An Autonomic Cloud Environment for Hosting ECG Data Analysis Services

Suraj Pandey[1], William Voorsluys[1], Sheng Niu[1], Ahsan Khandoker[2], Rajkumar Buyya[1]

[1]Cloud Computing and Distributed Systems Laboratory
Department of Computer Science and Software Engineering
The University of Melbourne, Australia
{spandey, williamv, sniu, raj}@csse.unimelb.edu.au

[2] Electrical and Electronic Engineering
The University of Melbourne, Australia
ahsank@unimelb.edu.au

Abstract

Advances in sensor technology, personal mobile devices, wireless broadband communications, and Cloud computing are enabling real-time collection and dissemination of personal health data to patients and health-care professionals anytime and from anywhere. Personal mobile devices, such as PDAs and mobile phones, are becoming more powerful in terms of processing capabilities and information management and play a major role in peoples daily lives. This technological advancement has led us to design a real-time health monitoring and analysis system that is Scalable and Economic for people who require frequent monitoring of their health. In this paper, we focus on the design aspects of an autonomic Cloud environment that collects peoples health data and disseminates them to a Cloud based information repository and facilitates analysis on the data using software services hosted in the Cloud. To evaluate the software design we have developed a prototype system that we use as an experimental testbed on a specific use-case, namely, the collection of electrocardiogram (ECG) data obtained at real-time from volunteers to perform basic ECG beat analysis.

1 Introduction

Advancing the field of health informatics has been listed as one of the 14 engineering grand challenges for the 21st century [7]. Activities in this field include acquiring, managing, and using biomedical information, from personal to global levels, to enhance the quality and efficiency of medical care and the response to widespread public health emergencies. Particularly, in the personal level, biomedical engineers envision "a new system of distributed computing tools that will collect authorized medical data about people and store it securely within a network designed to help deliver quick and efficient care" [7].

In this direction, several technological advances and new concepts, such as wearable medical devices, Body Area Networks (BANs), pervasive wireless broadband communications and Cloud computing, are enabling advanced mobile health-care services that benefit both patients and health professionals. Specially, they enable the development of a system to perform remote real-time collection, dissemination and analysis of medical data for the purpose of managing chronic conditions and detecting health emergencies. For example, by leveraging a mobile phone processing capability, its integration with body sensors, and its Internet access, a personal health monitoring system could alert health professionals when the patient needs attention, or even perform automatic intervention, e.g. trigger automatic release of drugs into the body when necessary.

The usefulness of a pervasive health information system is clear to those who require continuous monitoring but often reside far from their health service provider and have difficulty attending frequent therapy sessions. This need and the availability of the aforementioned technologies has led us to envision and design a Cloud computing-based real-time health monitoring and analysis framework capable of aiding health-care professionals better manage patient bases by reducing or eliminating on-site consultations.

Our objective is to propose an architecturally generic Cloud-based system to accommodate multiple scenarios where patients need to be remotely monitored and recorded data must be analysed by a computing system and become available to be visualised by specialists or by the patients themselves. Although our design and prototype are generic to accommodate several use cases, in this paper, we focus on one motivational case, namely: the monitoring of patients who suffer from cardiac arrhythmias, requiring continuous episode detection. Electrocardiogram (ECG) data from commodity wearable sensors are obtained in real-time and used to perform episode detection and classification.

The overall functionality of an ECG monitoring and analysis system involves the following steps:

1. A patient is equipped with a wireless ECG sensor attached to their body and a mobile device that is capable of communicating to the Internet;
2. The wireless ECG sensor module collects patient's data and forwards it the mobile device via Bluetooth without user intervention;
3. A client software in the mobile device transmits the data to the ECG analysis Web Service, which is hosted by a Cloud computing-based software stack. This communication can happen with a home wireless gateway or directly via the mobile's data connectivity (e.g. mobile 3G network);
4. The analysis software carries out numerous computations over the received data taking the reference from the existing demographic data, and the patient's historic data. Computations concern comparison, classification, and systematic diagnosis of heartbeats, which can be time-consuming when done for long time periods for large number of users;
5. The software then appends the latest results to the patient's historic record maintained in private and secure Cloud-based storage, so that authenticated users can access it anytime from anywhere. Physicians will later interpret the features extracted from the ECG waveform and decide whether the heartbeat belongs to the normal (healthy) sinus rhythm or to an appropriate class of arrhythmia;
6. The diagnosis results are disseminated to the patient's mobile device and/or monitor, their doctor and/or emergency services at predefined intervals;
7. The monitoring and computing processes are repeated according to user's preference, which may be hourly or daily over a long period of time.

1.1 Challenges and Approach

To design a computing system architecture that efficiently supports the above mentioned activities, numerous challenges need to be addressed, including scalability and cost restrictions. Cloud computing fits well as an enabling technology in this scenario as it presents a flexible stack of computing, storage and software services at low cost. We now discuss these challenges in detail and explain in high level how we tackle them by leveraging various Cloud computing services in our ECG monitoring and analysis use case. Specific architectural details and examples are described in Section 4.

Scalability: In order to support efficient monitoring and automated analysis of large patient populations, it is essential to have an infrastructure that provides high throughput, high volume storage and reliable communication. We are especially interested in two scalability measures: a) horizontal scalability – the ability for a system to easily expand its resource pool to accommodate heavier load; b) geographic scalability – the ability to maintain performance, usefulness, or usability regardless of expansion from concentration in a local area to a more distributed geographic pattern. Similarly, the system must be able to contract its resource pool in situations when the load decreases. In the Cloud computing model, the ability of a system to seamlessly expand and contract is known as elasticity.

In our use case, the system may be used by a variable number of users located in different locations. In addition, configuration preferences available in the mobile software allow users to adjust the reporting frequency in which readings are sent for remote analysis. Our software architecture must be able to seamlessly handle the changes that these preferences cause in request pattern. For this purpose, our architecture encompasses services deployed at all three layers of the Cloud computing stack, i.e software, platform and infrastructure levels.

A scalable web server hosts a Web Service (the system's front-end) that receives and manages the distribution of user requests to subsequent components. At the platform level, we employ a middleware software that manages available resources and scheduling of computing tasks onto them. In other words, the middleware manages the system's elasticity, scaling compute resources so that ECG analysis results can be delivered quick enough to maintain a user acceptable Quality of Service (QoS). At the infrastructure level, we rely on third-party Infrastructure-as-a-Service providers, which offer pay-as-you-go compute and storage services that can be deployed in various geographical regions.

Economy: Our system offers a Software (ECG monitoring and analysis) as a Service for public users, who will pay for it on per-analysis basis. At the same time, our system is also a consumer of infrastructure level Cloud services. Cloud computing service providers charge its users according to a pay-as-you-go model. There are costs associated with the use of computation, network bandwidth, storage, software services, monitoring, accounting, and content delivery. In order to attract and retain end-users, we need to maintain a high standard of QoS at minimal cost to users, while minimizing the system's own underlying cost. To accomplish this objective, the middleware aims at maximizing resource utilization by judiciously distributing and redistributing workload to existing or newly provisioned resources. The challenge in this case is being able to simultaneously satisfy QoS and lower resource usage cost.

The remainder of this paper is organized as follows: Section 2 lists related work and discusses the uniqueness of our approach; Section 3 describes the ECG data analysis process and how we model the problem as a workflow; Section 4 presents details of the implemented prototype system; Section 5 discusses experimental results; finally, Section 6 concludes our paper listing future work.

2 Related Work

The focus of this work is proposing a novel architectural design and a use case for integrating Cloud and mobile computing technologies to realize a health monitoring and analysis system. Our design makes use of all abstraction levels of the Cloud stack. To the best of our knowledge there is no work that studies such comprehensive integration. However several works in the biomedical engineering and computer science areas approach the automation of personal healthcare systems using similar technologies (i.e. mobile computing, body sensor networks, and high performance computing).

Jones et al. [5] propose an architecture for mobile management of chronic conditions and medical emergencies. It focuses on defining generic mobile solution, in the sense that it is not limited to a particular condition but can be adapted to different clinical applications. It is also designed to be easy to wear and use and as unobtrusive as possible. Base on this architecture the authors implemented and trialled two systems, namely: (i) Personal Health Monitor, which focuses on personal/local monitoring and most of the processing is done by the mobile phone itself; and (ii) MobiHealth, which contains a processing and storage back-end to suit applications that require higher processing capability. Our architecture shares many of the objectives of [5], but with a stronger aim in the processing and storage back-end, which takes scalability, economy and QoS issues into account.

Analysis of heartbeat waveforms can be time-consuming and hence automated computer-based processing of ECG data serves as a useful clinical tool. One of the major tasks to be provided is the accurate determination of the QRS complex [6]. Several algorithms have been proposed to accurately detect and classify these signals by applying various signal processing techniques, including wavelet transforms [1], neural networks [4], and genetic algorithms [1] (also refer to Kohler et al. [6] for a comprehensive survey of QRS complex detection methods). Research in this area points to the need of more accurate analysis methods, which usually results in more compute and data intensive techniques. This fact reinforces the importance of using novel software and hardware platforms, such as our Cloud-based architecture.

In more general context, cloud computing technologies have been evaluated and considered viable to support scientists on their computational requirements. For instance, Deelman et al. [3] carried out a study to assess the cost of doing science in the cloud by renting compute and storage resources from Amazon Web Services to run a scientific workflow, and concluded that costs could be reduced with little impact on performance. Technologies, such as MapReduce and Dryad have also been evaluated in the scientific context to support data analysis problems that traditionally relied on MPI-style parallel programming [10]. Additionally, Amazon Web Services has recently announced specialised support for high-performance computing applications through its Cluster Compute Instances [12], which offer a set of virtual machines linked via a fast network interconnect.

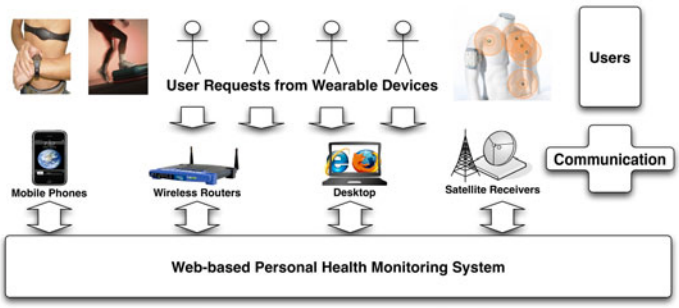

Figure 1: A software system that integrates mobile and Cloud computing services.

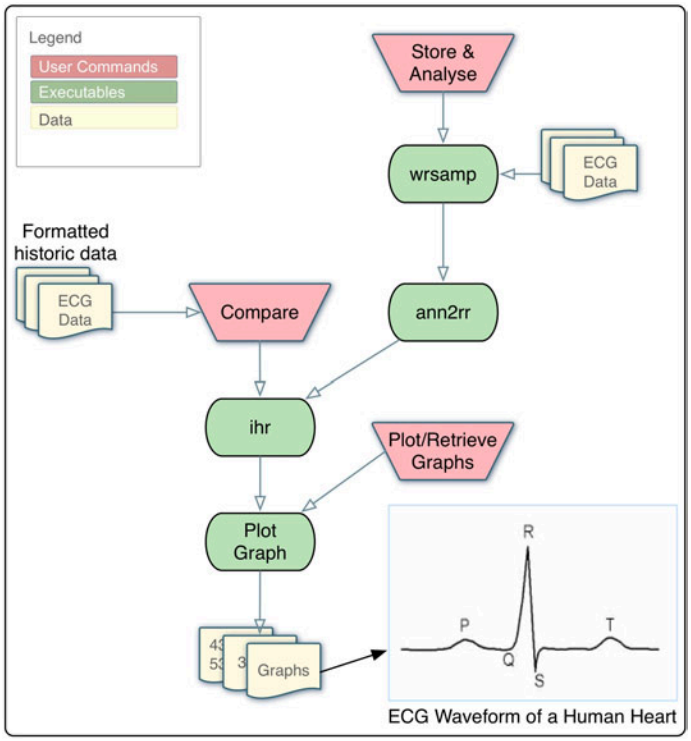

Figure 2: A workflow depicting ECG analysis.

Ranabahu et al.[11] identified the lack of scaling strategy in Cloud middleware. They propose a horizontally replicating best practice that include load balancing layer, application server layer and database layer. Their scaling strategy is horizontal replication and includes rules that trigger replications, very similar to how we replicate the workflow engine container.

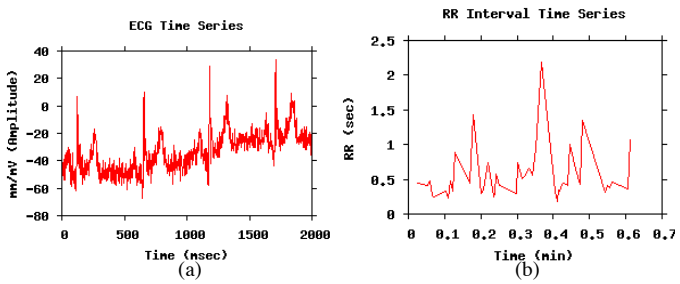

Figure 3: Graphs depicting results obtained after the analysis of a heart beat data (obtained from a volunteer). (a) The ECG time series graph clearly showing the P,Q,R,S and T points. (b) The RR interval time series graph of the ECG graph plotted in (a).

3 ECG Data Analysis: A Case Study

ECG is the electrical manifestation of the contractile activity of the hearts myocardium. The P, QRS, and T waves characterise the ECG waveform, as depicted in Figure 2 and 3(a). The most prominent feature is the QRS complex, where R denotes the peak of QRS complex. The ECG remains the most common non-invasive method for diagnosing heart diseases. Any disturbance in the regular rhythmic activity of the heart (amplitude, duration, and shape of rhythms) is known as arrhythmia. ECG is a low-cost, non-invasive test for cardiac monitoring, which has become the common diagnostic tool. Certain cardiac arrhythmias occur occasionally and up to a few days of ECG recording may be required using a Holter monitor in order to capture these beats and episodes. However, Holter monitors are used to record ECG data only and the analysis is performed offline. Therefore, a continuous cardiac monitoring and online analysis system could detect these rare episodes of cardiac arrhythmias as they occur. Identifying an arrhythmia requires the classification of heartbeats. The rhythm of the ECG signal can then be determined through the classification of consecutive heartbeats.

Figure 2 depicts a workflow for a simple analysis of raw ECG data. We chose this analysis process by referencing the PhysioNet tutorial [9]. This analysis process serves as a general analysis (not a clinical diagnosis) as also outlined by several authors [2, 14]. In depth analysis of the ECG data is out of scope for this paper.

The raw ECG data is the numerical readings of the signals obtained by placing electrodes at limbs of a subject. Table 1 briefly describes the functions of each of the tasks used in the workflow of Figure 2.

The results (numerical data and graphs) obtained after the execution of the ECG application are all stored in Amazon S3. Figure 3 depicts two of these results. They are organized and stored in the Cloud storage according to user details such as: user-id, age, gender, sleep time, etc. Categorizing the data helps in further analysis of the results at the comparison stage.

Table 1: Description of tasks used in ECG analysis.

Task Name	Description
wrsamp	Reads the raw data to produce a binary file with specified sample frequency, gain, format, etc
ann2rr	Creates an annotation file from ECG data; creates RR interval series from ECG annotation files
ihr	Reads an annotation file and produces an instantaneous heart rate signal
Plot Graph	Plots the graphs of the heart rate signal, RR interval, etc.

Apart from the ECG analysis, Figure 2 also depicts user directed commands, such as: store & analyse, compare, and plot/retrieve graphs. Store & analyse is issued by the mobile client without the user's intervention at regular intervals. This command initiates the process starting from sampling the ECG data (wrsamp) to plotting the graphs and storing in the Cloud storage. Medical practitioners (also applicable for users), who want to analyze historic data of patients, can issue the 'compare' command to compare all available ECG data of any patient. The 'plot/retrieve graph' command simply returns already computed result to the user by retrieving from the Cloud storage. These set of functions used in the workflow are for demonstration purposes only and thus can be expanded according to user requirements.

4 System Design

The advent of Cloud computing has enabled us to host the software pack that analyses ECG data as Software-as-a-Service (SaaS). The SaaS layer contains the tools for conducting custom designed analysis of current and historic ECG data of all users. We depict this in Figure 4, where the boxes represent the SaaS, PaaS, and the IaaS layers, in top-down order, respectively. This software is hosted as a web-service such that any client-side implementation can simply call the underlying functions (analyze, upload data, etc.) without having to go through the complexities of the underlying middleware. The PaaS layer controls the execution of the software using three major components: (i) Container scaling manager, (ii) Workflow Engine [8], and (iii) Aneka [13].

The workflow engine is hosted inside a container (e.g. Tomcat container). The engine manages the execution of tasks of the ECG application workflow depicted in Figure 2. As the number of requests from users grow, the container scaling manager instantiates more containers so that the user requests are distributed to workflow engines. This load balancing is done at run-time based on two parameters:

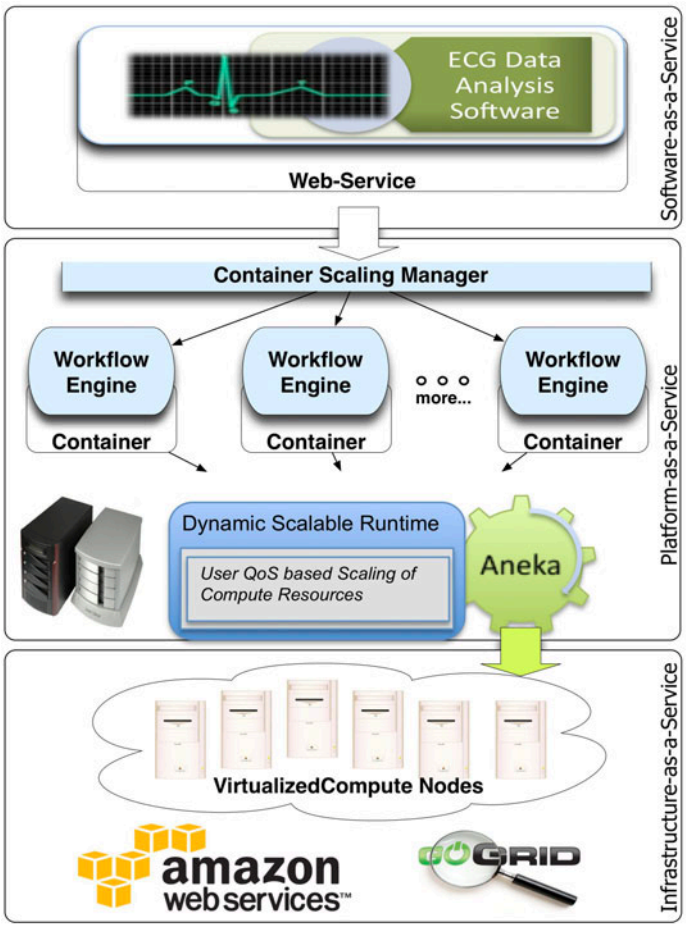

Figure 4: Components of the personal health monitoring system.

(i) the number of requests queued at any workflow engine waiting to be scheduled, (ii) the average number of requests a container managed within a certain period of time (e.g. 1 hour).

The workflow engine packages the tasks and submits them to Aneka. Aneka is a workload distribution and management platform (PaaS middleware) that accelerates applications in Microsoft .NET framework environments. As Aneka does not differentiate the tasks submitted by the workflow engine, it submits any task waiting in its queue to the first available resource. Aneka is thus responsible to handle the communication between the underlying infrastructure layer (IaaS) and the PaaS layer using a master-worker framework. The Aneka master is the service running in the PaaS layer, whereas the Aneka workers are the application executors installed in every VM instantiated at the infrastructure level.

The "Dynamic Scalable Runtime" (DSR) module, which we implemented as part of Aneka scheduling environment, is responsible for maintaining the QoS of the applications running as SaaS (e.g. ECG analysis software). For the ECG application, the 'response time' is the only QoS parameter we take into account in this paper. The DSR module keeps track of response time of tasks submitted in the past (limited time-frame), averages them and makes a decision whether to instantiate more VMs if the average response time has increased, or shut-down and release VMs if the response time has decreased than a pre-specified threshold value. This simple dynamic scaling in/out of VMs helps maintain the user defined response time.

5 Performance Analysis

In this section, we present the experimental results obtained as part of a demonstration at the Third IEEE International Scalable Computing Challenge (SCALE 2010) held in conjunction with the 10th IEEE International Symposium on Cluster, Cloud and Grid Computing (CCGrid 2010), Melbourne, Australia, May 17-20, 2010 (Figure 7).

5.1 Experiment Setup

Table 2 lists the characteristics of the compute resources we used during our experiment. We hosted Aneka PaaS layer on Amazon EC2 infrastructure, scaling manager and the workflow engine on a machine located at the University of Melbourne. Both Aneka master and worker nodes were hosted in Amazon EC2.

What is the problem? With a fixed number of Compute resources (worker nodes) and fixed number of application managers (Workflow Engine running in fixed number of Tomcat Containers), it is NOT possible to simultaneously handle increasingly large number of user requests and satisfy quality of service (QoS = response time in our application).

Proposed solution: Dynamically scale out/in the worker nodes and the application manager (workflow engine under Tomcat Container) so that any number of user requests can be served in parallel by dynamically created VM instances.

We implemented a simple heuristic to support our solution in the DSR module depicted in Figure 4:

1. If 'average' user response time is above the threshold value (e.g. 1 minute), instantiate more resources (worker nodes) until the response time decreases below a threshold.

2. If user requests are not forwarded to the Aneka Cloud timely, instantiate more containers with workflow engine at the PaaS layer until Aneka resource utilization increases to more than K% (e.g. 20%) of initial value.

Setup 1: We fix the number of Amazon EC2 resources to 25 VM instances and increase user requests from 80 to 2000 over time. We then monitor the response time of each of these requests. We depict the response time for this scenario in Figure 5[a-d].

Table 2: Characteristics of compute resources used in the experiment.

HW — SW	**Aneka Master**	**Aneka Workers**	**Workflow Engine running under Tomcat Containers**
CPU	2 virtual cores with 2.5 EC2 Compute Units	1 virtual core with 1 EC2 Compute Unit	4 Cores Intel(R) Xeon(TM) 2.80GHz
Memory	1.7 GB	1.7 GB	2 GB
Storage	320 GB	160 GB	320 GB
Platform	32 bit Windows	32 bit Windows	64 bit Linux

Setup 2: We use a simple dynamic resource allocation policy to instantiate Amazon EC2 resources at run time based on the response time (upto a maximum of 50 VMs) and increase user requests from 80 to 2000. We then monitor the response time for each request. We depict the response time for this setup in Figure 5[a-d]. The results obtained from this setup are then compared to that of Setup 1.

Setup 3: We use only one container running the workflow engine that accepts all the user requests to be forwarded to Aneka. The response time for this is depicted in Figure 6(a).

Setup 4: We use multiple containers, each running a workflow engine. The container scaling manager decides the instantiation of these containers according to the number of user requests, as described in Section 4. The response time obtained for this is depicted in Figure 6(b).

5.2 Description of Results

Setup 1 represents a non-scalable and costly method for executing the ECG application. The compute resources are fixed to 25 compute nodes (VMs), all the VMs are initially instantiated and there is no means of adding more VMs even when the response time increases due to increasing number of user requests. Initially, when the number of user requests were low ($\sim$80 requests), the response time gradually increased from 30 seconds to 90 seconds, within the first 30 seconds of job submission (Figure 5(a)). We then increased the number of user requests to upto 2000 gradually and monitored the response time within the first 300, 1500, and 3000 seconds, as depicted in Figures 5[b-d], respectively.

In Setup 2, we used a simple dynamic resource provisioning policy so that we could scale in/out according to the response time and also reduce the cost of using Cloud resources. Initially, we started with 2 VMs serving as worker nodes. This resulted in higher response time for user requests less than 80 than in the static case (Figure 5(a)). But, when the response time started to increase, the dynamic provisioning policy added more worker nodes to the system. This runtime provisioning helped decrease the response time, as clearly indicated by the rise and fall of the response time in Figure 5(a). Similar to Setup 1, we continued to increase the number of user requests and monitored the response time for 300, 1500, and 2000 seconds.

Table 3 compares the response time between Setup 1 and Setup 2. In this table, 25-minRT and 25-MaxRT represent the minimum and maximum response times when using 25 compute resources (Setup 1), respectively. Similarly, 50-minRT and 50-MaxRT represent the minimum and maximum response times when using upto 50 compute resources (Setup 2), respectively. Obviously, the response time we recorded using dynamic provisioning policy (Setup 2) was lower than that given by the static setup (Setup 1) for large number of requests. This is because Setup 2 had an additional 25 VMs to use than Setup 1. But, dynamic provisioning still outweighed the static method when we measured the percentage improvement in **reducing** the maximum response time.

The quantitative difference between the two schemes (static vs dynamic) can be solely measured by the percentage improvement (i.e. decrease in response time) in the *maximum response time*. From the values tabulated in Table 3, the percentage improvement for MaxRT starts from 5% and does not increase significantly upto 160 tasks (second row of Table 3). But, when the number of tasks was increased beyond 320, the percentage improvement is in double digits (upto 44.4%). Thus, the MaxRT value decreases for Setup 2, clearly lowering the average response time for all the tasks. These results are clearly in line with the principle that tries to minimize the maximum value of a set (MaxRT in our case) in order to get a global minimum of the set (get an upper bound on the MaxRT), so that the average value is lowered. This is supported by the trend lines we plotted for both the policies in Figures 5[a-e].

The trend lines (polynomial and/or linear fit of data) reflect the limitation on scalability of the system when using static provisioning for large number of requests. For example, in Figures 5[b-d], as we increased the number of requests, the static provisioning policy took longer to complete the requests with a higher response time than the dynamic provisioning policy. When the system is scaled out depending on the number of requests, the overall response time is much lower than in a rigid system as shown by Figure 5(d).

But, the dynamic provisioning policy performed poorly

Table 3: Difference between minimum Response Time (minRT) and Maximum Response Time (MaxRT) when using static (25 VMs) vs. dynamic (upto 50 VMs) resource provisioning policies.

#Tasks	25-minRT	50-minRT	%Improvement	25-MaxRT	50-MaxRT	%Improvement
80	25	15	40%	95	90	5.2%
160	40	25	37.5%	110	105	4.5%
320	50	50	0%	275	225	18.2%
640	55	50	9%	700	600	14.2%
2000	200	100	50%	2700	1500	44.4%

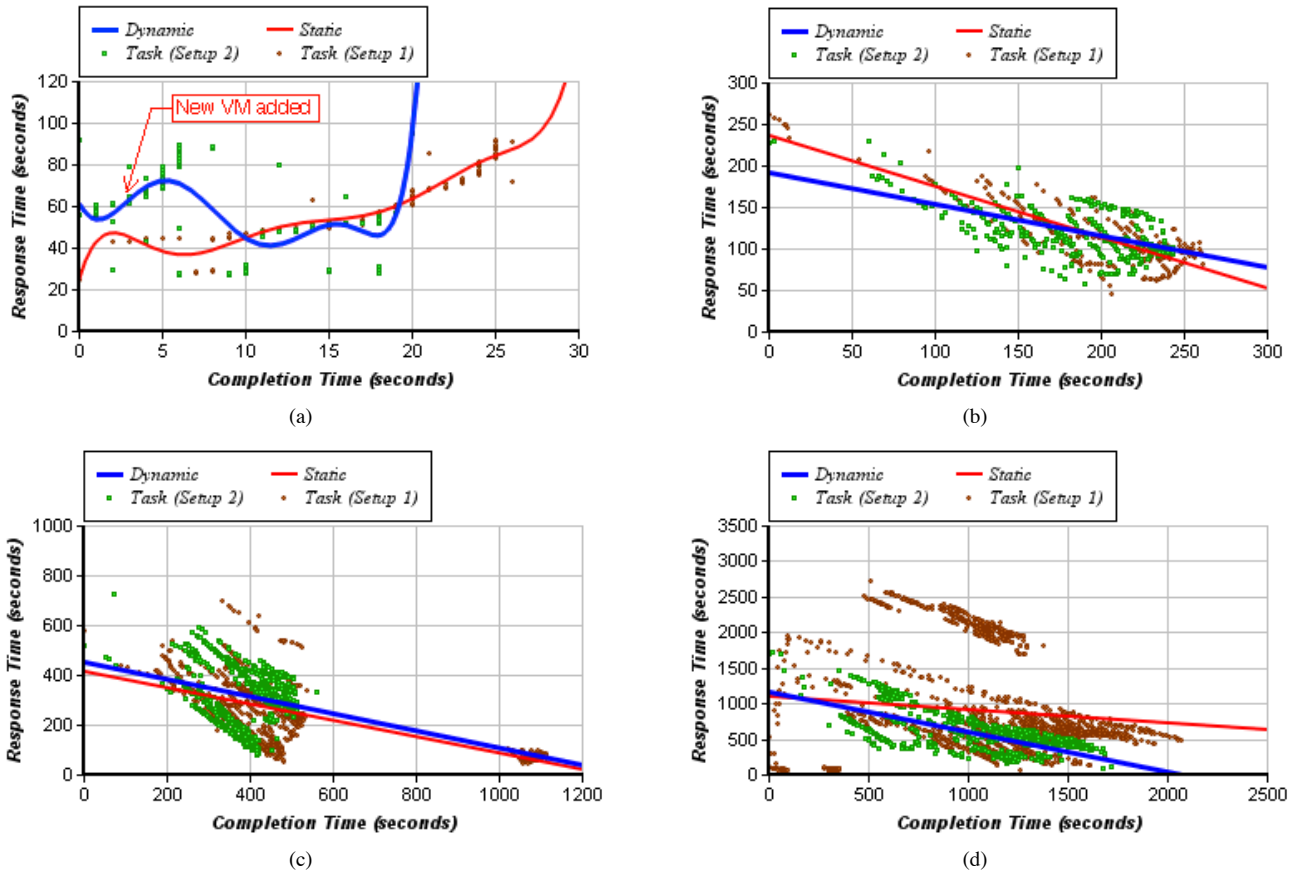

Figure 5: Response time (QoS) when using either static or dynamic resource provisioning policy. Solid lines represent general trend.

for a short period of time, when the number of requests were low ($\leq$80 in our experiment). Figure 5(a) shows the addition of a new VM when the response time increases beyond 1 minute. This addition was able to decrease the response time for some time (10 more seconds), but as the number of requests grew, the system could not distribute the load well as the dynamic provisioning policy did not have enough resources available (Setup 2). On the other hand, the static provisioning policy was able to sustain the load as there were 25 resources on standby for use (Setup 1). However, the response time increased linearly for the static case.

In Setup 3, we used a single container to handle all the 2000 tasks submitted by users. Even though the tasks were submitted gradually (not all at once), majority of the tasks got queued at the workflow engine and did not get submitted for execution. This is evident by a sharp rise in response time at the beginning for all the tasks, as depicted in Figure 6(a). The figure includes three independent executions plotted in the same graph (square, triangle, and a cross represent ECG analysis tasks) to emphasize the similar nature of the response time for repeated executions. As time progressed, tasks started to complete and the response time de-

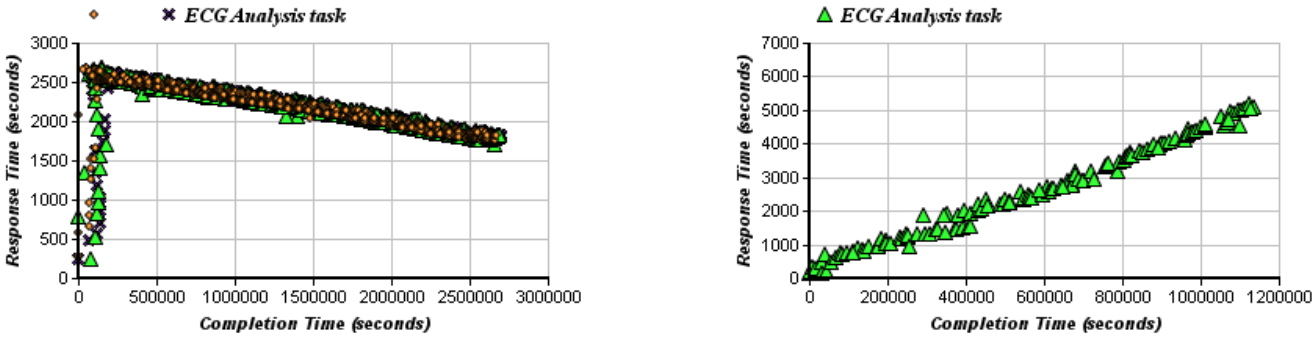

Figure 6: Response time obtained without (b) and with (a) container scaling manager.

creased from nearly 3000 seconds to ˜1800, as visible from the negative slope of the graph of Figure 6(a). This was a clear indication that the workflow engine was the bottleneck and hence prevented the scalability of the system irrespective to the number of compute resources used.

Realizing the limitation of Setup 3, we decided to dynamically scale out/in the number of Workflow Engines running so that the load (user requests) could be balanced across each of the running containers. We could achieve a gradual increase in response time as the number of user requests increased, as depicted in Figure 6(b).

6 Conclusions and Future Work

In this work, we presented an autonomic system that integrates mobile computing and Cloud Computing for analysing ECG data. We started by describing challenges end-user applications are facing when using traditional computing and service model. Our system addresses the issues related to scalability and cost in a non-disruptive manner. We demonstrate this with the help of an ECG Analysis application. We have implemented a prototype of the system and tested on volunteers.

As part of our ongoing work, we are working on heuristics that minimizes the cost of using Cloud resources maintaining user QoS satisfaction. This could be done by Cloud resource provisioning, matchmaking, and user allocations based on user priority and varying Cloud resource costs.

Acknowledgments

This work is partially supported through Australian Research Council (ARC) Discovery Project grant. Our experiments were conducted using Amazon Web Services (EC2 and S3) that kindly provided us an educational grant.

References

[1] M. Bahoura, M. Hassani, and M. Hubin. DSP Implementation of Wavelet Transform for Real Time ECG Wave Forms Detection and Heart Rate Analysis. *Computer methods and programs in biomedicine*, 52(1):35–44, 1997.

[2] G. D. Clifford. Advanced Methods & Tools For ECG Data Analysis, http://www.robots.ox.ac.uk/ gari/ecgbook.html, April Accessed - April 2010.

[3] E. Deelman, G. Singh, M. Livny, B. Berriman, and J. Good. The Cost of Doing Science on the Cloud: The Montage Example. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, pages 1–12. IEEE Press, 2008.

[4] Y. Hu, W. Tompkins, J. Urrusti, and V. Afonso. Applications of Artificial Neural Networks for ECG Signal Detection and Classification. *Journal of electrocardiology*, 26:66–73, 1993.

[5] V. Jones, A. V. Halteren, I. Widya, N. Dokovsky, R. Bults, D. Konstantas, and R. Herzog. *Mobihealth: Mobile Health Services Based on Body Area Networks*. Topics in Biomedical Engineering. Springer US, Boston, MA, 2006.

[6] B.-U. Kohler, C. Hennig, and R. Orglmeister. The Principles of Software QRS Detection. *Engineering in Medicine and Biology Magazine, IEEE*, 21(1):42 –57, jan.-feb. 2002.

[7] National Academy of Engineering. Grand Challenges for Engineering, http://www.engineeringchallenges.org, Accessed - July 2010.

[8] S. Pandey, W. Voorsluys, M. Rahman, R. Buyya, J. Dobson, and K. Chiu. A grid workflow environment for brain imaging analysis on distributed systems. *Concurrency and Computation: Practice & Experience*, 21(16):2118–2139, November 2009.

[9] PhysioNet. http://www.physionet.org/tutorials/hrv/, Accessed - April 2010.

[10] X. Qiu, J. Ekanayake, S. Beason, T. Gunarathne, G. Fox, R. Barga, and D. Gannon. Cloud Technologies for Bioinformatics Applications. In *Proceedings of the 2nd Workshop on Many-Task Computing on Grids and Supercomputers*, pages 1–10. ACM, 2009.

[11] A. Ranabahu and M. Maximilien. A Best Practice Model for Cloud Middleware Systems. In *Proceedings of the Best Practices in Cloud Computing: Designing for the Cloud, ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)*, Orlando FL, USA, 2009.

[12] J. Varia. *Cloud Computing: Principles and Paradigms*, chapter Architecting Applications for the Amazon Cloud. Wiley, New York, USA, 2010.

[13] C. Vecchiola, X. Chu, and R. Buyya. *High Speed and Large Scale Scientific Computing*, chapter Aneka: A Software Platform for .NET-based Cloud Computing, pages 267–295. IOS Press, 2009. ISBN: 978-1-60750-073-5.

[14] F. G. Yanowitz. A "Method" of ECG Interpretation, http://library.med.utah.edu/kw/ecg/ecg_outline/Lesson2/index.html, Accessed - April 2010.

Figure 7: Pictures taken during the SCALE 2010 demonstration of this work held at the CCGrid 2010 conference.