Future Generation Computer Systems 28 (2012) 147–154

Contents lists available at SciVerse ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

An autonomic cloud environment for hosting ECG data analysis services

Suraj Pandey[a,*], William Voorsluys[a], Sheng Niu[a], Ahsan Khandoker[b], Rajkumar Buyya[a]

[a] Cloud Computing and Distributed Systems Laboratory, Department of Computer Science and Software Engineering, The University of Melbourne, Australia
[b] Electrical and Electronic Engineering, The University of Melbourne, Australia

ARTICLE INFO

Article history:
Received 21 January 2011
Received in revised form
12 April 2011
Accepted 28 April 2011
Available online 15 May 2011

Keywords:
Autonomic middleware
ECG analysis
Cloud computing

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 Economical 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 [1]. 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, on 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" [1].

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.

1.1. Challenges and approach

To design a computing system architecture that efficiently supports the above mentioned objective, 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 discuss these challenges and explain 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

* Corresponding author.
E-mail addresses: spandey@csse.unimelb.edu.au (S. Pandey), williamv@csse.unimelb.edu.au (W. Voorsluys), sniu@csse.unimelb.edu.au (S. Niu), ahsank@unimelb.edu.au (A. Khandoker), raj@csse.unimelb.edu.au (R. Buyya).

0167-739X/$ – see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2011.04.022

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 computing 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 computing 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 a 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 modeling the problem as a workflow; Section 4 presents details of the implemented prototype system; Section 5 describes 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 health-care systems using similar technologies (i.e. mobile computing, body sensor networks, and high performance computing).

Jones et al. [2] propose an architecture for mobile management of chronic conditions and medical emergencies. It focuses on defining a 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. Based on this architecture the authors implemented and trialled two systems, namely: (i) the 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 [2], 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 [3]. Several algorithms have been proposed to accurately detect and classify these signals by applying various signal processing techniques, including wavelet transforms [4], neural networks [5], and genetic algorithms [4] (also refer to Kohler et al. [3] 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 computing and data intensive techniques. This fact reinforces the importance of using novel software and hardware platforms, such as our Cloud-based architecture.

In a more general context, cloud computing technologies have been evaluated and considered viable to support scientists in their computational requirements. For instance, Deelman et al. [6] carried out a study to assess the cost of doing science in the cloud by renting computing 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 [7]. Additionally, Amazon Web Services has recently announced specialized support for high-performance computing applications through its Cluster Compute Instances [8], which offer a set of virtual machines linked via a fast network interconnect.

Ranabahu et al. [9] identified the lack of scaling strategy in Cloud middleware. They propose a horizontally replicating best practice that includes a load balancing layer, an application server layer and a database layer. Their scaling strategy is horizontal replication and includes rules that trigger replications, very similar to how we replicate the workflow engine container.

The issue with data security when moving to Cloud computing is one of the primary challenges [10,11]. It becomes critical to secure data when they are related to medical history of a large number of people. Recent work focus on using encryption techniques and privacy management in Clouds. Chow et al. [11] described the use of trusted computing and applied cryptographic techniques to secure data in the Cloud. Li and Ping [12] developed a trust model for enhancing security and interoperability between Clouds. Pearson et al. [13] described several architectures for privacy management in Cloud computing. These works together with the use of best practices, as proposed by the Cloud Security Alliance (http://www.cloudsecurityalliance.org), can help secure private data in the Cloud.

3. ECG data analysis: a case study

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 analyzed by a computing system and become available to be visualized by specialists or by the patients themselves. Although our design and prototype are generic to accommodate several use

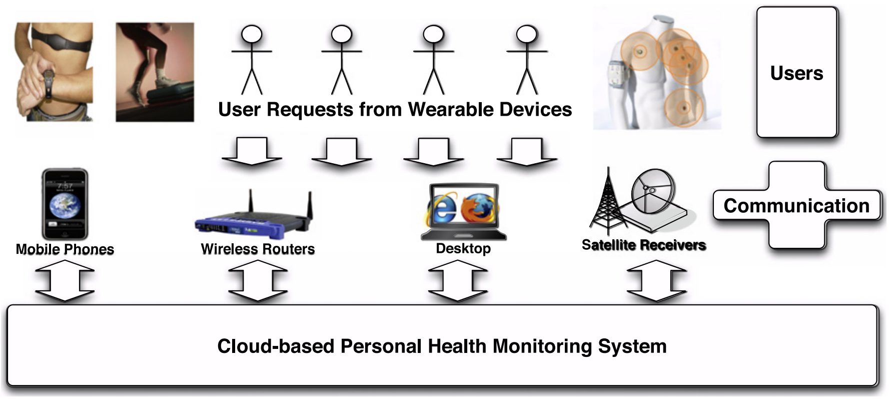

User Requests from Wearable Devices
Users
Mobile Phones
Wireless Routers
Desktop
Satellite Receivers
Communication
Cloud-based Personal Health Monitoring System

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

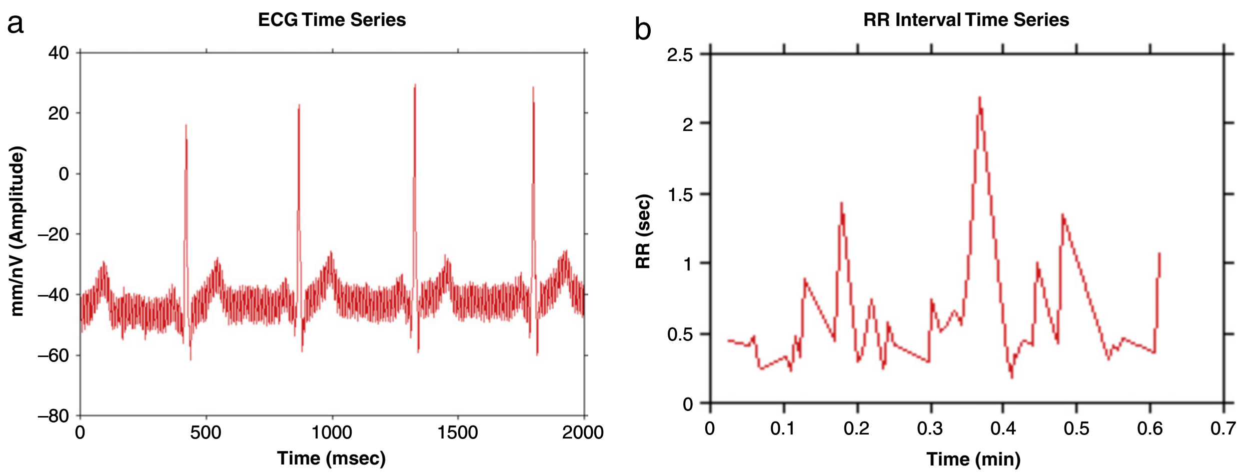


Fig. 2. Graphs depicting results obtained after the analysis of a heart beat data (obtained from a volunteer). (a) The ECG time series graph clearly showing the *P*, *Q*, *R*, *S* and *T* points. (b) The *RR* interval time series graph of the ECG graph plotted in (a).

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. Fig. 1 depicts a high level view of the use case considered in this paper.

ECG is the electrical manifestation of the contractile activity of the hearts myocardium. The *P*, *QRS* and *T* waves characterize the ECG waveform, as depicted in Figs. 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.

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 the user's preference, which may be hourly or daily over a long period of time.

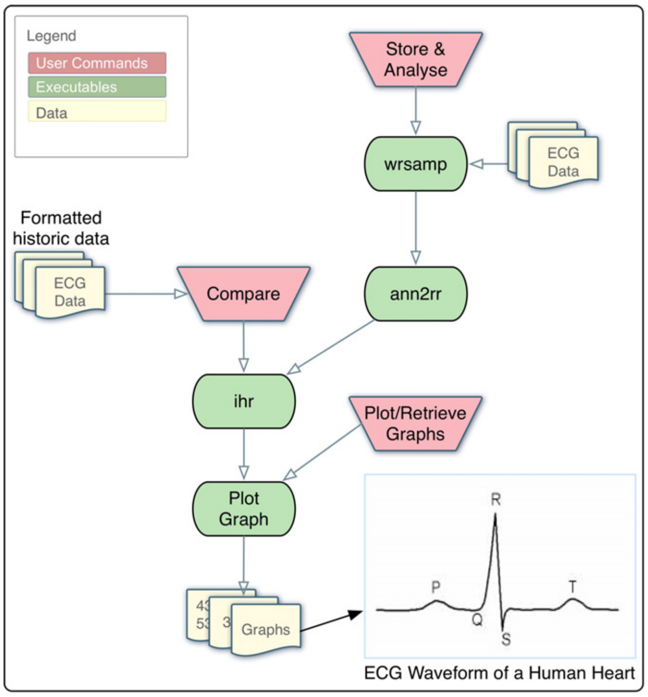


Fig. 3. A workflow depicting ECG analysis.

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.

Fig. 3 depicts a workflow for a simple analysis of ECG data. We chose this analysis process by referencing the PhysioNet tutorial [14]. This analysis process serves as a general analysis (not a clinical diagnosis) as also outlined by several authors [15,16]. An in-depth analysis of ECG data is out of scope for this paper.

The raw ECG data are the numerical readings of the signals obtained by placing electrodes at the limbs of a subject. The data is fed into the workload of computational tasks. Table 1 briefly describes the role of each of task.

The results (numerical data and graphs) obtained after the execution of the ECG application are all stored in a Cloud storage. Fig. 2 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.

Apart from the ECG analysis, Fig. 3 also depicts user directed commands, such as: store & analyze, compare, and plot/retrieve graphs. Store & analyze 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 results to the user by retrieving them 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 Fig. 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 application. The PaaS layer controls the execution of the software using three major components: (i) Container scaling manager, (ii) Workflow Engine [17], and (iii) Aneka [18].

The workflow engine [17] is hosted inside a container (e.g. Tomcat container). The engine manages the execution of tasks of the ECG application workflow depicted in Fig. 3. 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: (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 h).

The workflow engine packages the tasks that were created from user requests 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 to 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.

We propose the use of a trusted third party/certification authority to implement a Public Key Infrastructure (PKI) based authentication and authorization system. Using PKI, any data being interchanged between the application layer and the middleware can be encrypted/decrypted. Similarly, data can be encrypted using the keys from CA before storing on Cloud storage. This way, even when a malicious user, who has administrative privileges, at the IaaS layer tries to access the stored data, he would need access to the decryption keys from the CA, which he would not be able to get impersonating an existing SaaS user. Hence, assuming that the PKI system is secure, any data interchanged and stored when using Cloud-based systems can be considered secure. Work on securing data on Clouds is emerging, as also mentioned in Section 2. The prototype system and the experimental results presented in this paper does not utilize this security infrastructure. However, we would like to explore this area in greater detail in our future work.

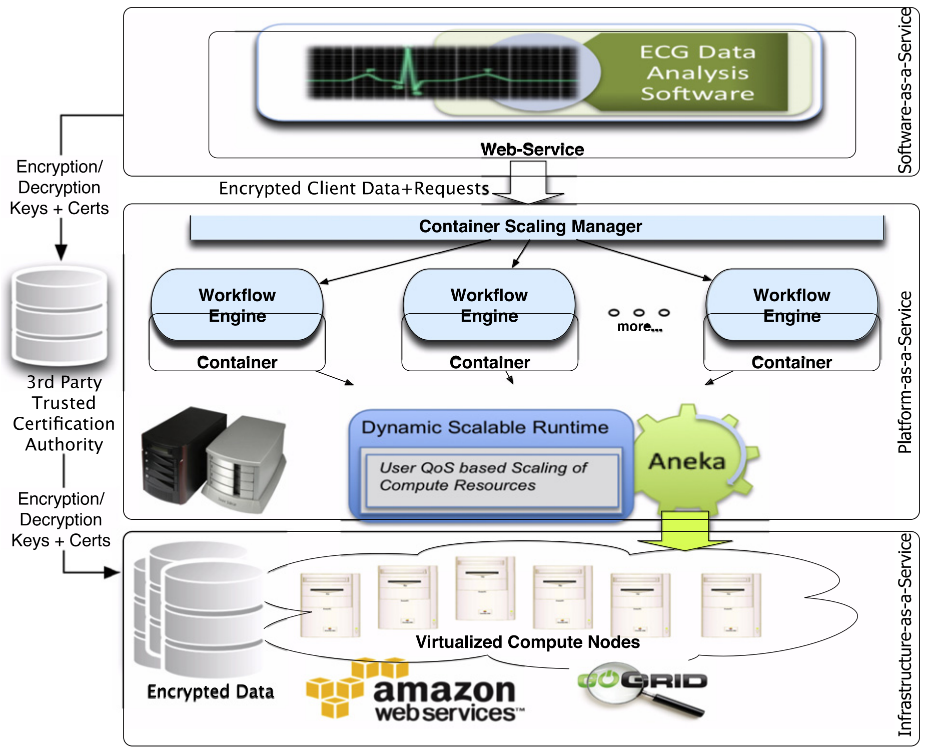

ECG Data Analysis Software
Web-Service
Software-as-a-Service
Encryption/ Decryption Keys + Certs
Encrypted Client Data+Requests
Container Scaling Manager
Workflow Engine
Container
more...
3rd Party Trusted Certification Authority
Dynamic Scalable Runtime
User QoS based Scaling of Compute Resources
Aneka
Platform-as-a-Service
Encryption/ Decryption Keys + Certs
Virtualized Compute Nodes
Encrypted Data
amazon webservices
GoGRID
Infrastructure-as-a-Service

Fig. 4. Components of the personal health monitoring system.

Table 2
Characteristics of computing 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 Computing Units	1 virtual core with 1 EC2 Computing Unit	4 Cores Intel(R) Xeon(TM) 2.80 GHz
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

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.

5.1. Experiment setup

Table 2 lists the characteristics of the computing resources we used during our experiment. We hosted the 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 computing resources (worker nodes) and a fixed number of application managers (Workflow Engine running in a fixed number of Tomcat Containers), it is *not* possible to simultaneously handle an increasingly large number of user requests and satisfy quality of service requirements (QoS = response time in our application scenario).

Proposed solution: Dynamically scale out/in the worker nodes and the application manager 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 Fig. 4:

1. If the 'average' observed user response time (e.g. in the last 1 h) is above the threshold value (1 min), instantiate more resources (worker nodes) until the response time decreases below the threshold.
2. If user requests are not forwarded to the Aneka Cloud timely (indicated both by an increase in the number of requests queued and a decrease in the utilization of the Aneka Cloud), instantiate more containers with workflow engine at the PaaS layer until Aneka resource utilization increases to more than K% (e.g. 20%) of the previous value in the same time interval.

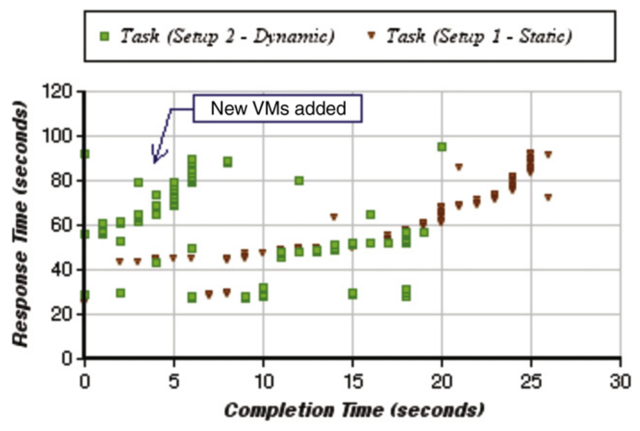
We group our experiments into the following four setups:

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 Figs. 5 and 6 by labeling the tasks (points in the graph) as *Setup 1 - Static*.

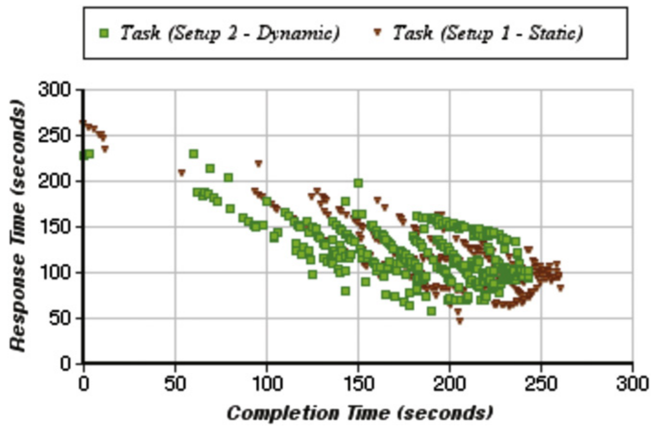
Setup 2: We use a simple dynamic resource allocation policy to instantiate Amazon EC2 resources at run-time based on the response time (up to a maximum of 50 VMs) and increase user requests from 80 to 2000. We then monitor the response time for each request. We depict the response time for this setup alongside the static setup in Fig. 5 and Fig. 6. The tasks for this setup are labeled as *Setup 2 - Dynamic*.

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 scenario is depicted in Fig. 7(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 scenario is depicted in Fig. 7(b).

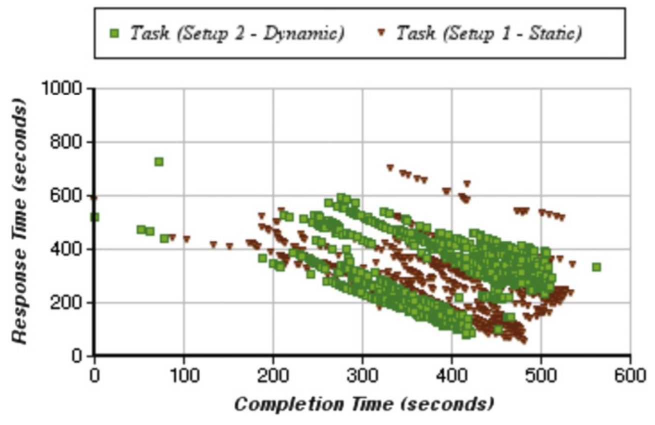


(a) Number of user requests were low (<80 requests).

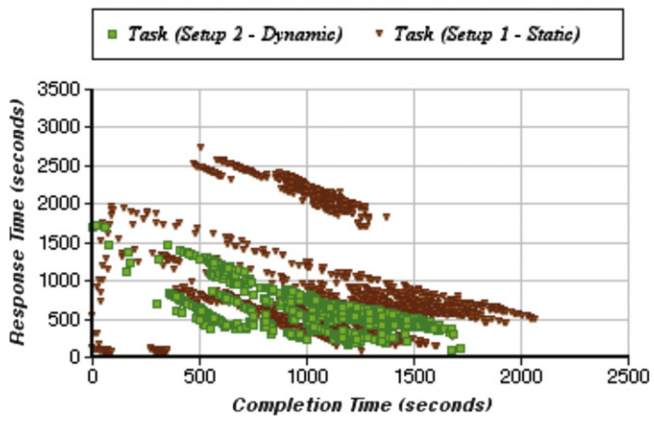


(b) Number of user requests were increased gradually up to 2000 requests.

Fig. 5. Response time (QoS) when using either static or dynamic resource provisioning policy for varying user requests. Solid lines represent the general trend.

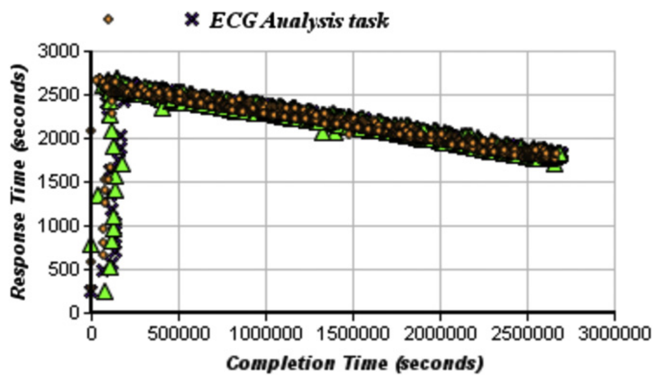


(a) Number of user requests were increased aggressively up to 2000 requests.

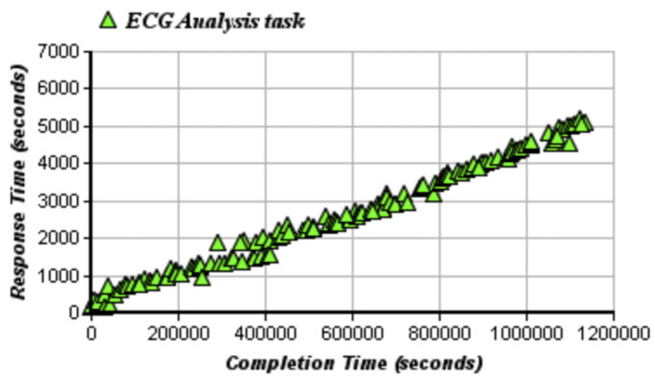


(b) Number of user requests were increased at the highest rate, up to 2000 requests.

Fig. 6. Response time (QoS) when using either static or dynamic resource provisioning policy for varying user requests. Solid lines represent the general trend.



(a) Response time when not using a container scaling manager.



(b) Response time when using a container scaling manager.

Fig. 7. Response time depends on the use of container scaling manager.

5.2. Description of results

Setup 1 represents a non-scalable and costly method for executing the ECG application. The computing resources are fixed to 25 computing 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 an increasing number of user requests. Initially, when the number of user requests were low (∼80 requests), the response time gradually increased from 30 s to 90 s, within the first 30 s of job submission (Fig. 5(a)). We then increased the number of user requests to up to 2000 gradually and monitored the response time within the first 300, 600, and 2500 s, as depicted in Fig. 5(b) and 6(a), (b).

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 (Fig. 5(a)). But, when the response time started to increase, the dynamic provisioning policy added more worker nodes to the

Table 3
Difference between minimum Response Time (minRT) and Maximum Response Time (MaxRT) when using static (25 VMs) vs. dynamic (up to 50 VMs) resource provisioning policies. (%Imprv. = Improvement).

#Tasks	25-minRT	50-minRT	%Imprv. (%)
80	25	15	40
160	40	25	37.5
320	50	50	0
640	55	50	9
2000	200	100	50

system. This runtime provisioning helped decrease the response time, as clearly indicated by the rise and fall of the response time in Fig. 5(a). Similar to Setup 1, we continued to increase the number of user requests and monitored the response time for 300, 600, and 2500 s.

Table 3 compares the response time between Setup 1 and Setup 2. In this table, 25-minRT represents the minimum response time when using 25 computing resources (Setup 1), and 50-minRT represents the minimum response time when using up to 50 computing resources (Setup 2). 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 could provision an additional 25 VMs than Setup 1 that processed user requests.

When using static provisioning, the system fails to maintain low response time when the system receives large number of requests. For example, in Fig. 5(b) and Fig. 6, as we increased the number of requests, the static provisioning policy took a longer time to complete the requests with a higher response time than the dynamic provisioning policy. In contrast, when we used the dynamic provisioning policy, the system scaled out depending on the number of requests, and the overall response time became much lower than when using static provisioning, as shown by Fig. 5(d).

But, the dynamic provisioning policy performed poorly for a short period of time, when the number of requests were low (≤ 80 in our experiment). Fig. 5(a) shows the addition of new VMs when the response time increases beyond 1 min. 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) at that point in time, as it takes time to provision addition resources. 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).

The advantage of using dynamic provisioning over static provisioning technique can also be identified from Fig. 5 and Fig. 6 when we compare the maximum response time of a majority of the tasks. Most of the tasks (excluding the outliers) that used Setup 2 (dynamic environment) produced a maximum response time lower than those tasks that were executed using Setup 1 (static environment). This improvement in system performance ultimately reduces the amount of time most users wait for obtaining the ECG analysis results.

In Setup 3, we used a single container to handle all the 2000 tasks submitted by users. Even though tasks were submitted gradually (not all at the same time), 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 Fig. 7(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 decreased from nearly 3000 s to $\tilde{1}800$, as visible from the negative slope of the graph of Fig. 7(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 computing 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 Fig. 7(b). This showed that our system could achieve both user-request parallelism (using multiple workflow containers) and process parallelism (using dynamic provisioning of computing resources).

6. Conclusions and future work

In this work, we presented an autonomic system that integrates mobile computing and Cloud Computing for analyzing 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, obtained real data from volunteers, and also demonstrated a working system in the Third IEEE International Scalable Computing Challenge (SCALE 2010), Australia, May 17–20, 2010.

As part of our ongoing work, we are working on heuristic-based techniques that minimize the cost of using Cloud resources while 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. We are also exploring techniques to address problems related to data security while using distributed Cloud storage.

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] National Academy of Engineering, Grand challenges for engineering. URL: http://www.engineeringchallenges.org (accessed July 2010).
[2] V. Jones, A.V. Halteren, I. Widya, N. Dokovsky, R. Bults, D. Konstantas, R. Herzog, Mobihealth: Mobile Health Services Based on Body Area Networks, in: Topics in Biomedical Engineering, Springer US, Boston, MA, 2006.
[3] B.-U. Kohler, C. Hennig, R. Orglmeister, The principles of software *QRS* detection, IEEE Engineering in Medicine and Biology Magazine 21 (1) (2002) 42–57.
[4] M. Bahoura, M. Hassani, 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) (1997) 35–44.
[5] Y. Hu, W. Tompkins, J. Urrusti, V. Afonso, Applications of artificial neural networks for ECG signal detection and classification, Journal of Electrocardiology 26 (1993) 66–73.
[6] E. Deelman, G. Singh, M. Livny, B. Berriman, J. Good, The cost of doing science on the cloud: the montage example, in: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing, IEEE Press, 2008, pp. 1–12.
[7] X. Qiu, J. Ekanayake, S. Beason, T. Gunarathne, G. Fox, R. Barga, D. Gannon, Cloud technologies for bioinformatics applications, in: Proceedings of the 2nd Workshop on Many-Task Computing on Grids and Supercomputers, ACM, 2009, pp. 1–10.
[8] J. Varia, Cloud Computing: Principles and Paradigms, Wiley, New York, USA, 2010, (Chapter) Architecting Applications for the Amazon Cloud.
[9] A. Ranabahu, 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.

[10] R. Buyya, S. Pandey, C. Vecchiola, Cloudbus toolkit for market-oriented cloud computing, in: CloudCom'09: Proceedings of the 1st International Conference on Cloud Computing, in: LNCS, vol. 5931, Springer, Germany, 2009, pp. 24–44.
[11] R. Chow, P. Golle, M. Jakobsson, E. Shi, J. Staddon, R. Masuoka, J. Molina, Controlling data in the cloud: outsourcing computation without outsourcing control, in: Proceedings of the 2009 ACM Workshop on Cloud Computing Security, CCSW '09, ACM, New York, NY, USA, 2009, pp. 85–90.
[12] W. Li, L. Ping, Trust model to enhance security and interoperability of cloud environment, in: Proceedings of the 1st International Conference on Cloud Computing, CloudCom'09, Springer-Verlag, Berlin, Heidelberg, 2009, pp. 69–79.
[13] S. Pearson, Y. Shen, M. Mowbray, A privacy manager for cloud computing, in: Proceedings of the 1st International Conference on Cloud Computing, CloudCom'09, Springer-Verlag, Berlin, Heidelberg, 2009, pp. 90–106.
[14] PhysioNet. URL: http://www.physionet.org/tutorials/hrv/ (accessed April 2010).
[15] G.D. Clifford, Advanced methods & tools for ECG data analysis. URL: http://www.robots.ox.ac.uk/~gari/ecgbook.html (accessed April 2010).
[16] F.G. Yanowitz, A method of ECG interpretation. http://library.med.utah.edu/kw/ecg/ecg_outline/Lesson2/index.html (accessed April 2010).
[17] S. Pandey, W. Voorsluys, M. Rahman, R. Buyya, J. Dobson, K. Chiu, A Grid workflow environment for brain imaging analysis on distributed systems, Concurrency and Computation: Practice & Experience 21 (16) (2009) 2118–2139.
[18] C. Vecchiola, X. Chu, R. Buyya, High Speed and Large Scale Scientific Computing, IOS Press, ISBN: 978-1-60750-073-5, 2009, (Chapter). Aneka: A Software Platform for.NET-Based Cloud Computing, pp. 267–295.

Suraj Pandey is a post-doctoral research fellow at the Cloud Computing and Distributed Systems (CLOUDS) Laboratory at the University of Melbourne, Australia. His Ph.D. thesis focused on scheduling data intensive application workflows on Grids and Cloud computing environments. He completed his Ph.D. from the University of Melbourne, Australia in 2010. He received his Masters degree from Inha University, Korea in 2007 and Bachelors degree from Tribhuvan University, Nepal in 2003. His research spans many areas of high performance computing, including data intensive applications, workflow scheduling, resource provisioning, and accelerating scientific applications using distributed computing paradigms. He has been participating in international software demonstrations and has been awarded on several occasions. The work presented in this article was demonstrated at the Third IEEE International Scalable Computing Challenge (SCALE 2010), Melbourne, Australia.

William Voorsluys is a Ph.D. student in the Cloud Computing and Distributed Systems Laboratory (Clouds Lab), Department of Computer Science and Software Engineering, University of Melbourne, Australia. He holds a Bachelor degree in Computer Science from the State University of Ponta Grossa, Brazil and a Masters degree in Computer Science from the University of Sao Paulo, Brazil. He has been part of the Cloudbus project since 2008, when Cloud Computing became the main focus of his research. His main works are on migration of virtual machines for data center optimization, virtual machine allocation algorithms, fault tolerance in clouds, and cost optimization in clouds with variable pricing.

Sheng Niu is a web developer at CVT Global Pty Ltd. His Masters thesis focused on scaling data intensive application via server cluster and Amazon Cloud resources. He completed his Masters degree from the University of Melbourne, Australia, in 2010 and Bachelors degree from Hunan University, China, in 2008. As part of his Masters project, he worked on web application development and scaling, and developed software that was demoed in the Third IEEE International Scalable Computing Challenge 2010.

Ahsan Khandoker brings extensive experience in Biosensors and networks, Bio-signal and image processing, and intelligent medical diagnosis through his roles as Senior Research Fellow and Research Program Manager for Australian Research Council Research Networks on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP) at the University of Melbourne. He has made significant contributions to the research field of physiological signal processing and modeling, heart rate variability in sleep disordered breathing, diabetic autonomic neuropathy, fetal cardiac disorder and human gait dysfunction, and is passionate about research helping clinicians to diagnose diseases at an early stage. He continues to lead the Sensor Networks and Information Processing in Healthcare. He has also worked with Australian Medical device manufacturing industries, hospitals focusing on integration of technology in clinical settings. Trained as an Electrical Engineering scientist, his Ph.D. examined modeling of complex physiological signals and designing medical instrumentation and was completed in 2004 with Muroran Institute of Technology in Japan and the University of North Texas.

Rajkumar Buyya is Professor of Computer Science and Software Engineering; and Director of the Cloud Computing and Distributed Systems (CLOUDS) Laboratory at the University of Melbourne, Australia. He is also serving as the founding CEO of Manjrasoft Pty Ltd., a spin-off company of the University, commercializing its innovations in Grid and Cloud Computing. He has authored and published over 300 research papers and four text books. The books on emerging topics that Dr. Buyya edited include, High Performance Cluster Computing (Prentice Hall, USA, 1999), Content Delivery Networks (Springer, Germany, 2008), Market-Oriented Grid and Utility Computing (Wiley, USA, 2009), and Cloud Computing (Wiley, USA, 2011). He is one of the highly cited authors in computer science and software engineering worldwide. Software technologies for Grid and Cloud computing developed under Dr. Buyya's leadership have gained rapid acceptance and are in use at several academic institutions and commercial enterprises in 40 countries around the world. Dr. Buyya has led the establishment and development of key community activities, including serving as foundation Chair of the IEEE Technical Committee on Scalable Computing and four IEEE conferences (CCGrid, Cluster, Grid, and e-Science). He has presented over 250 invited talks on his vision on IT Futures and advanced computing technologies at international conferences and institutions in Asia, Australia, Europe, North America, and South America. These contributions and international research leadership of Dr. Buyya are recognized through the award of "2009 IEEE Medal for Excellence in Scalable Computing" from the IEEE Computer Society, USA. Manjrasofts Aneka technology for Cloud Computing developed under his leadership has received the 2010 Asia Pacific Frost & Sullivan New Product Innovation Award.