Automated Demand-Based Vertical Elasticity for Heterogeneous Real-Time Workloads

Geoffrey Phi C. Tran*†, Yu-An Chen*†, Dong-In Kang*, John Paul Walters*, and Stephen P. Crago*†

*Information Sciences Institute
University of Southern California, Arlington, VA 22203
Email: {gtran, dkang, jwalters, crago}@isi.edu

†Department of Electrical Engineering
University of Southern California, Los Angeles, CA 90089
Email: {geoffret, chen116}@usc.edu

***Abstract*—Cloud computing is revolutionizing the information technology field. However, clouds today have not yet addressed real-time applications. Current work has shown that real-time hypervisors are capable of allowing virtual machines to meet real-time requirements. Other work has looked at statically allocating resources to virtual guests, which allow those guests to meet deadlines. In this paper, we present DART-C (Demand-based Allocation for Real-Time Clouds), a dynamic real-time cloud framework that allows automated adaptation to these types of workloads. Many applications follow dynamic multi-modal execution patterns, with varying computational requirements over time. DART-C provides demand-based elasticity to support changing real-time requirements by enabling applications to report mode changes to a resource manager, which reallocates resources based on need. We also describe a prototype and demonstrate up to 62% in system resource utilization savings compared to a static allocation, when running a synthetic application set.**

***Index Terms*—Virtualization; KVM; Xen; Real-Time; Schedulability; Cloud; Dynamic; OpenStack**

I. Introduction

Cloud computing has revolutionized modern IT by enabling seemingly limitless data storage, pay-as-you-go computing, and big data processing. In many domains, it is now more cost-effective to purchase computing time from a cloud provider rather than building and maintaining in-house solutions. While current cloud infrastructures support general-purpose computing, support for real-time applications is nascent [1]. While steps have been taken toward real-time clouds, to date all real-time clouds are limited to static workloads, which are not representative of many common real-time applications. Dynamic real-time workloads have not been addressed within the cloud context. Addressing those workloads is the focus of this work.

An example of a cloud-based real-time application where dynamic behavior could be present is in an online securities broker. The processing and execution of orders placed by customers should be completed as quickly as possible to meet customers' expectations. If orders are completed with excessive latency, that may result in unfavorable results for the customer and broker (e.g. financial loss). Therefore, a broker may set deadlines for each order to be processed. This defines a hard real-time constraint on the execution time to process an order, which means that the deadline should be met, or the computation's value to the broker and users may be zero. When more orders are expected, such as if a large influx of users logging in is detected, the broker may wish to scale resources to meet those deadlines.

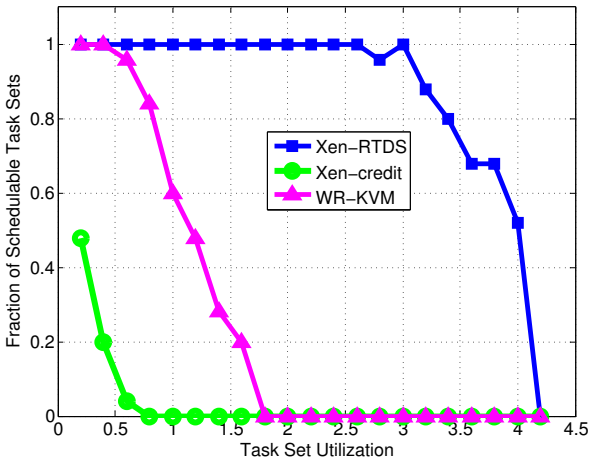

Fig. 1: Comparison of Hypervisor Real-Time Performance

In order to construct a real-time cloud, current hypervisors and schedulers need to be evaluated to see which enable the best real-time performance. In Figure 1, we compare the Xen Real-Time Deferrable Server (RTDS) and credit schedulers, in addition to the WindRiver Low-Latency KVM hypervisor. Each system runs 25 task sets where each task in the task set has a light utilization uniformly distributed between (0.001, 0.1) and a period uniformly distributed between (10ms,100ms). We compare the fraction of schedulable task sets versus total task set utilization for a task set with low individual task utilization. A task set is schedulable if no deadlines are missed during the execution. Here it can be seen that deadlines for given task sets can indeed be met for higher utilizations when using a real-time hypervisor, compared to the general-purpose hypervisors. We utilize RT-Xen (the project that developed the RTDS scheduler) because it demonstrated the higher real-time performance.

Our work is motivated by the realization that many real-time workloads exhibit dynamic multi-modal execution patterns. Current work has been limited to static allocation, which is

not suitable for dynamic workloads [2]. In traditional cloud computing (general-purpose), this increasing and decreasing workload is addressed through the use of horizontal elasticity, or adding cloud instances. The other dimension is vertical elasticity, or increasing the computational resources of currently running instances, such as re-sizing a 2-core VM into a 4-core VM. Vertical elasticity is currently available only through coarse grained instance re-sizing.

We believe that automated demand-based vertical elasticity is critical for efficient use of resources in real-time clouds. This is because it allows for real-time requirements to be met, regardless of the changing execution patterns, while also freeing up resources when in a lower utilization mode. Much like the horizontal case, general-purpose applications benefit by utilizing the freed resources. However, real-time applications may not be able to take advantage of the increased parallelism across multiple VMs, but are able to take advantage of the increased resources provided by vertical scaling.

In this paper, we present DART-C (Demand-based Allocation for Real-Time Clouds), a framework for automated demand-based vertical scaling of resources. In DART-C, resources dynamically scale according to real-time demands to meet temporary increases or decreases in real-time requirements. We further implement this by extending the OpenStack cloud framework. We also leverage existing work by the RT-Xen and LITMUS^{RT} projects [1][3]. Using DART-C, users that wish to run real-time applications can do so efficiently without over-allocating computational resources.

The remainder of the paper is organized as follows: Section II presents background information and related work. Section III and section IV present our DART-C framework and experimental setup, respectively. Section V presents our results and their discussion. Finally, Section VI presents our conclusions and future work.

II. Background and Related Work

In this work, we use a periodic task model to represent the workload, as presented by Liu and Layland [4]. Here, tasks are represented by period, worst-case execution time, and deadline. Compositional Scheduling Analysis (CSA) is an technique that defines models that allow the resources required to guarantee schedulability in a hierarchical manner [5][6][7]. The CARTS tool, developed by Phan et al., is a tool for applying CSA [8]. CARTS can be used to calculate the amount of resources needed to guarantee that deadlines will be met, according to the aforementioned models. In addition to CARTS, we utilize two projects that enable real-time processing at different levels: LITMUS^{RT} in the operating system, and RT-Xen in the hypervisor.

LITMUS^{RT} provides a real-time scheduler that runs on single or multi-core systems [3]. LITMUS^{RT} is an extension of the Linux kernel. It provides a pluggable scheduling interface to implement real-time scheduling policies with minimal altering of the native Linux kernel.

RT-Xen is an extension of Xen which enables real-time support at the CPU virtualization level by providing a Real-Time-Deferrable-Server (RTDS) scheduler [1]. The RTDS scheduler is an EDF scheduler that exposes the VCPU resource allocations as budgets and periods. Each VCPU is guaranteed to receive the assigned budget every period.

There has also been various work on dynamic resource allocation for cloud environments. Xi et al. present RT-OpenStack, which integrates RT-Xen and OpenStack to achieve a flexible cloud that hosts both general-purpose VMs and real-time VMs through extensions to the RT-Xen scheduler [2]. In their work, general-purpose VMs and real-time VMs share the same CPU resources. The scheduler is able to guarantee that the resources requested by the real-time VMs will not be interrupted by general-purpose VMs. The general-purpose VMs utilize any remaining resources to execute their tasks.

III. Dynamic Real-Time Cloud

This paper addresses the problem of resource allocation in a real-time cloud. Current work has used a static allocation. We explore dynamic allocations to support multi-modal workloads.

To begin, enough resources should be allocated such that real-time tasks are able to meet all deadlines. In general-purpose situations, oversubscribing a system could result in increased latency and decreased throughput per VM, depending on the scheduling and priority scheme. However, when real-time workloads are present, oversubscribing a system can result in missed deadlines, which may not be tolerable. We can utilize Compositional Schedulability Analysis to determine the required resources such that all tasks meet their deadlines, given a real-time task set [7].

Our work addresses resource management for dynamic real-time applications. The previous work in [2] integrates a cloud framework (OpenStack in their case) and real-time capabilities (using RT-Xen). We build upon that by addressing the changing real-time requirements inside each VM. The aforementioned work does not address this issue.

In addition to the real-time capable hypervisor, which enables resource sharing for real-time virtual machines, and the real-time capable guest kernel, which enables tasks to communicate their requirements to the kernel, there are other details that need to be addressed in order to enable real-time computing in the cloud. We detail these below:

1) **Guest Instrumentation**: Guest instrumentation enables dynamic workloads by providing an API that allows a VM to communicate future task set characteristics to the cloud framework. This information is used to calculate new resource allocations. This enables dynamic hard real-time tasks to be supported, as the resources can be adjusted before the application needs it. As a future extension, this instrumentation may be expanded to allow other data to be collected, such as latency and response time.
2) **Dynamic Resource Adjustment**: Many applications exhibit multi-modal execution patterns. These modes result in changing resource requirements to meet real-time constraints. The aforementioned guest API enables the

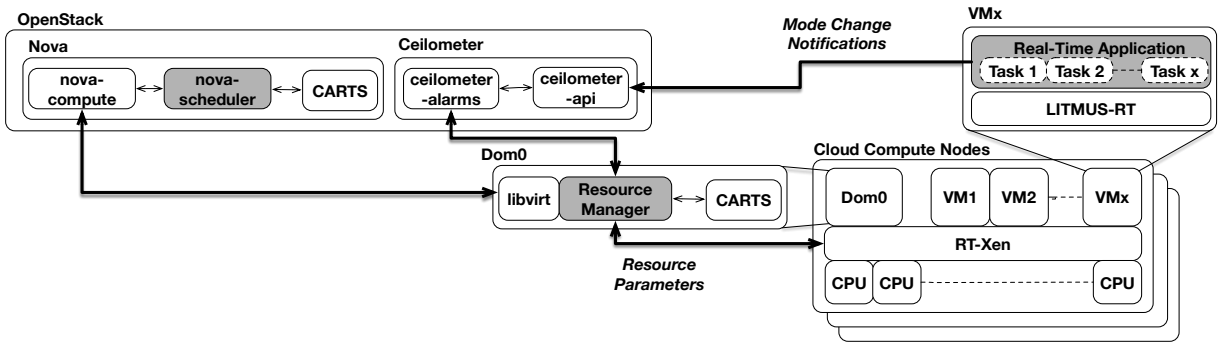

Fig. 2: Dynamic Real-Time Cloud Framework

requirements to be reported. We use CARTS to calculate the new resource requirements. The deadlines are guaranteed to be met due to using CSA theory. The new resource allocation is updated in RT-Xen.

Our framework is shown in Figure 2. The real-time hypervisor is RT-Xen 2.2. In addition to providing an interface to allocate resources according to what is demanded by real-time models, RT-Xen 2.2 allows each VCPU to have its budget and period set individually. Each VM image contains a LITMUS^{RT} 2015.1 patched kernel. This allows real-time scheduling of tasks in the guest.

Some OpenStack services are explicitly used in this work as well. Modifications are made to nova-scheduler to allow real-time scheduling. When a VM is requested, its applications' characteristics are known and provided as scheduler hints. This allows nova-scheduler to utilize CARTS to determine the initial resource allocation. The flavor is used to determine the maximum number of VCPUs that may be allocated to a host. When applications detect an upcoming requirement change, they report it through ceilometer samples. These samples are processed by a resource manager (hosted in Dom0) that will process the revised task characteristics using CARTS and updates the hypervisor settings. All deadlines are guaranteed to be met because CARTS is used to determine the required parameters.

IV. Experimental Setup and Methodology

A. Experimental Setup

In this section, we define our setup for conducting the experiments. The physical configuration is comprised of a HP Proliant DL380 Gen8 server. The server utilizes two Intel Xeon E5-2650 processors in a NUMA configuration, each with eight cores running at 2.00 GHz. Turbo Boost is disabled to allow consistent and predictable performance. Furthermore, hyperthreading is disabled and the power management and frequency scaling are disabled in the BIOS. Each machine is configured with 48 GB of memory with 24 GB per NUMA node. Dom0 is allocated four VCPUs, which leaves 12 PCPUs for domUs. There is no VCPU pinning, meaning all VCPUs are allowed to migrate across any PCPU.

The guest VM images share a common configuration as well. The guest OS is Ubuntu 12.04 running the GNU/Linux 4.1.3 x86_64 kernel. The kernel is patched with LITMUS^{RT} 2015.1. In each VM, the global earliest deadline first scheduler with synchronization support (GSN-EDF) is used.

We utilize a synthetic, computationally-heavy benchmark that is based on the base_task.c provided with the LITMUS^{RT} userspace library. The computation amount is calibrated using accurate measurements from the feather-trace tools provided by LITMUS^{RT} [3]. The computation is then scaled to provide the required worst-case execution time.

B. Methodology

1) Application Definition: A real-time application consists of one or more modes. Each mode can be represented by a task set and an application is in a single mode at any given time. A task set is composed of a number of independent tasks $T_i, i \in 0, ..., n$ where each task $T_i = (p_i, e_i)$ releases jobs with period of p_i and worst-case execution time e_i. The deadline is implicitly set to the period. If a job's finish time exceeds its deadline, then that job has missed its deadline.

For this experiment, we use a real-time application that is inspired by a trading platform application. This platform monitors transactions and news regarding a particular investment, then makes a decision. This can be represented as separate modes of computation. Within each mode, the task period for each task is uniformly distributed between 2ms and 20ms, as motivated by [9]. There are a constant number of tasks across all modes, uniformly distributed between [1,3]. The execution time is also uniformly distributed between 1ms and 1.8ms, and constant across modes. Our rationale for this application model is that the number of stocks that will be monitored will stay relatively constant. We also estimate that the amount of computation to make a decision will remain constant, regardless of how often the user wants a decision made. The period for each mode differs, as a user may want more frequent monitoring depending on events such as earnings releases or trading volume changes. As the period changes for each mode, the utilization demands will vary, representing different modes of computation. In these experiments, the mode duration and transitions are random. In our experiments,

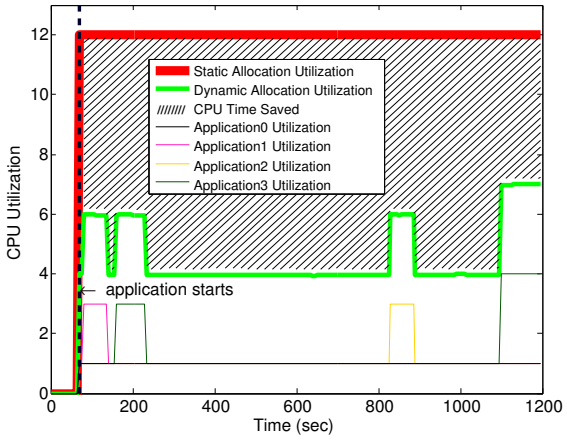

Fig. 3: Trading Platform Resource Utilization

we run four instances of the application, each executing in a dedicated VM.

2) Metrics: The primary metric we evaluate this system by is the total system VCPU utilization required by the running applications. We assume that each VM runs a single application. Each application consists of task sets, where $T_i(t)$ is the task set at time t. For a cloud with n applications and therefore n VMs, the resource requirement R in the static case is: $R_{static} = CARTS(T^{max})$, where $T^{max} = \{T_0^{max}, T_1^{max}, ..., T_n^{max}\}$ and $T_i^{max} =$ *the task set with maximum CPU utilization for VM* i. The requirement for the dynamic case is: $R_{dynamic}(t) = CARTS(T(t))$, where $T(t) = \{T_0(t), T_1(t), ..., T_n(t)\}$. Therefore, it can be seen that the static utilization is the upper bound of the dynamic case.

V. Results and Discussion

Figure 3 illustrates the results from our system resource utilization experiments. The resource allocation is sampled every five seconds. The Static Allocation Utilization is the maximum required utilization required by the most intensive modes in the application. This is the amount that would have to be allocated if there were no adjustment (as in the static allocation case). When using the most demanding modes as input to CARTS, we find that the required resources are 12 cores. Dynamic Allocation Utilization is the utilization using our framework. Each instance's individual utilization is also illustrated. Finally, the beginning of the application set execution is annotated.

We observe that the amount of CPU time saved over the applications execution is 62% without any deadline misses. The amount is calculated by computing the difference in area between the Static and Dynamic Allocation curves, shown in Figure 3. These savings can then be used by other tasks, as demonstrated by [2].

VI. Conclusion and Future Work

In this paper, we presented DART-C, a framework for automated demand-based vertical scaling of resource allocations in a dynamic, real-time capable cloud. We then described and evaluated the benefits that this work provides, in terms of the consumed resource utilization. We collected this data by running synthetic workloads inspired by real-life applications. Our work shows a 62% savings compared to a static allocation cloud.

While our framework implementation in this paper enables handling dynamic real-time cases, more study needs to be done to explore the trade-offs between performance and resource usage for soft real-time tasks, where deadlines are not strictly required. DART-C can enable this by making use of periodic reporting of run-time characteristics through the same API that is used for reporting mode changes. That data could be used to scale resources to meet performance targets, similarly to what has been done for servers to optimize energy efficiency [10].

Many dynamic real-time workloads also require a high level of reliability. Enabling migration while still meeting real-time requirements can enable high-availability in real-time clouds. We would like to investigate a framework that can enable high-availability in real-time clouds using migration of VMs.

VII. Acknowledgments

This work was developed with support from the Office of Naval Research under grant #N00014-14-1-0035. The authors would also like to acknowledge Mikyung Kang for her assistance with experimental setup and debugging.

References

[1] S. Xi, J. Wilson, C. Lu, and C. Gill, "RT-Xen: Towards Real-Time Hypervisor Scheduling in Xen," in *Embedded Software (EMSOFT), 2011 Proceedings of the International Conference on*, Oct 2011, pp. 39–48.

[2] S. Xi, C. Li, C. Lu, C. Gill, M. Xu, L. Phan, I. Lee, and O. Sokolsky, "RT-Open Stack: CPU Resource Management for Real-Time Cloud Computing," in *Cloud Computing (CLOUD), 2015 IEEE 8th International Conference on*, June 2015, pp. 179–186.

[3] J. Calandrino, H. Leontyev, A. Block, U. Devi, and J. Anderson, "LITMUS^{RT} : A Testbed for Empirically Comparing Real-Time Multiprocessor Schedulers," in *Real-Time Systems Symposium, 2006. RTSS '06. 27th IEEE International*, Dec 2006, pp. 111–126.

[4] C. L. Liu and J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment," *J. ACM*, vol. 20, no. 1, pp. 46–61, Jan. 1973.

[5] I. Shin and I. Lee, "Periodic Resource Model for Compositional Real-Time Guarantees," in *Real-Time Systems Symposium, 2003. RTSS 2003. 24th IEEE*, Dec 2003, pp. 2–13.

[6] A. Easwaran, I. Shin, and I. Lee, "Optimal Virtual Cluster-based Multiprocessor Scheduling," *Real-Time Syst.*, vol. 43, no. 1, pp. 25–59, Sep. 2009.

[7] M. Xu, L. Phan, I. Lee, O. Sokolsky, S. Xi, C. Lu, and C. Gill, "Cache-Aware Compositional Analysis of Real-Time Multicore Virtualization Platforms," in *Real-Time Systems Symposium (RTSS), 2013 IEEE 34th*, Dec 2013, pp. 1–10.

[8] L. T. X. Phan, J. Lee, A. Easwaran, V. Ramaswamy, S. Chen, I. Lee, and O. Sokolsky, "CARTS: A Tool for Compositional Analysis of Real-time Systems," *SIGBED Rev.*, vol. 8, no. 1, pp. 62–63, Mar. 2011.

[9] J. Hasbrouck and G. Saar, "Low-Latency Trading," in *Johnson School Research Paper Series*. Chicago Meetings Paper, 2013, no. 35-201.

[10] D. Lo, L. Cheng, R. Govindaraju, L. Barroso, and C. Kozyrakis, "Towards Energy Proportionality for Large-Scale Latency-Critical Workloads," in *Computer Architecture (ISCA), 2014 ACM/IEEE 41st International Symposium on*, June 2014, pp. 301–312.