On Providing Response Time Guarantees to a Cloud-Hosted Telemedicine Web Service

Waqar Haider, Waheed Iqbal(✉), Fawaz S. Bokhari, and Faisal Bukhari

Punjab University College of Information Technology,
University of the Punjab, Lahore, Pakistan
{mscsf13m024,waheed.iqbal,fawaz,faisal.bukhari}@pucit.edu.pk.edu

Abstract. Traditionally healthcare services are deployed on dedicated physical systems and the functionalities are limited to the local network. Mostly, dedicated physical systems are either under-provisioned or over-provisioned. Cloud Computing technology addresses these limitations by dynamically allocating required resources to applications being hosted on such cloud platforms. In this paper, we study the viability of hosting a telemedicine service over Amazon Elastic Compute Cloud (EC2); a public cloud architecture. In particular, we study the performance of our telemedicine service under linearly increasing workloads by using multiple hosting options available in Amazon EC2. The performance analysis of our telemedicine service is based on fulfilling the specific number of requests per seconds under constraint response times. We find that dynamic resource provisioning on the web tier using medium type instances gives better results compared to static allocation using large and xlarge type instances without incurring any bottleneck issues, thereby, making it a feasible solution for telemedicine service providers.

Keywords: Cloud computing · Amazon EC2 · Telemedicine · Auto-scaling · Resource allocation · Web services

1 Introduction

Telemedicine is an enabling technology to facilitate the provision of health-care services based on information and communication technologies to serve a large population living in remote and underprivileged areas. In a typical telemedicine system, patients interact with the telemedicine server using Internet connection from a distant location by providing personal information, symptoms, and lab test reports. Once a patient's information reaches to the telemedicine server, an automated process assigns the patient to a doctor. Then the doctor provides the prescription or feedback to the patient from remote location. However, managing computing and storage resources to offer high availability and response time guarantees for a telemedicine service is quite challenging and an emerging research area. Traditionally, such healthcare services are deployed on dedicated physical systems which are mostly either under-provisioned or over-provisioned and their functionalities are limited to the local network. Since, a large number of users are accessing a particular telemedicine service concurrently from various locations, efficient resource utilization is a challenge to reduce operational cost and maintain specific application's response time.

Cloud computing technology has emerged as a promising technology for the provision of low cost, on-demand, high available, and dynamic resource provisioning services to enterprises and individual users [1]. With the increasing use of Internet applications, the demand for cloud computing services has seen an unprecedented rise recently. One of the motivating factor for the adoption of cloud platform for such applications is that, it provides high availability and better performance with a low operational cost to the hosted application. To offer these features, cloud computing model allows provisioning of resources dynamically on varying workloads [2–4]. Therefore, hosting a telemedicine service on a cloud provides better availability and response time guarantees to the end users.

There has been some research efforts in providing medical services over the cloud. For example, Shilin Lu et al. [5] have conducted several experiments to evaluate the performance of their custom designed medical service over the cloud and compared its performance with a traditional system. In another research by Jui-chien et al. [6], the authors have proposed a cloud based service which facilitates transmission and interpretation of ECG through mobile phones. Their contribution includes pre-hospital diagnosis and to enhance inter-operability of ECG results in rural and urban areas. The proposed service can be provided on running vehicles and it is claimed that the service is cheap, efficient and convenient. Charalampos et al. [7] has proposed a mobile health care system based on cloud computing. A mobile application was developed in Google's Android OS which provided patients data transmission and retrieval with the help of a mobile service. They have used Amazon's simple storage service (S3) in order to store and manage patients' data and presented a prototype of their proposed solution. Princy et al. [8] proposed a cloud based telemedicine health service in India in which the authors main focus were on providing real time video steaming by utilizing cloud services. Xiaoliang et al. [9] have presented a mobile based telemedicine service and unveiled some opportunities by which mobile cloud can be better optimized. Amitav et al. [10] proposed an implementation of a patient monitoring teledermatology system. However, none of these addresses the challenge of efficiently allocating cloud resources to minimize cost and maintain specific response time guarantees to the end users.

In this paper, we present our proposed telemedicine service and study its performance using different Amazon Elastic Compute Cloud (EC2) instance types namely medium, large, and xlarge to profile throughput and average response time on linearly increasing workloads. In addition to this, we also investigate the provisioning of dynamic resources to satisfy specific response time requirements of our telemedicine service. Our results show that dynamic provisioning helps to offer response time guarantees for the proposed telemedicine service using medium type instances on increasing workloads.

In the rest of this paper, we briefly explain our proposed telemedicine service, experimental design, and experimental results obtained using different deployment scenarios.

2 Design of Telemedicine Service

We have developed a telemedicine web service using Java Jersey [11] and MySQL database. We have deployed this service on Oracle WebLogic server. Figure 1 shows Entity Relationship Diagram (ERD) of our developed telemedicine service explaining main entities and their relationships. There are three main user roles; `patient`, `doctor`, and `admin` (administrator). The `admin` role is used to manage user accounts and access control. The `doctor` interacts with patient's `visit` and issues `prescription`. Each patient may have many visits and each visit may have multiple `visistdata` associated with it. A patient may optionally upload images and textual data with his/her `visit`. We also maintain `audit` logs of every interaction of users with the service.

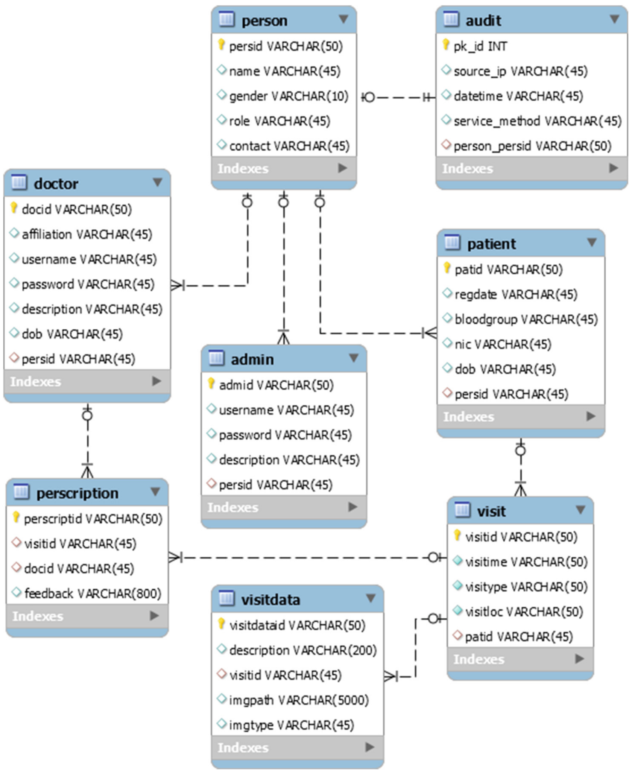

Fig. 1. *EERD of Telemedicine Web Service*

We have exposed our web service to be consumed in different client side implementations including mobile, desktop, and web applications through well defined URIs. For example, we provide specific end points in terms of URIs to perform create, read, update and delete (CRUD) operations for every entity explained in the ERD. Our proposed web service methods generate output in JSON format.

3 Experimental Setup

In this section, we describe our experimental cloud testbed, design of experiments to evaluate our proposed telemedicine web service, and workload generation method.

3.1 Cloud Testbed

We have used Amazon Web Services (AWS) to host and evaluate the performance of our telemedicine service using various different types of Elastic Compute Cloud (EC2) instances. An EC2 instance provides a virtual machine with a specific hardware resources. Table 1 shows the resource allocation and cost of EC2 instances used in our experimental evaluation. In each experiment, we have deployed web service tier and database tier on separate EC2 instances of a specific type.

Table 1. Resource allocation and cost of EC2 instances used in experiments.

Instance Type	vCPUs	Memory (GiB)	SSD Storage (GB)	Cost (USD/hour)
m3.medium	1	3.75	4	0.067
m3.large	2	7.5	32	$0.133
m3.xlarge	4	15	80	$0.266
m3.2xlarge	8	30	160	$0.532
c3.large	2	3.75	32	$0.105
c3.2xlarge	8	15	160	$0.42

3.2 Experimental Design

We have conducted five set of experiments to evaluate the performance of the telemedicine system. Table 2 provides details of the conducted experiments. In each experiment, we pre-allocate specific type of EC2 instance to web service tier and database tier and generate the synthetic workload to profile throughput (requests/second) and average response time of the application. However, in Experiment 5 we have enabled auto-scaling on web server tier using rule-based technique. We configure Amazon's auto-scale policy to increase one EC2 instance whenever average response time reaches to 1000 ms or CPU utilization of any instance allocated to web tier reaches to 70 %. We also configure Amazon's Elastic Load Balancing (ELB) service to load balance workload among allocated web tier instances.

Table 2. Experimental details.

Exp#	Experiment	Description
1	Static allocation using EC2 medium instance	Pre-allocated one EC2 instance of type `m3.medium` to web service tier and one EC2 instance of type `m3.2xlarge` to database tier
2	Static allocation using EC2 large instance	Pre-allocated one EC2 instance of type `m3.large` to web service tier and one EC2 instance of type `m3.2xlarge` to database tier
3	Static allocation using EC2 xlarge instance	Pre-allocated one EC2 instance of type `m3.xlarge` for web service tier and one EC2 instance of type `m3.2xlarge` to database tier
4	Static allocation with distributed workload generation	Pre-allocated one EC2 instance of type `m3.large` to web service tier and `c3.2xlarge` type of instance to database tier. For distributed workload generation, we used two instances of type `c3.large`
5	Dynamic allocation using EC2 medium instances	Horizontal auto-scaling enabled for web service tier using `m3.medium` EC2 instances and static allocation of `m3.2xlarge` instance type to database tier

3.3 Synthetic Workload Generation

We have used `httpef` [12] to generate a synthetic workload in linearly increasing fashion for the telemedicine service. We generate workload for 40 min emulating specific number of user session per second in a step-up fashion. A synthetic user session emulate a use case scenario to search a patient and then insert a new record of a patient. In each user session, we have two requests consisting of searching a patient which outputs a large number of patient's records from database and then issuing a put request to insert a new patient.

The workload generator is deployed on a separate EC2 instance of type `m3.2xlarge` to avoid any saturation at workload generator. However, we observe a bandwidth limitation in Experiment 2 and Experiment 3. In order to overcome this bandwidth limitation, we have distributed the workload generation using two instances in Experiment 4.

4 Experimental Results

In this section, we describe the results obtained in Experiment 1, 2, 3, 4, and 5 described in Table 2. For each experiment, we provide throughput (request/sec), average response time (milliseconds), and CPU utilization of allocated resources.

4.1 Experiment 1: Static Allocation Using EC2 Medium Instance

Figure 2 shows the throughput, average response time, and CPU utilization of EC2 instances allocated to web and database tiers during Experiment 1. It can be seen from the figure that by 16^{th} min of the experiment, the throughput stops increasing linearly and response time of the application starts increasing exponentially. It can be clearly observed that the CPU utilization of web server tier reaches near to 100 % and shows the bottleneck here. By 32^{nd} min, web server tier instance reaches to an unresponsive mode and we are unable to obtain throughput and response time metrics after this time lapse. However, we still obtain CPU utilization metrics for both instances from Amazon Cloud Watch service. The maximum throughput that we have observed in this experiment is 892 requests/second.

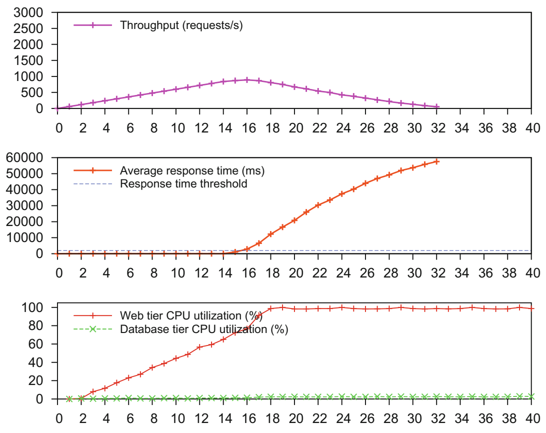

Fig. 2. Experiment 1: throughput (requests served/second), average response time, and CPU utilization of web server and database tier instances.

4.2 Experiment 2: Static Allocation Using EC2 Large Instance

Figure 3 shows the throughput, average response time, and CPU utilization of EC2 instances allocated to web and database tiers during Experiment 2. By 18^{th} min of the experiment, the throughput stops increasing linearly, however, we do not observe any dramatic growth in the response time during this experiment. The average response time remains under 50 ms. Notice, that there is no dramatic increase of CPU utilization in the web server and database tier instances. The maximum throughput achieved in this experiment is 1020 requests/second. Ideally, the throughput should have continuously increased during this experiment, however, this is because the bandwidth became the bottleneck at 18^{th} min of the experiment and web server tier instance is utilizing 201 MB/seconds and 416 MB/second in average respectively for network input and output.

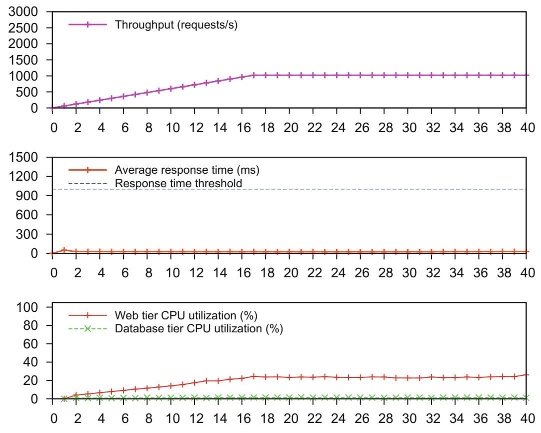

Fig. 3. Experiment 2: throughput (requests served/second), average response time, and CPU utilization of web server and database tier instances.

4.3 Experiment 3: Static Allocation Using EC2 Xlarge Instance

Figure 4 shows the throughput, average response time, and CPU utilization of EC2 instances allocated to web and database tiers during Experiment 3. It is shown in the figure that at 18th min, the throughput stops increasing linearly, however, there is no sign of any dramatic growth in response time during this experiment. The average response time remains under 50 ms. No saturation in CPU utilization of web and database tier instances has been observed. The maximum throughput we achieved in this experiment is 1020 requests/second. Notice, that the output of this experiment is similar to Experiment 2, mainly because, we observe the same bandwidth limitation in this experiment as existed in the previous one. It therefore, clearly shows that increasing resources to web tier instance does not help in overcoming bandwidth limitations.

4.4 Experiment 4: Static Allocation with Distributed Workload Generation

Figure 5 shows the throughput, average response time, and CPU utilization of EC2 instances allocated to web and database tiers during Experiment 4. It can be seen that at 25th min, the throughput stops increasing linearly and there is dramatic growth in average response time. It is also evident that the CPU utilization of web server tier reaches close to 100 % and turns to be the bottleneck in this experiment. The maximum throughput that we have achieved in this experiment is 2882 requests/second. By 31st min of the experiment, web server tier instance reaches to an unresponsive mode and we terminate the experiment at 33rd min of the experiment.

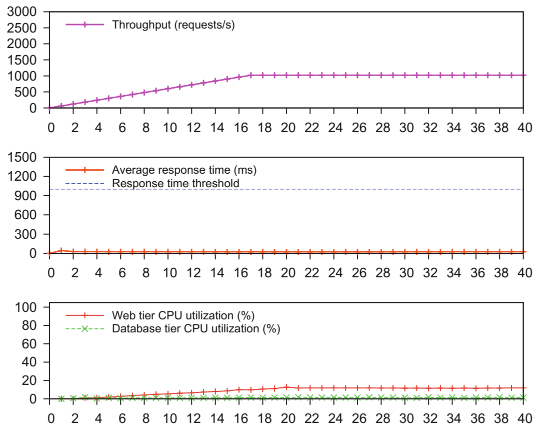

Fig. 4. Experiment 3: throughput (requests served/second), average response time, and CPU utilization of web server and database tier instances.

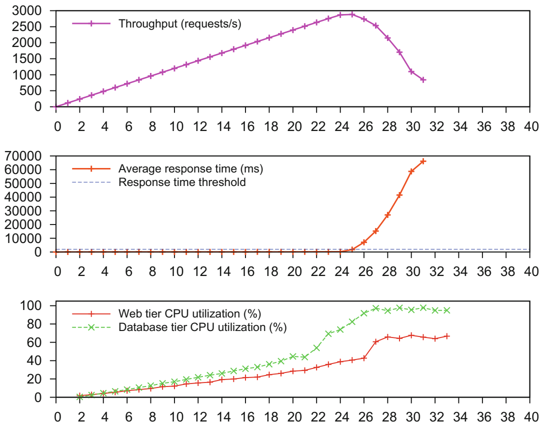

Fig. 5. Experiment 4: Throughput (requests served/second), average response time, and CPU utilization of web server and database tier instances.

4.5 Experiment 5: Dynamic Allocation Using EC2 Medium Instances

Figure 6 shows the throughput, average response time, dynamic addition of web tier instances, and CPU utilization of EC2 instances allocated to the web and database tiers during Experiment 5. It can be seen from the figure that at 17th min, the average response time crosses the acceptable response time threshold and our auto-scale policy kicks in, invokes another instance and adds it to the web tier. As soon as, the effect of the newly added instance is realized, the response time again reaches under acceptable threshold. However, at 37th min, CPU of the web tier instances cross the acceptable threshold of CPU utilization and then another instance is added to the web tier dynamically in order to cope up with the situation. Notice, that the throughput of our telemedicine application linearly increases in this experiment except at the times when response time violation occurred. The maximum throughput we have achieved in this experiment is 2398 requests/second. It is noteworthy to mention here, that in this experiment we have not observed any bottleneck resources. Therefore, we believe that using `m3.medium` instance with auto-scaling for web tier and `m3.2xlarge` type of instance for database tier would help us to offer response time guarantees to the users of our proposed telemedicine service without observing any bottleneck resources.

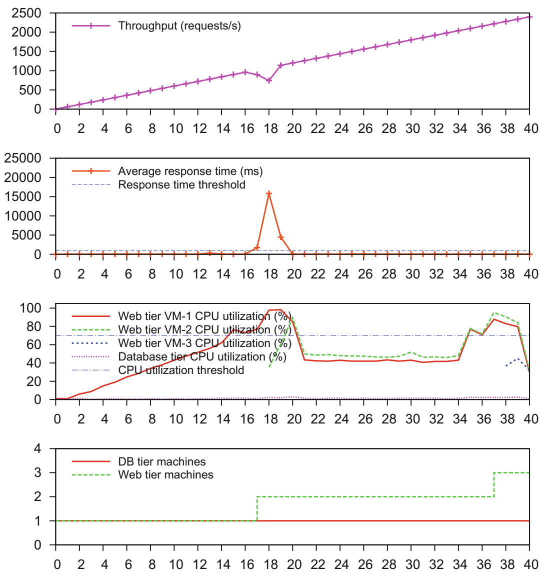

Fig. 6. Experiment 5: throughput (requests served/second), average response time, dynamic addition of web tier instances, and CPU utilization of web server and database tier instances.

5 Conclusion and Future Work

In this paper, we presented our developed telemedicine service and studied its performance using different Amazon EC2 instances on linearly increasing workloads. We found that dynamic resource provisioning on the web tier using medium type instances gives better results compared to static allocation using large and xlarge type instances without showing any bottleneck resources. We believe that this study would help the telemedicine service providers to use appropriate cloud resources in order to offer response time guarantees with minimal operational cost.

Currently, we are investigating the possibility of using NoSQL-based database to dynamically scale-out database tier instead of statically allocating over-provisioned resources to the database tier. It may greatly help to further reduce operational cost effectively.

Acknowledgments. We would like to thank Amazon Web Services (AWS) for providing us with a generous grant in terms of credits to use its cloud computing resources. The grant was part of its Amazon Educational Grant AWS Fund.

References

1. Buyya, R., Yeo, C.S., Venugopal, S.: Market-oriented cloud computing: vision, hype, and reality for delivering it services as computing utilities. In: Proceedings of the 2008 10th IEEE International Conference on High Performance Computing and Communications, Computer Society, HPCC 2008, pp. 5–13. IEEE, Washington, DC (2008)
2. Bodik, P., Griffith, R., Sutton, C., Fox, A., Jordan, M., Patterson, D.: Statistical machine learning makes automatic control practical for internet datacenters. In: Proceedings of the Workshop on Hot Topics in Cloud Computing, HotCloud 2009 (2009)
3. Iqbal, W., Dailey, M.N., Carrera, D., Janecek, P.: Adaptive resource provisioning for read intensive multi-tier applications in the cloud. Future Gener. Comput. Syst. **27**(6), 871–879 (2011)
4. Inc, A.: Amazon Web Services auto scaling (2009). http://aws.amazon.com/autoscaling/
5. Shilin, L., Ranjan, R., Strazdins, P.: Reporting an Experience on Design and Implementation of e-Health Systems on Azure Cloud (2013). http://arxiv.org/abs/1306.3624/
6. Hsieh, J.-c., Hsu, M.-W.: A cloud computing based 12-lead ECG telemedicine service (2012). http://www.biomedcentral.com/1472-6947/12/77/
7. Doukas, C., Pliakas, T., Maglogiannis, I.: Mobile healthcare information management utilizing Cloud Computing and Android OS (2010). http://www.ncbi.nlm.nih.gov/pubmed/21097207
8. Matlani, P., Londhe, N.D.: A cloud Computing Based Telemedicine Service (2013). http://www.biomedcentral.com/content/pdf/1472-6947-12-77.pdf
9. Wang, X., Gui, Q., Bingwei Liu, Y., Chen, Z.J.: Leveraging Mobile Cloud for Telemedicine: A Performance Study in Medical Monitoring (2013). http://harvey.binghamton.edu/ychen/NEBEC_2013.pdf
10. Mahapatra, A., Dash, M.: Design and Implementation of a Cloud based TeleDermatology System (2013). http://www.ijert.org/view-pdf/2269/design-and-implementation-of-a-cloud-based-teledermatology-system
11. Corporation, O.: Jersey: RESTful Web Services in Java (2010). https://jersey.java.net/
12. Mosberger, D., Jin, T.: httperf: A tool for measuring Web server performance. In: First Workshop on Internet Server Performance, pp. 59–67. ACM (1998)