilities of the selected 4 hosts, the queuing delay is dominant. On the other hand, the larger the value of a number of hosts, at a high transmission range equals 1 (km), the better average

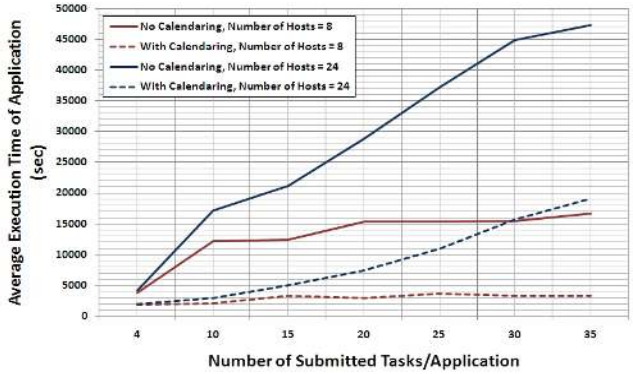

Figure 3. Average Execution Time of Applications Vs number of submitted tasks at different number of hosts.

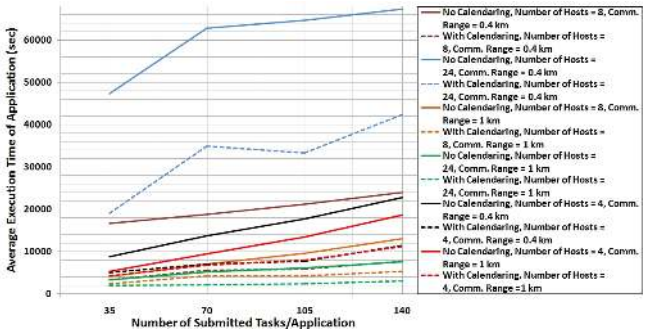

Figure 4. Average Execution Time of Applications Vs number of submitted tasks at different number of hosts and Comm. Ranges.

execution time of an application is, e.g. at 24 hosts.

The results of Fig. 5 show that the smaller the number of submitted applications, e.g. 7 applications, the better performance is obtained. Applications arrive into the system following a Poisson process with arrival rate 7. Also, the results show that the execution of submitted applications on a smaller number of hosts, e.g. 2 hosts/application, has a worst performance than of executing them on larger number of hosts, e.g. 8 hosts/application. This is because at a small number of hosts, e.g. 2, the queuing delay is dominant. The more the available number of hosts participated in the formed cloud the more available cores to execute these tasks. Consequently, the average execution time of an application decreases with the increase of a number of mobile nodes, e.g. 8 hosts/application. On the other hand, the larger the value of a number of hosts/application, the worst average execution time of an application is, e.g. at 20 hosts/application. This is because the communication delay is dominant.

D. *Findings*

Our findings can be summarized as follows.

1) There is a tradeoffbetween the communication delay and the queuing delay as a number of hosts per submitted application is varied.The higher number of hosts per an application, the higher total computing capability within the cloud is. Therefore, the queuing delay of a task is decreased. While, this leads to increase the time until the primary node collects results from other resource provider nodes, and therefore this increases the communication delay.

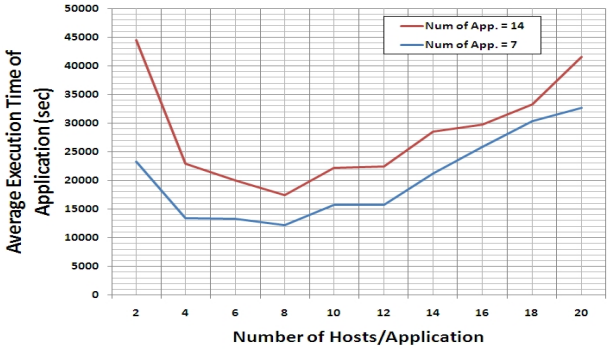

Figure 5. Average Execution Time of Applications Vs number of hosts per application at different number of applications.

2) A better performance may be obtained, at a shorter transmission range, if we select the smallest number of hosts that have computing capabilities which minimize the queuing delay to participate in a cloud. While at long transmission range of nodes, where the communication delay could be neglected, we have to select the highest number of hosts to maximize the computing capability and reduce the queuing delay.

3) The average execution time of an application is impacted by the connectivity among hosts of a cloud, the load of submitted applications, and the total resources, computing capabilities, confined in these hosts. The major factors affecting connectivity are hosts' transmission range, node mobility, and node density. The mobility is impacted by the hosts' speed and movement direction (relative to primary nodes).

V. CONCLUSION AND FUTURE WORK

We presented CARMS, a distributed autonomic resource management system to enable resilient dynamic resource allocation and task scheduling for mobile cloud computing. In addition, we proposed the GRPS-driven ALSALAM, an adaptive scheduling and allocation algorithm implemented in the resource manager of CARMS to enable efficient selection of cloud participants and to provide a stable cloud in a dynamic resource environment. Results have shown that CARMS enables effective and efficient cloud formation and maintenance over mobile devices.

Our ongoing research extends CARMS to enhance the resilience and cost efficiency of cloud management by considering the reliability and security aspects of mobile resources in the selection of cloud nodes while minimizing the execution and communication costs.

REFERENCES

[1] I. Chandrasekaran, "Mobile computing with cloud," Advancesin Parallel Distributed Computing, Communications in Computer and Information Science, vol. 203, 2011, pp. 513–522.

[2] Y. Yuan and W. Liu, "Efficient resource management for cloud computing," International Conference on System Science, Engineering Design and Manufacturing Informatization (ICSEM), China, 2011, pp.233-236.

[3] Z. Liu, W. Tong, Z. Gong, J. Liu, Y. Hu, and S. Guo, "Cloud Computing Model without Resource Management Center," International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery, 2011, pp.442–446.

[4] G. Huerta-Canepa and D. Lee, "A virtual cloud computing provider for mobile devices," Proc. 1st ACM Workshop on Mobile Cloud Computing & Services: Social Networks and Beyond, California, USA, 2010, pp.1-5.

[5] N. Fernando, S.W. Loke, and W. Rahayu, "Dynamic mobile cloud computing: Ad hoc and opportunistic job sharing," Fourth IEEE International Conference on Utility and Cloud Computing (UCC), Australia, 2011, pp.281-286.

[6] E. Marinelli, "Hyrax: cloud computing on mobile devices using MapReduce," Master thesis, Carnegie Mellon University, 2009.

[7] L. F. Bittencourt and E. R. M. Madeira, "HCOC: a cost optimization algorithm for workflow scheduling in hybrid clouds," Journal of Internet Services and Applications, vol. 2, Dec 2011, pp. 207–227.

[8] C. Lin, S. Lu, "Scheduling scientific workflows elastically for cloud computing," in IEEE 4th International Conference on Cloud Computing, USA, 2011, pp. 746 - 747.

[9] K. Bessai, S. Youcef, A. Oulamara, C. Godart, and S. Nurcan, "Bi-criteria workflow tasks allocation and scheduling in cloud computing environments," 5th International Conference on Cloud Computing, USA, 2012, pp. 638-645.

[10] B. Yang, X. Xu, F. Tan, and D. H. Park, "An utility-based job scheduling algorithm for cloud computing considering reliability factor," International Conference on Cloud and Service Computing (CSC), Hong Kong, 2011, pp. 95-102.

[11] L. Wang, G. von Laszewski, J. Dayal, and F. Wang, "Towards energy aware scheduling for precedence constrained parallel tasks in a cluster with DVFS," Proc. 10th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGRID'10, Australia, 2010, pp. 368–377.

[12] A. Khalifa, R. Hassan, and M. Eltoweissy, "Towards ubiquitous computing clouds," in The Third International Conference on Future Computational Technologies and Applications, Rome, Italy, September, 2011, pp. 52–56.

[13] A. Khalifa and M. Eltoweissy, "A global resource positioning system for ubiquitous clouds," in the Eighth International Conference on Innovations in Information Technology (IIT), UAE, March, 2012, pp. 145–150.

[14] S. K. Garg and R. Buyya, "NetworkCloudSim: modelling parallel applications in cloud simulations," Proc. 4th IEEE International Conference on Utility and Cloud Computing (UCC 2011), Melbourne, Australia, Dec. 2011, pp.105–113.

[15] M. Eltoweissy, S.Olariu, and M.Younis, "Towards autonomous vehicular clouds," Ad Hoc Networks, Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, vol. 49, 2010, pp 1-16.

APPENDIX

Algorithm 1 Initial task scheduling and assignment based on priorities

```
1:  The EST of every task is calculated.
2:  The LST of every task is calculated.
3:  Empty list of tasks L and auxiliary stack S.
4:  Push tasks of CN tree into stack S in decreasing order of
    their LST.
5:  while the stack S is not empty do
6:      If there is unlisted predecessor of top(S) then
7:          Push the predecessor with least LST first into stack S
8:  else
9:  enqueue top(S) to the list L
10:         pop the top(S)
11:     end if
12: end while
13: while the list L is not empty do
14: dequeue top(L).
15:     Send task requests of top(L) to all participant nodes in
        the list of hosts H which match the task requirements.
16:     Receive the earliest resource available time responses
        for top(L) from all responders.
17:     Empty auxiliary responders stack RS.
18:     Push IDs of hosts which respond to requests into
        responders stack RS in increasing order according to
        their CPUs in use.
19:     while the host stack RS is not empty do
20:         find the responder R_min with less CPUs in use.
21:         assign task top(L) to responder R_min.
22:         remove top(L) from the list L.
23:     end while
24: end while
```