ISSN No: - 2456- 2165

LTE Based High-Performance Mobile-Cloud Computing

Deepika Sharma
M.Tech Scholar , Computer Science
Jaipur National University , Jaipur
deepika1703.sharma@gmail.com

Sonu Mittal
Professor, Computer Science
Jaipur National University , Jaipur
sonum7772@rediffmail.com

Abstract :- The rise of the mobile-cloud computing paradigm in recent years has enabled mobile devices with processing power and battery life limitations to achieve complex tasks in real-time. While mobile-cloud computing is promising to overcome the limitations of mobile devices for real-time computing, the lack of frameworks compatible with standard technologies and techniques for dynamic performance estimation and program component relocation makes it harder to adopt mobile-cloud computing at large. Most of the available frameworks rely on strong assumptions such as the availability of a full clone of the application code and negligible execution time in the cloud. In the base paper [21] ,they present a dynamic computation offloading model for mobile-cloud computing, based on autonomous agents. Their approach does not impose any requirements on the cloud platform other than providing isolated execution containers, and it alleviates the management burden of offloaded code by the mobile platform using stateful, autonomous application partitions. They also investigate the effects of different cloud runtime environment conditions on the performance of mobile-cloud computing, and present a simple and low-overhead dynamic make span estimation model integrated into autonomous agents to enhance them with self performance evaluation in addition to self-cloning capabilities. In this research , we present the concept of LTE . LTE (Long-Term Evolution, commonly marketed as 4G LTE) is a standard for wireless communication of high-speed data for mobile phones and data terminals. It is based on the GSM/EDGE and UMTS/HSPA network technologies, increasing the capacity and speed using a different radio interface together with core network improvements. We are introducing LTE and reducing the execution time of different cloud runtime environment conditions on the performance of mobile-cloud computing, and present a simple and low-overhead dynamic make span estimation model.

Keywords-Mobile-Cloud Computing; Autonomous Agents; Context; Performance, LTE.

I. INTRODUCTION

Mobile computing devices have replaced desktops and mainframes for daily computing needs for the past decade. Despite the everyday advances in mobile computing technology, size restrictions impose limitations on the processing power and battery life of these devices, which limits their capabilities for soft real-time, computing-intensive applications such as image processing. Cloud computing offers the ability to fill the gap between the resource needs of mobile devices and availability of those resources, through the concept of mobile-cloud computing (MCC), which partitions mobile applications between mobile and cloud platforms for execution, by dynamically offloading parts of mobile computation to cloud hosts.

Achieving high performance with mobile-cloud computing requires optimal partitioning of the mobile application components between the mobile and cloud platforms based on runtime conditions, as well as the dynamic monitoring of the performance of application components during their execution. Recent work on this problem has resulted in frameworks with various partitioning and optimization techniques.

Most of these frameworks impose strict requirements on the cloud side, such as a full clone of the application code/virtual machine or special application management software, hindering wide applicability in public clouds. They are also based on the assumption that execution time on a cloud platform is negligible compared to execution on a mobile device, hence do not monitor the performance of offloaded computation, which may not hold in the case of multi-tenancy or insufficient resources on the cloud host.

A. Architecture Of Mobile Cloud Computing

The generic architecture as defined by Dinh et all[1] basically consists of a mobile device connected to mobile networks, The mobile networks then transfer the request from mobile devices to cloud for processing via internet. Figure 1 explains the basic architecture for Mobile cloud.

Fig 1. Basic Architecture of Mobile Cloud

II. RELATED WORK

ISSN No: - 2456- 2165

Significant research efforts have been put into computation offloading since the early 1990s. Upgrades in virtualization innovation, accessibility of high system data transfer capacities and distributed computing foundations expanded the achievability of constant calculation offloading from versatile to cloud stages in the previous decade. Early work in element versatile distributed computing models incorporates Clone Cloud [1] and MAUI [2], both of which allotment applications utilizing a system that consolidates static project investigation with element program profiling, and streamlines execution time and vitality utilization on the cell phone utilizing an improvement solver. The drawback of these methodologies is that they require a duplicate of the entire application code/virtual machine at the remote execution site, which is both a very strict prerequisite for open cloud machines and makes the application code defenseless against investigation by malevolent gatherings on the same stage. Yang et al. [3] propose an apportioning and execution system particularly for versatile information stream applications. Huang et al. [4] propose a low-many-sided quality offloading calculation to minimize vitality utilization on a cell phone, nonetheless they don't give points of interest of the framework engineering. Park et al. [5] propose an offloading structure restricted to JavaScript based applications. The ThinkAir [6] structure gives preferred adaptability and parallelism highlights over its forerunners, in any case despite everything it requires the presence of the complete application code on the cloud server, displaying the drawbacks of CloneCloud and MAUI. Xian et al. [7] propose a calculation offloading strategy that does not require the estimation of makespan for every application segment. They utilize online measurements of the application makespan, and the calculation is offloaded to a remote server just in the event that it is not finished on the gadget by a particular timeout, which may not be generalizable for various cell phone attributes. Mei et al. [9] propose a sharing-mindful plan for idleness tolerant instead of continuous portable cloud applications, where information sharing over different applications is misused for better outsourcing execution. Ferber et al. [10] propose a middleware structure for moving just registering escalated parts of Java applications to cloud assets, which does not consider dynamic offloading of other application parcels. Lin et al. [11] present a model for vitality mindful undertaking planning on cell phones, where errands are doled out to centers on the gadget or a cloud asset in view of their priority necessities. Their methodology offers priority to minimizing vitality utilization instead of aggregate execution time.

III. MOBILE AGENT AND JADE AGENT FRAMEWORK

Mobile computing devices have become increasingly popular during the past decade, replacing desktops and mainframes for daily computing needs. Many of these devices have limited processing power, storage and battery compared to their wall-socket-powered, tethered counterparts, which limits their capabilities for real-time, computation-intensive applications such as image processing. Computation offloading to more powerful servers is the solution to provide these devices with the resources they need to achieve complex tasks. Cloud computing, emerging as a new computing paradigm in the recent years, offers computing resources to users on demand, obviating the need to actually own those resources. With increasing popularity and availability, cloud computing has the potential to fill the gap between the resource needs of mobile devices and availability of those resources. Cloud computing is already utilized by many mobile applications today. Current mobile applications mostly involve an inflexible split of computation between the mobile and cloud platforms, following the client-server paradigm with hardcoded interactions with the server. This inflexibility prevents applications from adapting to conditions such as high network latency, which could result in poor performance when cloud resources are preferred over computation on the device. For applications requiring large data transfers to the remote server, performing the computation on the device can have better performance than relying on processing by the remote server when the device's Internet connection is poor.

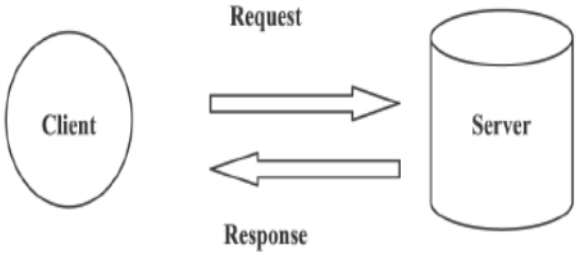
Achieving high performance with mobile-cloud computing is contingent upon an optimal partitioning of the mobile application components between the mobile and cloud platforms based on runtime conditions. Recent work on this problem has resulted in frameworks with various partitioning and optimization techniques. Most of these frameworks impose strict requirements on the cloud side, such as a full clone of the application code/virtual machine or special application management software, hindering wide applicability in public clouds.

A mobile agent is a software program with mobility, which can be sent out from a computer into a network and roam among the computer nodes in the network [13]. It can be executed on those computers to finish its task on behalf of its owner. When an application using mobile agents needs to request a service from a remote server, it gathers the required information and passes it to the agent's execution environment. At some point during its lifetime, the agent executes an instruction for migration, which results in the following sequence of events [13]:

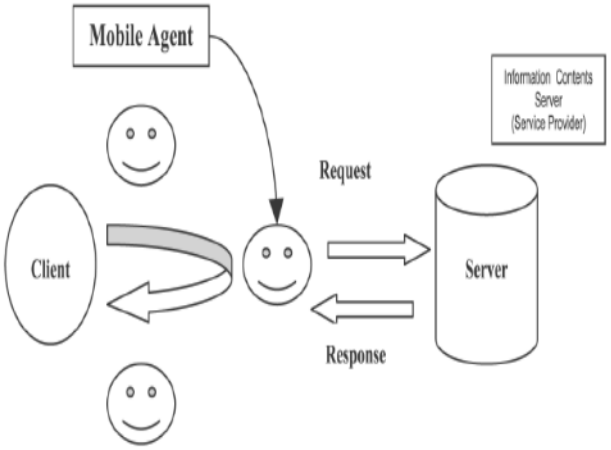
- The current agent process is suspended or a new child process is created.
- The suspended (or new child) process is converted into a message including all of its state information (process state, stack, heap, external references). This message is addressed to the destination where execution will continue.
- The message is routed to the destination server, which delivers it to the server's agent execution environment.
- The message is reconstituted to an executable and the associated process is dispatched.
- Execution continues with the next instruction in the agent program. Upon completion of the agent program at the remote server, the agent might terminate its

execution, become resident at the server or migrate back to the originating client or another server.

Figure 2 demonstrates the difference between the mobile agent paradigm and the client-server paradigm in terms of the communication involved upon the request of a client to a data server. As seen in the figure, while the client has to maintain a communication link with the service provider for the transfer of data in the client-server architecture, a mobile agent is shipped to the service provider and returns to the client only when it has collected all the requested data in the mobile agent paradigm.



(a) Client / server communication



(b) mobile agent communication

Fig 2. Mobile agent vs. client-server communication [14].

IV. AGENT-BASED COMPUTATION OFFLOADING

Figure 3 shows a high level view of the autonomous agents-based computation offloading framework, which consists of the main components described below.

A. Autonomous Application Modules

An autonomous application module is a chunk of application code packed in an autonomous (mobile) agent that is executable on a cloud host.

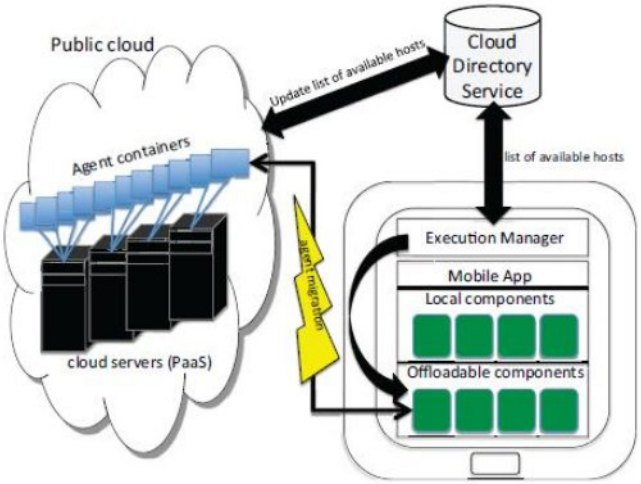


Fig 3. High level view of computation framework.

Autonomy of these application modules provides great advantages in the context of MCC due to the capability of transparently moving between mobile and cloud platforms without requiring management by their caller and self cloning to different virtual machines. Each mobile application in the framework consists of a set of autonomous agent-based application modules that are offloadable to the cloud for execution, in addition to a set of native application components that are always executed on the device due to constraints such as accessing native sensors of the device or providing the user interface of the application. Partitioning of the application into these two types of components is performed statically before installation of the application on the mobile device as described in [12]. During the offline application partitioning process, any program partition that is not computation-intensive is not set as an offloadable component even if it does not have to be pinned to the device, as this would incur additional runtime processing overhead for partitions that would likely never be offloaded. In the framework, these modules are implemented as JADE [13] agents and they exhibit all features of a mobile agent [14] including stateful execution, asynchronous communication, autonomic decision-making and migration capability.

V. PROPOSED METHODOLOGY

Today's mobile cellular networks are highly centralized and not optimized for high-volume data applications, which will evolve with 4G (e.g., LTE) and beyond technologies. Operators typically use centralized network architectures that lead to very high bandwidth requirements on core network equipment and long communication paths between users and servers, which wastes network resources and increases delay. Shared distributed mobile network architectures, are needed to avoid bottlenecks, better utilize available resources, and minimize delay.

ISSN No: - 2456- 2165

A. Long Term Evolution(LTE)

Long Term Evolution Background The evolution towards LTE started as early as 2004 when the 3GPP initiated work on the LTE radio interface, and by mid-2005 it released a technical report with the design objectives. Some of the design targets were: high data rates, low user plane latency, requirements for normal capacity and also for peak data rates, flexibility in spectrum usage, and reduced time for state changes. The motivation for the LTE included the need to ensure competitiveness of the 3G system for the future, user demand for higher data rates and quality of service, and low system complexity among others.

LTE, an abbreviation for Long-Term Evolution, commonly marketed as 4G LTE,is a standard for wireless communication of high-speed data for mobile phones and data terminals. It is based on the GSM/EDGE and UMTS/HSPA network technologies, increasing the capacity and speed using a different radio interface together with core network improvements. The standard is developed by the 3GPP (3rd Generation Partnership Project).

LTE is the natural upgrade path for carriers with both GSM/UMTS networks and CDMA2000 networks. The different LTE frequencies and bands used in different countries will mean that only multi-band phones will be able to use LTE in all countries where it is supported. Although marketed as a 4G wireless service, LTE (as specified in the 3GPP Release 8 and 9 document series) does not satisfy the technical requirements the 3GPP consortium has adopted for its new standard generation, and which were originally set forth by the ITU-R organization in its IMT-Advanced specification. However, due to marketing pressures and the significant advancements that WiMAX, HSPA+ and LTE bring to the original 3G technologies, ITU later decided that LTE together with the aforementioned technologies can be called 4G technologies. The LTE Advanced standard formally satisfies the ITU-R requirements to be considered IMT-Advanced. To differentiate LTE Advanced and WiMAX-Advanced from current 4G technologies, ITU has defined them as "True 4G".

LTE stands for Long Term Evolution and is a registered trademark owned by ETSI (European Telecommunications Standards Institute) for the wireless data communications technology and a development of the GSM/UMTS standards. However other nations and companies do play an active role in the LTE project. The goal of LTE was to increase the capacity and speed of wireless data networks using new DSP (digital signal processing) techniques and modulations that were developed around the turn of the millennium. A further goal was the redesign and simplification of the network architecture to an IP-based system with significantly reduced transfer latency compared to the 3G architecture. The LTE wireless interface is incompatible with 2G and 3G networks, so that it must be operated on a separate radio spectrum.

B. Physical Channel

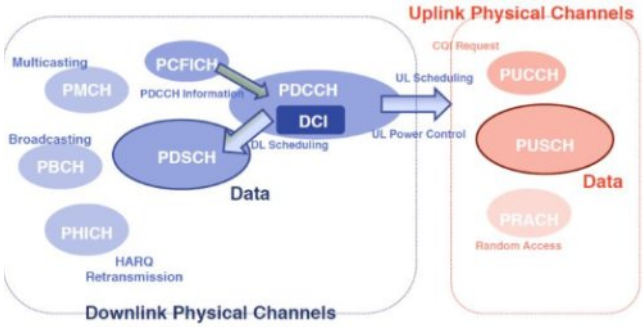


Fig 4. LTE Physical layer

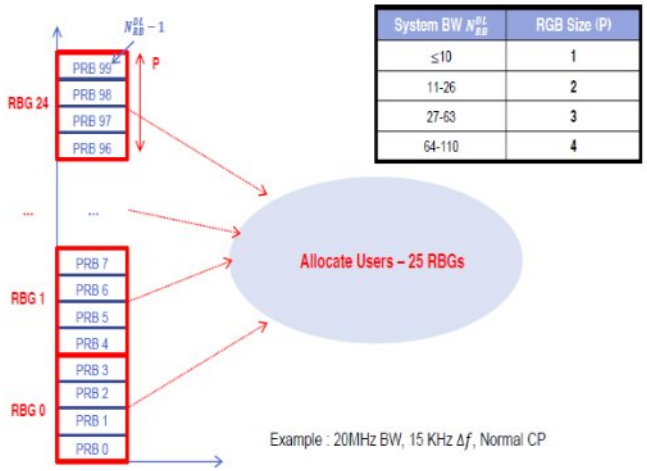


Fig 5. Resources Block Group

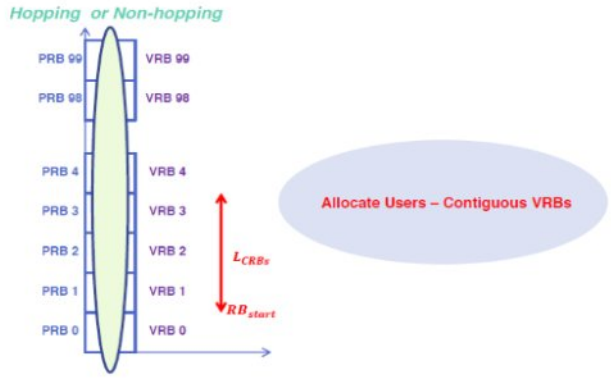


Fig 6. Virtual resources block

VI. RESULTS

The first set of experiments was performed with a face recognition application based on the program at [16], which given the picture of a person, identifies the most similar face to it in a set of pictures. This application assumes that the set of pictures to compare against is only locally available on the

ISSN No: - 2456- 2165

mobile device. if the application component processing the pictures is offloaded, the data for each picture is sent with it too. The experiments were run using an Android emulator over a period of 12 hours to capture possible variations in network conditions and the results reported are the averages for 75 runs [21] . Figure 7 provides a comparison of executing the application entirely on the mobile device vs. using the offloading framework, for different number of pictures (of 75 KB each) to compare against. We see that agent-based offloading achieves a significant 13 times shorter execution time than the device-only approach for all picture set sizes.

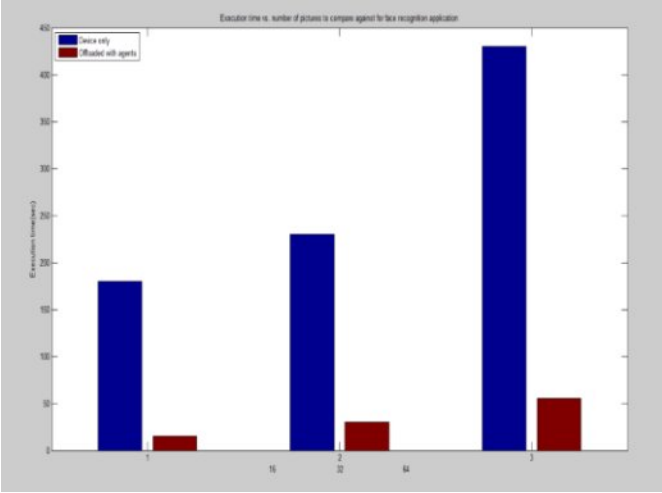


Fig 7: Execution time vs. number of pictures to compare against for face recognition application.

Figure 7 shows the effect of the host type on the execution time of the face recognition application for different number of pictures to compare against. We observe that the performance of the application degrades by 50% when a small machine instance is used instead of a medium instance.

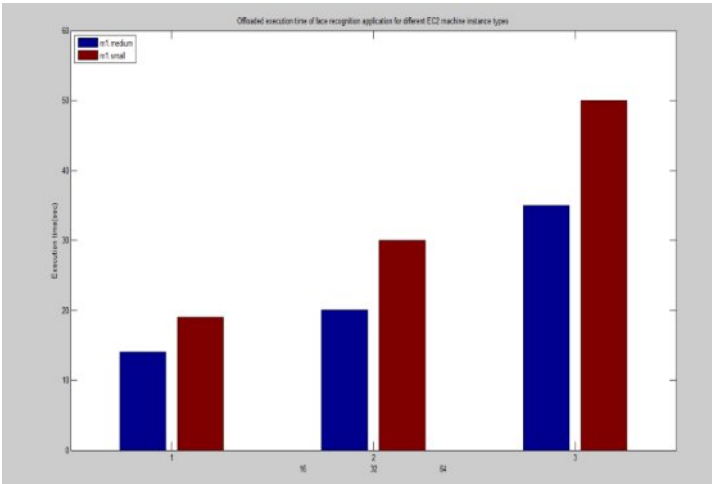


Fig 8. Offloaded execution time of face recognition application for different EC2 machine instance types.

Figure 8 compares execution times for the multi-threaded NQueens puzzle solver for different cloud host types in Amazon EC2. We observe that for 14 and 15 queens, the performance speed-up is commensurate with the number of cores in the host machine, with the 2x large machine instance taking 8 times shorter than the medium instance and 4 times shorter than the large instance for the same number of queens.

These results suggest that dynamic performance profiling has a significant effect on the performance of mobile cloud computing, and relocation to different hosts based on continuous monitoring could help increase performance.

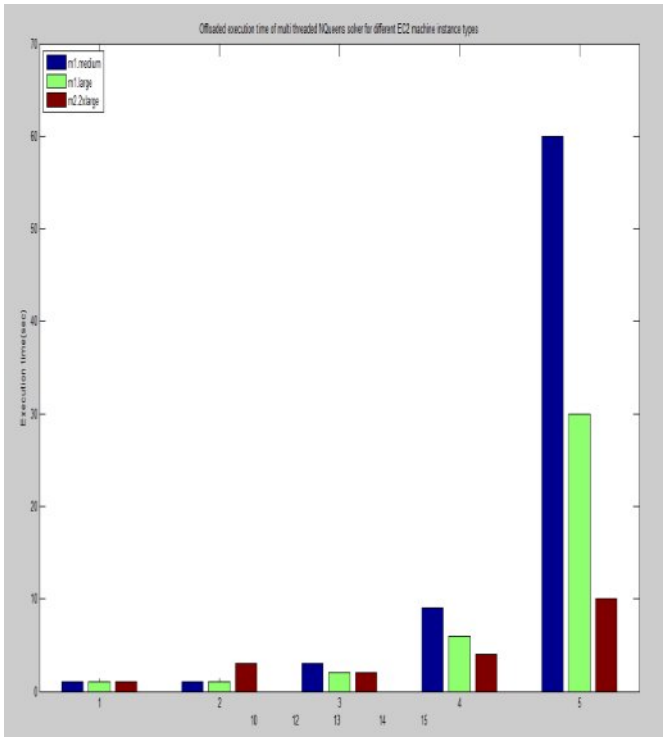


Fig 9:- Offloaded execution time of multi-threaded NQueens solver for different EC2 machine instance types.

The performance of the offloading manager was evaluated with a synthetic application consisting of 6 offloadable modules obtained from parts of the NQueens puzzle application. Each module in the application has a different execution time on the mobile device and different amount of data transfer requirements for offloading. Figure 10 shows a comparison of the total execution times of the application for the cases of (a) monolithic execution on the device, (b) complete offloading to the cloud, and (c) using the offloading manager to decide which modules to offload to the cloud, where the device's available bandwidth is upper-bounded by the connection speed types on the x-axis. The experiments were run on a Motorola Atrix 4G device for each connection speed type [21].

ISSN No: - 2456- 2165

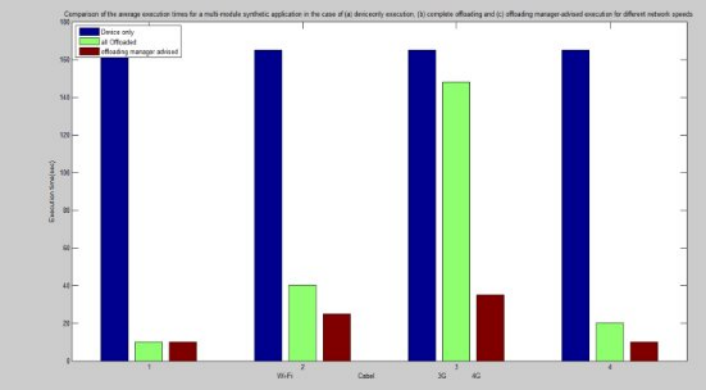


Fig 10: Comparison of the average execution times for a multi-module synthetic application in the case of (a) deviceonly execution, (b) complete offloading and (c) offloading manager-advised execution for different network speeds.

As seen in figure 10, using the proposed LTE(4G) for execution platform decisions always achieves low execution time(or same) as compare to Wi-Fi , Cabel and 3G.

Instance type	Avg. true Makespan(ms)	Avg. absolute estimation error (ms)	Absolute error percentage (%)
m1.small	15142	292	1.9
c1.medium	10147	125	1.2
m1.medium	7277	132	1.8
m1.large	7234	75	1.0
LTE m1.small	11331	133	0.6
LTE c1.medium	7211	95	0.5
LTE m1.medium	4351	83	0.7
LTE m1.large	2561	32	0.3

Table I. summarizes the average of the absolute errors and the ratio

Table I summarizes the average of the absolute errors and the ratio of the average absolute error to the average true makespans. As seen in the table, the average absolute error is less than or equal to 292 ms, which translates into less than 2% of the total makespan for all instance types, which is negligible. We also observe that the error increases slightly on instances with smaller computing power. These results prove that the proposed LTE(4G) performance is better for avg. true makespan , avg. absolute estimation error and absolute error percentage .All these three parameters gets decrease as we apply LTE.

VII. CONCLUSION

In this paper, we proposed a LTE based mobile-cloud computing using mobile agent based application partitions, imposing minimal structural requirements on the cloud. Experiments performed with two real-world applications demonstrate that the proposed framework is promising for improved performance and wide adoption in mobile-cloud computing. As we can see from the results session , in this thesis we find to problems for high execution time and high Avg. absolute estimation error. We have been reduced the execution time and Avg. absolute estimation error from the proposed methodology .

REFERENCES

[1] B. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: Elastic execution between mobile device and cloud," in *Proceedings of the 6th ACM European Conference on Computer Systems (EuroSys'11)*, 2011.

[2] E. Cuervo, A. Balasubramanian, D. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "Maui: Making smartphones last longer with code offload," in *Proceedings of the 8th ACM International Conference on Mobile Systems, Applications, and Services (MobiSys'10)*, 2010.

[3] L. Yang, J. Cao, S. Tang, T. Li, and A. T. Chan, "A framework for partitioning and execution of data stream applications in mobile cloud computing," in *Proceedings of the 5th IEEE International Conference on Cloud Computing (CLOUD'12)*, 2012.

[4] D. Huang, P. Wang, and D. Niyato, "A dynamic offloading algorithm for mobile computing," *IEEE Transactions on Wireless Communications*, vol. 11, no. 6, pp. 1991–1995, June 2012.

[5] S. Park, Y. Choi, Q. Chen, and H. Y. Yeom, "SOME: Selective offloading for a mobile computing environment," in *Proceedings of the IEEE International Conference on Cluster Computing (CLUSTER'12)*, 2012.

[6] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang, "Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading," in *Proceedings of the 31st IEEE International Conference on Computer Communications (INFOCOM'12)*, 2012.

[7] C. Xian, Y. H. Lu, and Z. Li, "Adaptive computation offloading for energy conservation on battery-powered systems," in *Proceedings of the IEEE International Conference on Parallel and Distributed Systems*, vol. 2, 2007.

[8] X. Li, H. Zhang, and Y. Zhang, "Deploying mobile computation in cloud service," in *Proceedings of the 1st International Conference on Cloud Computing (CloudCom'09)*, 2009.

[9] C. Mei, D. Taylor, C. Wang, A. Chandra, and J. B. Weissman, "Sharing-aware cloud-based mobile outsourcing," in *Proceedings of the 5th IEEE International Conference on Cloud Computing (CLOUD'12)*. IEEE, 2012.

[10] M. Ferber, T. Rauber, M. H. C. Torres, and T. Holvoet, "Resource allocation for cloud-assisted mobile applications," in *Proceedings of the 5th IEEE International Conference on Cloud Computing (CLOUD'12)*. IEEE, 2012.

ISSN No: - 2456- 2165

[11] X. Lin, Y. Wang, Q. Xie, and M. Pedram, "Energy and performance-aware task scheduling in a mobile cloud computing environment," in *Proceedings of the 7th IEEE International Conference on Cloud Computing (CLOUD'14)*. IEEE, 2014.
[12] P. Angin and B. Bhargava, "An Agent-based Optimization Framework for Mobile-Cloud Computing," *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications*, vol. 4, no. 2, pp. 1–17, 2013.
[13] Telecom Italia Lab. Java agent development framework. http: //jade.tilab.com/. Accessed: 2015-02-18.
[14] D. Chess, C. Harrison, and A. Kershenbaum, "Mobile agents: Are they a good idea?" in *Mobile Object Systems Towards the Programmable Internet*, J. Vitek and C. Tschudin, Eds. Springer, 1997.
[15] Amazon Web Services Inc. Amazon elastic compute cloud. http://aws.amazon.com/ec2. Accessed: 2015-02-18.
[16] K. Darnok. http://darnok.org/programming/face-recognition. Accessed: 2015-02-18.
[17] Motorola Mobility LLC. Motorola Atrix 4G. http: //www.motorola.com/us/consumers/Motorola-ATRIX-4G/ 72112,en US,pd.html. Accessed: 2015-02-18.
[18] Amazon EC2 instances. http://aws.amazon.com/ec2/ previous-generation/. Accessed: 2015-02-18.
[19] I. Bate, G. Bernat, G. Murphy, and P. Puschner, "Low level analysis of a portable Java byte code WCET analysis framework," in *Proceedings of the 7th IEEE International Conference on Real-Time Computing Systems and Applications*, 2000.
[20] AspectJ. http://eclipse.org/aspectj/. Accessed: 2015-02-18.

[21] Pelin Angin, Bharat Bhargava, Zhongjun Jin," A Self-Cloning Agents Based Model for High-Performance Mobile-Cloud Computing".