An Auto-Scaling Mechanism for Virtual Resources to Support Mobile, Pervasive, Real-Time Healthcare Applications in Cloud Computing

Yong Woon Ahn and Albert M. K. Cheng, University of Houston
Jinsuk Baek, Winston-Salem State University
Minho Jo, Korea University
Hsiao-Hwa Chen, National Cheng Kung University

Abstract

Cloud computing with virtualization technologies has become an important trend in the information technology industry. Due to its salient features of reliability and cost effectiveness, cloud computing has changed the paradigms of development for mobile pervasive services, effectively permeating the market. While most types of best effort mobile pervasive applications can be seamlessly migrated to cloud computing infrastructures, we need to consider specialized elements to make cloud computing infrastructures more effective in real-time healthcare applications. The client side of those applications dramatically increases its transmission rate whenever it detects an abnormal event. However, the existing server side mechanisms have limitations in adaptively allocating necessary computing resources in order to handle these various data volumes over time. In this article, we propose a novel server-side auto-scaling mechanism to autonomously allocate virtual resources on an on-demand basis. The mechanism is tested in an Amazon EC2, and the results show how the proposed mechanism can efficiently scale up and down the virtual resources, depending on the volume of requested real-time tasks.

The world is moving toward a cloud computing paradigm, where mobile pervasive services will be integrated with peoples' daily lives. The development of innovative mobile pervasive services can be greatly facilitated by publicly available cloud computing infrastructures that employ virtualization technologies [1]. In order to take advantage of this facility, many organizations have started to relocate their server groups and software to cloud computing infrastructures [2]. This new trend has provided great advantages, such as reduced operation expenses and energy consumption, while achieving high utilization of computing resources.

Cloud computing infrastructures create a virtual computing environment, providing service interfaces for their users to launch applications for importing/exporting virtual machine (VM) images with a variety of operating systems (OSs). On these public infrastructures, it is common for a user's VMs to be collocated with other anonymous VMs belonging to other users on the same physical machine. Also, operations of the VMs and their associated usage of virtual resources are controlled by a shared virtual machine monitor (VMM).

Our ultimate goal is to develop a cloud-assisted mobile pervasive system with medical software as a service (SaaS) and its back-end real-time application server stacks. It should store and manage patient health records. A possible first technological evolution to this ultimate system is addressed in this article. We consider deadline-critical real-time medical data generated by sensor-based medical devices, such as wireless electrocardiogram (ECG), as an example of a need for a more streamlined computing platform. In order to handle the time-sensitive and mission-critical medical data in a public cloud computing infrastructure, a real-time application (RTA) server is required and should be operated as a VM.

However, before this goal can be successfully realized, there are issues that need to be resolved due mainly to the fact that the amount of data generated by a sensor-based medical device tends to fluctuate over time, depending on the physical condition of a patient. Generally, multiple sensors are attached to each medical device, and once a medical device detects an abnormal event, it is supposed to dramatically increase its data transmission rate to accommodate data from the sensors detecting the aforementioned abnormalities. The server side of the platform then has to launch a new virtual

Minho Jo and Hsiao-Hwa Chen are the corresponding authors for this article.

 0890-8044/13/$25.00 © 2013 IEEE

RTA server to process the increased data volume. However, this process always introduces a delay to load disk images to the new VM due to boot-up latency. While delay varies in duration depending on what OS and software are loaded from a disk image, the pending real-time tasks cannot be processed until the boot-up process is completed.

Therefore, we cannot consider a fair resource sharing mechanism available at existing cloud computing infrastructures to support our target system because such a solution evenly assigns the limited virtual hardware resources to every real-time and non-real-time VM. Of course, scaling mechanisms at a certain level are supported by some of the public cloud computing infrastructures, functioning to scale up or down the amount of virtual resources by taking into account the current data volume. Let us first acknowledge that those mechanisms are only designed to support best effort tasks, requiring a relatively conservative scaling with predefined static thresholds. In addition to this, most of these mechanisms require frequent human intervention in preparation for an emergency case.

In this article, we propose a novel auto-scaling mechanism in order to dynamically adjust the number of VMs to handle deadline-critical real-time data, which varies in size over time. In consequence, the resizing of the virtual resources for processing the given data is achieved on an on-demand basis. The key mechanism is to predict the volume of future data. Although it is not necessarily trivial to predict the exact moment at which a large volume of data will be delivered from sensor-based medical devices, most objects monitored by sensor-based devices typically show symptoms before transitioning to an abnormal state. The proposed mechanism is implemented in Amazon EC2 [3], and our evaluation results verify that it can reliably support real-time data by efficiently scaling up or down the number of VMs using the proposed prediction mechanism. We also need to mention that we achieve this effect without introducing any performance degradation in other non-real time applications. That is, we do not modify the existing virtual resource sharing modules in the VMM to support real-time applications. Instead, we implement an independent and specific session manager operating only for the RTAs.

The rest of this article is organized as follows. First, we introduce the existing auto-scaling mechanisms employed in public cloud computing infrastructures. We then explain our system model for the client and server sides, respectively. Next, e propose an auto-scaling mechanism that takes into account the size of future data volume sent by sensor-based medical devices. We conduct a performance evaluation of the proposed mechanism, followed by the conclusion of this article.

Related Work

In many public cloud computing infrastructures, available virtual hardware resources are fairly shared by all VMs in a physical machine. This fairness fails to support RTAs requiring differentiated levels of available virtual resources from other non-real-time applications. Although some certain infrastructure [3] provides a mechanism to statically increase or decrease virtual resources for each VM, the allowable scaling period (typically several minutes) is too optimistic to support RTAs. Even just a few minutes can be too risky in a period for sensor-based RTAs such as remote structural or patient monitoring systems.

In order to efficiently support the RTAs with currently available hardware resources, a mechanism predicting a future data volume is essential. The workload prediction model was recently introduced in [4] for cloud services. The prediction model was designed for best effort data generated by human-controlled behaviors without considering processing timeliness constraints. Therefore, its simplicity introduces a limitation to be applied to sensor-based RTAs. This more intuitive approach works well with best effort web services. Other research [5] proposed a virtual resource scaling mechanism that considers both timeliness and resource constraints. However, the aforementioned timeliness is not adaptively determined based on the dynamic transmission rate generated by sensor-based medical devices. Although the approach allows the deadline to be changed depending on the observed data volume, the adjustment is still manually controlled by a human system administrator who has to modify a configuration file.

An autonomous computing system without human intervention was considered in [6, 7]. The system defined multiple strategic steps to develop an autonomous system, and showed how to apply the proposed development steps to implement a Java EE application server running in a cloud computing infrastructure. Unfortunately, the approach is more appropriate for designing best effort applications in cloud computing infrastructures, supported with limited computing capacity. Another autonomous management solution was proposed in [8]. In order to reduce the energy consumption of battery powered user devices, this approach detects and localizes thermal hotspots in cloud data centers. Obviously, this approach has completely different goals and methods from ours. However, due to its real-time sensing concept, it provides meaningful clues to help solve our problems.

In summary, to the best of our knowledge, no mechanism has yet been designed to support sensor-based RTAs that also processes deadline-critical real-time data generated by medical devices. More important, all aforementioned approaches neglect to consider the booting-up delay that occurs when launching a new VM. Likewise, the cooling-down mechanism, which configures the proper moment to decrease the number of running VMs, is missing. The aforementioned limitations of the existing approaches are taken into consideration in our proposed system without disturbing normal operations of other VMs in the same physical machine.

System Model

Client Side

Let us consider a sensor-based medical device, such as an ECG equipped with a local controller and multiple sensors as a client entity. A local controller periodically collects the sampled analog signals from its sensors, performs an analog-to-digital conversion, and compresses the digitized signals. The sampling rate is dynamically managed to determine the data transmission rate. For example, when the attached sensors detect an abnormal event, the local controller transmits the collected sampled data to its local outgoing queue at an increased transmission rate with a predetermined and specified deadline. The transmission deadline represents the maximum allowed time until the sampled data should be transferred to the outgoing queue of the medical device. As such, this deadline should be determined by comprehensively taking into account various delay factors, including time overhead for digitization and compression. Usually, a transmission deadline of data in each sampling period is set to the starting time of the following sampling period. If no abnormal event is detected, the sampled data does not need to be immediately transmitted for emergency treatment. In such a case, reliable transmission is more important than fast transmission. Once the sampled data is transferred to the outgoing queue, the data are mapped into one of the appropriated sub-queues

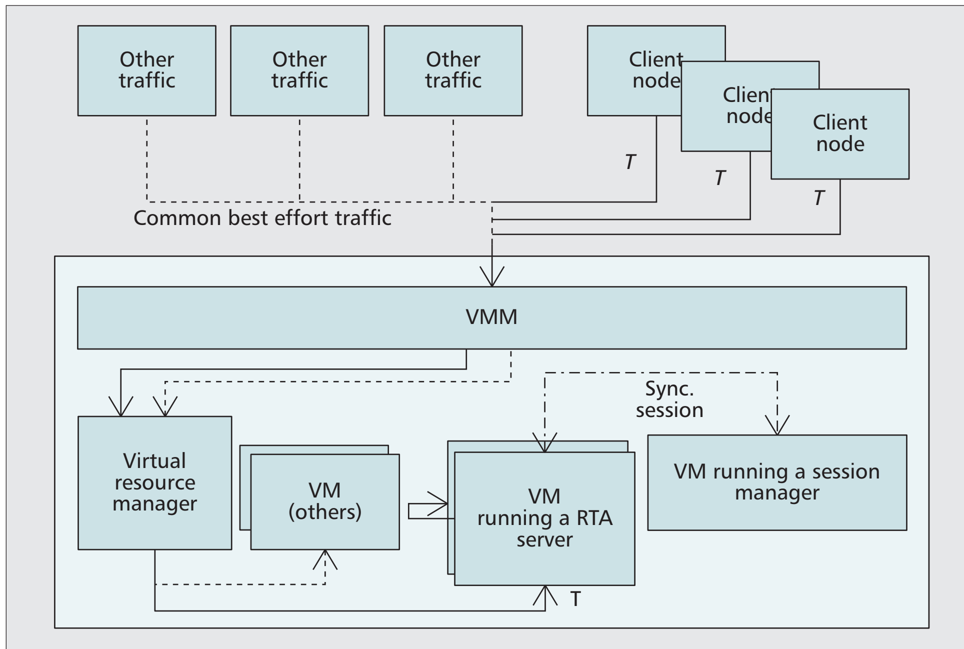

Figure 1. *Physical machine architecture with virtual RTA servers as guest domains.*

having a specific processing deadline d_n, where n is the index of the sub-queue. This deadline-based mapping is also managed by the local controller. The controller now forms a group of consecutive real-time tasks T having the same processing deadline d. When s different sensors are involved in data transmission, variable T has four properties:

- The deadline for a group of tasks
- A task type
- Network address of the device
- The total amount of virtual or physical resources required to finish T within the specific group deadline

The server side will reference the task type to figure out virtual resource requirements for task T. For example, if the task type is set to be deadline-critical and CPU-intensive, the RTA server will reserve more virtual CPU (VCPU) resources for the task. Otherwise, if the task type is set to be mission-critical and I/O intensive, the server only needs to pass task T to virtual I/O devices. More detailed procedures to convert the sampled raw signals to ordinary real-time tasks were discussed in [9]. Hereafter, the sensor-based medical device is referred to as a client node.

Server Side

On the server side, the RTA server parses sampled data sent by client nodes. After that, it extracts, processes, and stores them to a shared data repository. Also, it transmits the processed data to remote client nodes if necessary.

Figure 1 shows that the RTA server installed in multiple VMs possibly coexists with multiple other independent general-purpose VMs. Therefore, each VM is isolated and protected from external malfunctions. A shared VMM cooperates with a virtual resource manager to control all VMs for resource allocation purposes.

With the given architecture, our inclination is to provide a non-stop service by allowing the multiple RTA servers to share common client sessions. Without this consideration, each client node has to attach its detailed session information to every packet header, causing unnecessary network bandwidth consumption.

As such, we design an independent session manager located in another VM. The session manager controls and synchronizes the sessions of all real-time client nodes connected to the RTA servers. This eventually allows the sampled data sent by the same client node to be processed by different RTA servers. To addressing fault tolerance and scalability issues, a session manager can be duplicated to multiple VMs.

We utilize one designated root RTA server to process the sampled data. More RTA servers will be launched and defined as child RTA servers upon receiving the request to launch more RTA servers. The root RTA server has an incoming and outgoing queue to buffer the requested tasks by its client nodes. If the root RTA server does not have enough computing resources to finish all of the real-time tasks within their specified deadlines, it assigns those tasks to its child RTA server.

In order to check the available computing resources against a given real-time task T_i, it calculates projected system response time R_i for the real-time task T_i and compares it with a given absolute processing deadline d_i. For the calculation, we consider:

- Expected processing time for task T_i
- Waiting time to de-queue task T_i from an incoming queue
- Waiting time to de-queue task T_i from an outgoing queue

The expected processing time again consists of:

- Time for scheduling task T_i in a root RTA server
- Time for computation for task T_i
- Time for completing I/O operations for task T_i

To meet the processing deadline, the projected response time R_i should be shorter than absolute time difference between two consecutive absolute processing deadlines d_i and d_{i+1}.

Auto-Scaling Mechanism

Due largely to our hierarchical structure among the RTA servers, our system does not need to be governed by the centralized conventional auto-scaling mechanism provided by the VMM. Instead, the root RTA server acts as an auto-scaling controller to launch a new child RTA server or terminate an existing child RTA server. In order to design the auto-scaling mechanism, we partially adopt four iterative, stepwise, and functional concepts of autonomous computing proposed in [10], as follows:

- Monitor: It collects, filters, and reports condition of the managed resources.

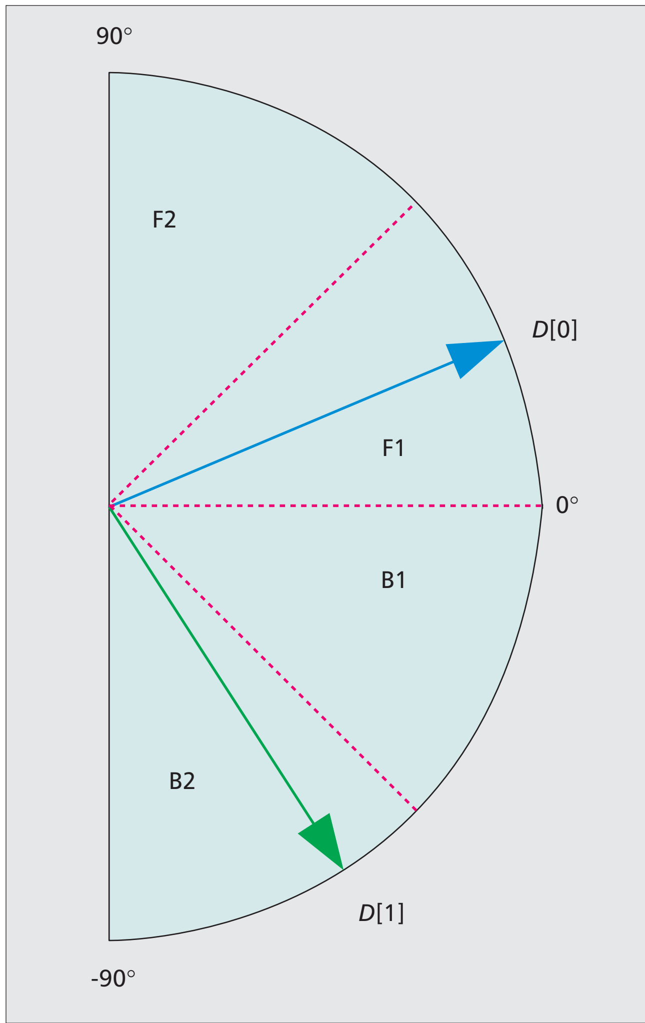

Figure 2. *State diagram for transitions of the proposed auto-scaling mechanism.*

- Analyzer: It analyzes collected data of managed resources and predicts future states based on these data.
- Planner: It generates an appropriate plan to achieve technical goals.
- Executer: It controls managed resources based on a recommended plan received from the planner.

Monitor

The simplest way to monitor resource usage of multiple RTA servers can be achieved by straightforwardly adopting a default resource monitor [11] provided by a public cloud infrastructure. However, the predefined optimistic monitoring interval does not work well with our RTA servers. This is because in our system each RTA server may need to have different monitoring intervals and performance metrics, which should be dynamically adjusted for better system-wise performance. Therefore, we implement an independent real-time resource monitor as a sub-component on a guest OS running in the root RTA server.

The developed monitoring module initially collects system parameters, such as associated private and public IP addresses, associated instance IDs, initial monitoring interval, and performance metrics. Within the monitoring system, we launch various monitoring software such as the one reported in [12]. The monitor can now capture and parse the on-screen results from the software. The results are stored in a shared knowledge repository for other functional components. Each component has peer-to-peer communication modules to request and respond to virtual resources usage and queue states. Another important role of the monitoring system is to monitor the child RTAs in the same way as the other managed resources. The monitored results will be used by the analyzer.

Analyzer

The analyzer predicts future states of the RTA servers based on the collected performance metrics. It decides whether the root RTA server will take on a new real-time task after checking available resources against the deadline of the task. If the root RTA server has insufficient resource capacity to run the task, it shifts the task to the child RTA that has the smallest number of buffered tasks in its incoming queue. If there is no available child RTA server, a root RTA server launches a new VM to run an additional child RTA server. It introduces boot-up delay D, which is the required time to launch a new VM with a guest OS image.

In order to assign the task to a new RTA, the projected response time requirement for the new RTA server should be revised to include the boot-up delay. That is, the calculated and projected system response time R_i should be even shorter than the time difference between two consecutive absolute deadlines, d_i and d_{i+1}, plus boot-up delay D. This deadline checking procedure is required to automatically assign real-time tasks to available RTA servers. A new child RTA server should be launched only when absolutely necessary. However, if D is too long to satisfy the deadline requirement, a proper prediction mechanism is required. Our prediction is performed with a moving average filter (MAF) module. Let us suppose that Avg_i is the ith moving average value of VCPU usage, CU_i is the amount of the ith VCPU usage, and k is the number of observed intervals. When we calculate the value of S_{avg} by subtracting Avg_{i-1} from Avg_i, S_{avg} becomes the current slope of Avg_i, which will then be used to predict Avg_{i+1}. Accordingly, if S_{avg} is larger than zero, the root RTA server determines that the tasks may require more computing resources in the next interval. Otherwise, it requires less computing resources.

The existence of an independent resource monitor with an adjustable monitoring interval allows us to reference real-time VCPU usage records to predict the system states for the next interval more accurately. However, increasing or decreasing the number of VMs only depending on the observed S_{avg} would introduce suboptimal resource utilization, because the value S_{avg} is likely to be oscillated drastically even within a very short time interval. Therefore, the system should define multiple logical states to prevent frequent but unnecessary variations of the number of VMs. Let N be the total number of states that each RTA server has, and $D[i]$ is the degree representing a current state i of a RTA server, which is ranged from –90° to 90°. The $D[i]$ value can be calculated by converting S_{avg} to the angular value of each timing point.

Our system initially indicates that the root RTA server is in "State 1" in normal operational mode. In order to provide more accurate predictions, the number of transitions should vary depending on the value of $D[i]$. Let us assume that there are M sections between –90° and 90°. If the value of $D[i]$ is equal to or greater than w90°/M/2, but smaller than $(w+1)90°/M/2$, and $D[i]$ is bigger than 0°, the analyzer moves the state w transitions forward. If the value of $D[i]$ is equal to or smaller than $w(-90°)/M/2$, but still smaller than $(w+1)(-90°)/M/2$, and $D[i]$ is smaller than 0°, the analyzer moves the state w transitions backward. Figure 2 shows an example, when there are four different sections, including F1, F2, B1, and B2.

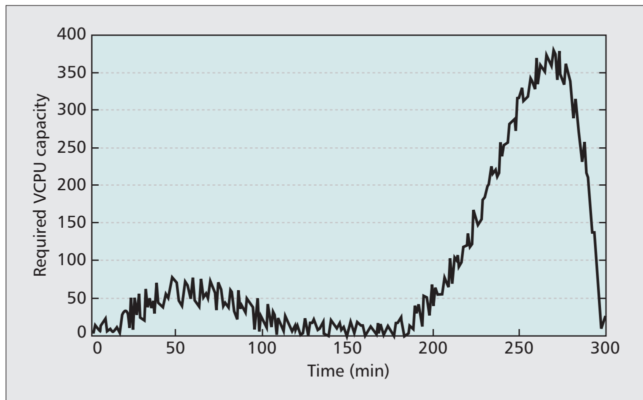

Figure 3. *VCPU workload used for evaluation.*

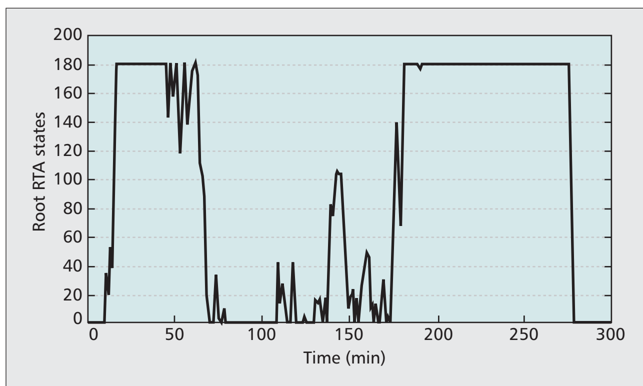

Figure 4. *States of root RTA server over time with a given VCPU workload.*

If the root RTA server reaches State *N*, it determines whether it needs more computing resources or not by checking the states of other child RTA servers. If it is needed, it launches a new child RTA server, where the overloaded pending real-time tasks will be assigned. On the other hand, if *D* is smaller than 0°, it now makes *w* backward transitions. When it reaches at State 1, it terminates one of its child VMs and intercepts the workload of the terminating child RTA. Note that the actual launching and termination of child RTAs will be performed in executer.

In order to figure out appropriate parameters such as *N*, we need to run a certain number of iterations with four autonomous computing concepts. The parameter values can eventually be obtained by repeatedly referencing a knowledge repository, where the previous parameter values and history of missing deadlines are stored. The parameter values obtained at the current iteration are passed to the planner.

Planner

The planner makes a plan for the next iteration based on the forwarded parameter values from the analyzer. If the analyzer reports that a root RTA server reaches State *N*, the planner sends a launch command to the executer. In addition to this, the planner will make a plan to assign overloaded pending tasks to the newly launched child RTA server. If the analyzer indicates that a VM reaches State 1, the planner needs to make a plan to terminate the child VM running the fewest tasks. The running tasks at the child RTA that are supposed to be terminated will shift to the root RTA server or another child RTA server. In such a case, the system notifies these activities for the client node. We require that the root RTA server take over most of these shifted tasks as long as the resource is available because this mechanism allows the child RTA servers to have smaller workloads and be terminated sooner. In our system, this module is implemented using AWS Java SDK [13].

Executer

This module executes a plan recommended by the planner. If the root RTA needs to launch a new child RTA, it is executed by a remote procedure call. In the case of child RTA termination, the executer must complete a similar procedure by sending the terminating message to the cloud. In order to use these remote procedure calls, the executer must collect the instance ID of the child RTA along with its private and public IP address.

Performance

We set up our experiment environment in Amazon EC2 that provides 1.7 Gbyte memory space and moderate I/O performance. To observe the operations of the proposed auto-scaling mechanism, we turned off the default resource manager provided by the VMM. Instead of actual workload generated by the client nodes, we used a virtual workload indicating required VCPU capacities to process all real-time tasks within the specified deadlines. It allows us to eliminate any possible impact of the shared VMM for our evaluation. It is necessary to provide reliable and generic evaluation results, which can later be applied to other publicly available cloud computing infrastructures equipped with a general-purpose VMM. The proposed mechanism was implemented in Java 1.6 and included in the Fedora 16 image [14]. It starts automatically as a daemon process while booting up the image.

Figure 3 shows VCPU workload requirements for 300 min to process the requested real-time tasks. As we reflect on cases of erroneous environments such as packet losses, the curves show a faintly audible level of noise, which makes it difficult to predict the workload shouldered by the next interval. The maximum VCPU capacity of each RTA is set to 50. Therefore, the first existing root RTA experiences two pending groups of real-time tasks at 28 min and 196 min, respectively. The workload also shows two peaks. The first peak occurs at 55 min, and approximately 68 units of the VCPU capacity are required to process the given real-time tasks. Therefore, the root RTA has to launch a new child RTA before dropping the real-time tasks. The boot-up delay for the 64-bit Fedora 16 image is usually about 1 min.

Figure 4 shows values $D[i]$ with the VCPU workload. We set the interval value k to 10, and N to 90°. As we can see, the root RTA reaches the final state *N* at 36 and 198 min, and is required to launch a new child RTA. Since these results were evaluated based on the prediction mechanism, we can ensure that the mechanism allows a new RTA to be launched before the real-time tasks are overloaded. We also can simultaneously launch multiple child RTAs by analyzing the amount of incoming tasks in the queue.

When the analyzer detects that the root RTA reaches State *N*, it calculates the number of RTAs still to be booted up. Figure 5 shows the number of running child RTAs with the proposed auto-scaling mechanism. We can see that the root RTA launches the first running child RTA at about 20 min. On the other hand, the root RTA terminates all child RTAs at about 93 min in order to scale down. As a result, the root RTA does not need to have a child RTA until the 197-min mark.

However, at 198 min, two child RTAs become available. It is crucial to note that these two child RTAs have already been launched at the 181-min mark, showing an achievement the success of which is due to our prediction mechanism.

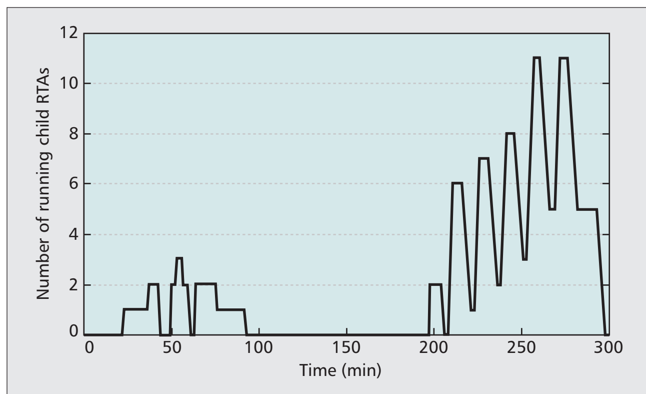

Figure 5. *The number of child RTA servers over time.*

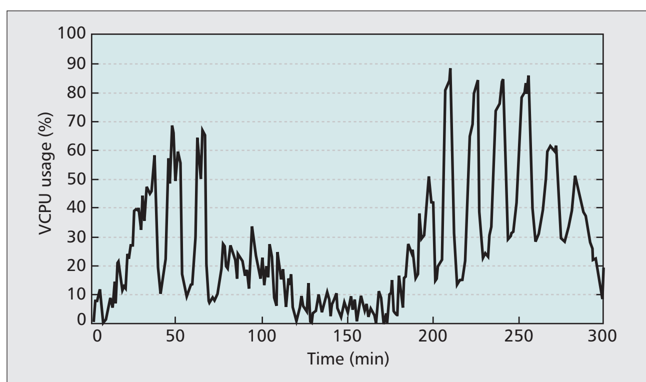

Figure 6. *VCPU usage of the root RTA.*

Since the workload includes unstable and nonlinear values of required VCPU capacities, it is possible that the VM pool has more child RTAs than it actually needs. However, this is still acceptable to run RTA servers in the public cloud infrastructures, due to the fact that the VM pool maintains only a small number of VMs when the objects sensed by the client node are in a normal condition. Therefore, meeting the specific deadlines is much more important than saving the computing resources for RTA servers, especially when time-critical real-time tasks are requested by the client nodes.

Figure 6 shows VCPU usage of the root RTA with the proposed mechanism. After launching its child RTAs, the VCPU usage of the root RTA immediately drops down to around 10 percent. As described in the previous section, we allow the root RTA to process as many tasks as it can. In most cases, the child RTAs complete a relatively small amount of tasks when compared to their root RTA. Also, the child RTAs are supposed to be terminated as soon as their incoming queues are empty. This can maximize VCPU utilization of the system and minimize the number of child RTAs. Accordingly, at around 40 min, our auto-scaling mechanism determines that the root RTA can process more tasks and tries to terminate its child RTA.

As shown in Fig. 5, there are two running child RTAs at that moment. As a result of these terminations, the VCPU usage of the root RTA increases over 50 percent. Immediately afterward, the system detects that the tasks are overloaded once again. The root RTA unfortunately needs to launch child RTAs again at the 51-min mark. However, from around 100 to 190 min, the workload shows stabilization due to the normal condition of the detected object. Our system accurately recognizes this condition and minimizes the number of running RTAs during this period.

Conclusion

We investigate limitations of the existing scaling mechanisms implemented in publicly available cloud computing infrastructures. In order to overcome the limitations, we propose a novel auto-scaling mechanism supported by sessions used to support RTAs. The reliability and efficiency of the proposed mechanism come from cooperating with an independent real-time resource monitor, a virtual session manager, and a workload prediction algorithm. The evaluation was performed with workload in terms of VCPU usage in Amazon EC2 with Fedora 16 image. The results verify that the proposed mechanism can efficiently scale the number of RTA servers up and down by considering the available computing resources against the given workload. In the future, we will define new parameter groups to consider and categorize subject (or patient) groups to differentiate our state transition mechanism. For example, if a group's severity is higher than others, the RTA server's state would be moved to another state relatively faster with differentiated parameters, which can be determined by physicians or medical professionals.

References

[1] S. Ahn *et al.*, "Isolation Schemes of Virtual Network Platform for Cloud Computing," *KSII Trans. Internet and Info. Sys.*, vol. 6, no. 11, Nov. 2012, pp. 2764–83.

[2] W. Hui, C. Lin, and Y. Yang, "MediaCloud: A New Paradigm of Multimedia Computing," *KSII Trans. Internet and Info. Sys.*, vol. 6, no. 5, May 2012, pp. 1153–70.

[3] Amazon Elastic Compute Cloud, http://aws.amazon.com/ec2/, last retrieved June 2013.

[4] N. Roy, A. Dubey, and A. Gokhale, "Efficient Autoscaling in the Cloud Using Predictive Models for Workload Forecasting," *Proc. 2011 IEEE Int'l. Conf. Cloud Computing*, July 2011, pp. 500–07.

[5] M. Mao, J. Li, and M. Humphrey, "Cloud Auto-Scaling with Deadline and Budget Constraints," *Proc. 2010 IEEE/ACM Int'l. Conf. Grid Computing*, Oct. 2010, pp. 41–48.

[6] B. Solomon *et al.*, "Decentralized Predictive Control of Autonomic Computing Environments," *Proc. 2006 Int'l. Info. and Telecommun. Technologies Symp.*, Dec. 2006, pp. 94–103.

[7] B. Solomon *et al.*, "A Real-Time Adaptive Control of Autonomic Computing Environments," *Proc. 2007 Centre for Advanced Studies Conf.*, Oct. 2007, pp. 1–6.

[8] H. Viswanathan, E. K. Lee, and D. Pompili, "Self-Organizing Sensing Infrastructure for Autonomic Management of Green Datacenters," *IEEE Network*, vol. 25, no. 4, Aug. 2011, pp. 34–40.

[9] Y. W. Ahn *et al.*, "Improving QoS for ECG Data Transmission with Enhanced Admission Control in EDCA-Based WLANs," *Proc. IEEE GLOBECOM*, Dec. 2011, pp. 1–5.

[10] "SMART (Self Managing and Resource Tuning)," IBM Research, 2003.

[11] Xen, http://www.xen.org/products/, last retrieved in June 2013.

[12] Mpstat, http://www.linuxcommand.org/man pages/mpstat1.html, last retrieved June 2013.

[13] AWS Java SDK, http://aws.amazon.com/sdkforjava/, last retrieved June 2013.

[14] Linux, Fedora 16, http://fedoraproject.org/, last retrieved June 2013.

Biographies

YONG WOON AHN (yahn@cs.uh.edu) received B.S. and M.S. degrees in computer science and engineering from the Hankuk University of Foreign Studies, Korea, in 2001 and 2003, respectively. He is currently pursuing a Ph.D. degree in computer science with the Department of Computer Science, University of Houston, Texas. His current research interests include cloud computing, real-time systems, fault-tolerant computing, ubiquitous computing with embedded devices, and middleware for scalable network environments.

ALBERT MO KIM CHENG (cheng@cs.uh.edu) received B.A., M.S., and Ph.D. degrees, all in computer science, from the University of Texas, Austin. He is a full professor and former interim associate chair of the Department of Computer Science at the University of Houston, where he is also the founding director of the Real-Time Systems Laboratory. The author of the popular textbook *Real-Time Systems* (Wiley), he has published over 180 refereed publications in leading venues in the area of power and reliability-aware real-time, embedded, and cyber-physical systems.

JINSUK BAEK (baekj@wssu.edu) received B.S. and M.S. degrees in computer science and engineering from the Hankuk University of Foreign Studies in 1996 and 1998, respectively, and a Ph.D. degree in computer science from the University of Houston in 2004. He is currently an associate professor of with the Department of Computer Science, Winston-Salem State University, North Carolina. His current research interests include multimedia communications, scalable reliable multicast protocols, mobile wireless communications, and network security.

MINHO JO [M'07] (minhojo@korea.ac.kr) received his Ph.D. from the Department of Industrial and Systems Engineering, Lehigh University, in 1994. He is a professor with the College of Information and Communication at Korea University, Seoul. He is the founder and Editor in-Chief of *KSII Transactions on Internet and Information Systems*. He is an Editor of *IEEE Network* and *IEEE Wireless Communications*, respectively He has published many refereed academic publications in very high-quality journals and magazines. Areas of his current interest include cognitive radio, network algorithms, optimization and probability in networks, network security, wireless communications, energy efficient wireless communications, WBAN, and cloud computing.

HSIAO-HWA CHEN [S'89, M'91, SM'00, F'10] (hshwchen@mail.ncku.edu.tw) is currently a Distinguished Professor in the Department of Engineering Science, National Cheng Kung University, Taiwan. He obtained his B.Sc. and M.Sc. degrees from Zhejiang University, China, and a Ph.D. degree from the University of Oulu, Finland, in 1982, 1985, and 1991, respectively. He is the founding Editor-in-Chief of Wiley's *Security and Communication Networks Journal* (www.interscience.wiley.com/journal/security). He was the recipient of the Best Paper award at IEEE WCNC 2008 and the IEEE Radio Communications Committee Outstanding Service Award in 2008. Currently, he is also serving as Editor-in-Chief of *IEEE Wireless Communications*. He is a Fellow of IET and a Fellow of BCS.