A Cloud Middleware for Assuring Performance and High Availability of Soft Real-time Applications[☆]

Kyoungho An, Shashank Shekhar, Faruk Caglar, Aniruddha Gokhale[a], Shivakumar Sastry[b]

[a] *Institute for Software Integrated Systems (ISIS)*
Department of Electrical Engineering and Computer Science
Vanderbilt University, Nashville, TN 37235, USA
Email: {kyoungho.an, shashank.shekhar, faruk.caglar, a.gokhale}@vanderbilt.edu
[b] *Complex Engineered Systems Lab*
Department of Electrical and Computer Engineering
The University of Akron, Akron, OH 44325, USA
Email: ssastry@uakron.edu

Abstract

Applications are increasingly being deployed in the cloud due to benefits stemming from economy of scale, scalability, flexibility and utility-based pricing model. Although most cloud-based applications have hitherto been enterprise-style, there is an emerging need for hosting real-time streaming applications in the cloud that demand both high availability and low latency. Contemporary cloud computing research has seldom focused on solutions that provide both high availability and real-time assurance to these applications in a way that also optimizes resource consumption in data centers, which is a key consideration for cloud providers. This paper makes three contributions to address this dual challenge. First, it describes an architecture for a fault-tolerant framework that can be used to automatically deploy replicas of virtual machines in data centers in a way that optimizes resources while assuring availability and responsiveness. Second, it describes the design of a pluggable framework within the fault-tolerant architecture that enables plugging in different placement algorithms for VM replica deployment. Third, it

[☆]This work was supported in part by NSF awards CAREER/CNS 0845789 and SHF/CNS 0915976. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

illustrates the design of a framework for real-time dissemination of resource utilization information using a real-time publish/subscribe framework, which is required by the replica selection and placement framework. Experimental results using a case study that involves a specific replica placement algorithm are presented to evaluate the effectiveness of our architecture.

Keywords: high availability, real-time, quality of service, cloud computing, middleware, framework.

---

1. Introduction

Cloud computing is a large-scale distributed computing platform based on the principles of utility computing that offers resources such as CPU and storage, systems software, and applications as services over the Internet [1]. The driving force behind the success of cloud computing is economy of scale. Traditionally, cloud computing has focused on enterprise applications. Lately, however, a class of soft real-time applications that demand both high availability and predictable response times are moving towards cloud-based hosting [2, 3, 4].

To support soft real-time applications in the cloud, it is necessary to satisfy the response time, reliability and high availability demands of such applications. Although the current cloud-based offerings can adequately address the performance and reliability requirements of enterprise applications, new algorithms and techniques are necessary to address the Quality of Service (QoS) needs, *e.g.*, low-latency needed for good response times and high availability, of performance-sensitive, real-time applications.

For example, in a cloud-hosted platform for personalized wellness management [4], high-availability, scalability and timeliness is important for providing on-the-fly guidance to wellness participants to adjust their exercise or physical activity based on real-time tracking of the participant's response to current activity. Assured performance and high availability is important because the wellness management cloud infrastructure integrates and interacts with the exercise machines both to collect data about participant performance and to adjust the intensity and duration of the activities.

Prior research in cloud computing has seldom addressed the need for supporting real-time applications in the cloud.[1] However, there is a grow-

[1]In this research we focus on soft real-time applications since it is unlikely that hard

ing interest in addressing these challenges as evidenced by recent efforts [5]. Since applications hosted in the cloud often are deployed in virtual machines (VMs), there is a need to assure the real-time properties of the VMs. A recent effort on real-time extensions to the Xen hypervisor [5] has focused on improving the scheduling strategies in the Xen hypervisor to assure real-time properties of the VMs. While timeliness is a key requirement, high availability is also an equally important requirement that must be satisfied.

Fault tolerance based on redundancy is one of the fundamental principles for supporting high availability in distributed systems. In the context of cloud computing, the Remus [6] project has demonstrated an effective technique for VM failover using one primary and one backup VM solution that also includes periodic state synchronization among the redundant VM replicas. The Remus failover solution, however, incurs shortcomings in the context of providing high availability for soft real-time systems hosted in the cloud.

For instance, Remus does not focus on effective replica placement. Consequently, it cannot assure real-time performance after a failover decision because it is likely that the backup VM may be on a physical server that is highly loaded. The decision to effectively place the replica is left to the application developer. Unfortunately, any replica placement decisions made offline are not attractive for a cloud platform because of the substantially changing dynamics of the cloud platform in terms of workloads and failures. This requirement adds an inherent complexity for the developers who are responsible for choosing the right physical host with enough capacity to host the replica VM such that the real-time performance of applications is met. It is not feasible for application developers to provide these solutions, which calls for a cloud platform-based solution that can shield the application developers from these complexities.

To address these requirements, this paper makes the following three contributions described in Section 4:

1. We present a fault-tolerant architecture in the cloud geared to provide high availability and reliability for soft real-time applications. Our solution is provided as a middleware that extends the Remus VM failover solution [6] and is integrated with the OpenNebula cloud infrastructure software [7] and the Xen hypervisor [8]. Section 4.3 presents a hierarchical architecture motivated by the need for separation of concerns

real-time and safety-critical applications will be hosted in the cloud.

and scalability.

2. In the context of our fault-tolerant architecture, Section 4.4 presents the design of a pluggable framework that enables application developers to provide their strategies for choosing physical hosts for replica VM placement. Our solution is motivated by the fact that not all applications will impose exactly the same requirements for timeliness, reliability and high availability, and hence a "one-size-fits-all" solution is unlikely to be acceptable to all classes of soft real-time applications. Moreover, developers may also want to fine tune their choice by trading off resource usage and QoS properties with the cost incurred by them to use the cloud resources.
3. For the first two contributions to work effectively, there is a need for a low-overhead, and real-time messaging between the infrastructure components of the cloud infrastructure middleware. This messaging capability is needed to reliably gather real-time resource utilization information from the cloud data center servers at the controllers that perform resource allocation and management decisions. To that end Section 4.5 presents a solution based on real-time publish/subscribe (pub/sub) that extends the OMG Data Distribution Service (DDS) [9] with additional architectural elements that fit within our fault-tolerant middleware.

To evaluate the effectiveness of our solution, we use a representative soft real-time application hosted in the cloud and requiring high availability. For replica VM placement, we have developed an Integer Linear Programming (ILP) formulation that can be plugged into our framework. This placement algorithm allocates VMs and their replicas to physical resources in a data center that satisfies the QoS requirements of the applications. We present results of experimentation focusing on critical metrics for real-time applications such as end-to-end latency and deadline miss ratio. Our goal in focusing on these metrics is to demonstrate that recovery after failover has negligible impact on the key metrics of real-time applications. Moreover, we also show that our high availability solution at the infrastructure-level can co-exist with an application-level fault tolerance capability provided by the application.

The rest of this paper is organized as follows: Section 2 describes relevant related work comparing it with our contributions; Section 3 provides background information on the underlying technologies we have leveraged in our solution; Section 4 describes the details of our system architecture;

Section 5 presents experimental results; and Section 6 presents concluding remarks alluding to future work.

2. Related Work

Prior work in the literature of high availability solutions, VM placement strategies, and resource monitoring are related to the three research contributions we offer in this paper. In this section, we present a comparative analysis of the literature and how our solutions fit in this body of knowledge.

2.1. Underlying Technology: High Availability Solutions for Virtual Machines

To ensure high-availability, we propose a fault-tolerant solution that is based on the continuous checkpointing technique developed for the Xen hypervisor called Remus [6]. We discuss the details and shortcomings of Remus in Section 3.2.

Several other high availability solutions for virtual machines are reported in the literature. VMware fault-tolerance [10] runs primary and backup VMs in lock-step using deterministic replay. This keeps both the VMs in sync but it requires execution at both the VMs and needs high quality network connections. In contrast, our model focuses on a primary-backup scheme for VM replication that does not require execution on all replica VMs.

Kemari [11] is another approach that uses both lock-stepping and continuous check-pointing. It synchronizes primary and secondary VMs just before the primary VM has to send an event to devices, such as storage and networks. At this point, the primary VM pauses and Kemari updates the state of the secondary VM to the current state of primary VM. Thus, VMs are synchronized with lower complexity than lock-stepping. External buffering mechanisms are used to improve the output latency over continuous check-pointing. However, we opted for Remus since it is a mature solution compared to Kemari.

Another important work on high availability is HydraVM [12]. It is a storage-based, memory-efficient high availability solution which does not need passive memory reservation for backups. It uses incremental checkpointing like Remus [6], but it maintains a complete recent image of VM in shared storage instead of memory replication. Thus, it claims to reduce hardware costs for providing high availability support and provide greater flexibility as recovery can happen on any physical host having access to

shared storage. However, the software is not open-source or commercially available.

2.2. Approaches to Virtual Machine Placement

Virtual machine placement on physical hosts in the cloud critically affects the performance of the application hosted on the VMs. Even when the individual VMs have a share of the physical resources, effects of context switching, network performance and other systemic effects [13, 14, 15, 16] can adversely impact the performance of the VM. This is particularly important when high availability solutions based on replication must also consider performance as is the case in our research. Naturally, more autonomy in VM placement is desirable.

The approach proposed in [17] is closely related to the scheme we propose in this paper. The authors present an autonomic controller that dynamically assigns VMs to physical hosts according to policies specified by the user. While the scheme we propose also allows users to specify placement policies and algorithms, we dynamically allocate the VMs in the context of a fault-tolerant cloud computing architecture that ensures high-availability solutions.

Lee et al. [18] investigated VM consolidation heuristics to understand how VMs perform when they are co-located on the same host machine. They also explored how the resource demands such as CPU, memory, and network bandwidth are handled when consolidated. The work in [19] proposed a modified Best Fit Decreasing (BFD) algorithm as a VM reallocation heuristic for efficient resource management. The evaluation in the paper showed that the suggested heuristics minimize energy consumption while providing improved QoS. Our work may benefit from these prior works and we are additionally concerned with placing replicas in a way that applications continues to obtain the desired QoS after a failover.

2.3. Resource Monitoring in Large Distributed Systems

Contemporary compute clusters and grids have provided special capabilities to monitor the distributed systems via frameworks, such as Ganglia [20] and Nagios [21]. According to [22], one of the distinctions between grids and cloud is that cloud resources also include virtualized resources. Thus, the grid- and cluster-based frameworks are structured primarily to monitor physical resources only, and not a mix of virtualized and physical resources. Even though some of these tools have been enhanced to work in the cloud,

e.g., virtual machine monitoring in Nagios[1] and customized scripts used in Ganglia, they still do not focus on the timeliness and reliability of the dissemination of monitored data that is essential to support application QoS in the cloud. A work that comes closest to ours, [23], provides a comparative study of publish/subscribe middleware for real-time grid monitoring in terms of real-time performance and scalability. While this work also uses publish/subscribe for resource monitoring, it is done in the context of grid and hence incurs the same limitations. [24] also introduced a use of publish/subscribe middleware for real-time resource monitoring in distributed environment.

In other recent works, [25] presents a virtual resource monitoring model and [26] discusses a cloud monitoring architecture for private clouds. Although these prior works describe cloud monitoring systems and architectures, they do not provide experimental performance results of their models for properties such as system overhead and response time. Consequently, we are unable to determine their relevance to support timely dissemination of resource information and hence their ability to host mission-critical applications in the cloud. Latency results using RESTful services for resource monitoring are described in [27], however, they are not able to support diverse and differentiated service levels for cloud clients we are able to provide.

2.4. Comparative Analysis

Although there are several findings in the literature that relate to our three contributions, none of these approaches offer a holistic framework that can be used in a cloud infrastructure. Consequently, the combined effect of individual solutions has not been investigated. Our work is a step in the direction of fulfilling this void. Integrating the three different approaches is not straightforward and requires good design decisions, which we have demonstrated with our work and presented in the remainder of this paper.

3. Overview of Underlying Technologies

Our middleware solution is designed in the context of existing cloud infrastructure middleware, such as OpenNebula [7], and hypervisor technologies, such as Xen [8]. In particular, our solution is based on Remus [6],

[1] `http://people.redhat.com/~rjones/nagios-virt`

which provides high availability to VMs that use the Xen hypervisor, and the real-time pub/sub technology provided by the OMG DDS [9] for the scalable dissemination of resource utilization information. For completeness, we describe these building blocks in more detail here.

3.1. Cloud Infrastructure and Virtualization Technologies

Contemporary cloud infrastructure platforms, such as OpenStack [28], Eucalyptus [29], or OpenNebula [7], manage the artifacts of the cloud including the physical servers, networks, and other equipment, such as storage devices. One of the key responsibilities such infrastructure is to manage user applications on the virtualized servers in the data center. Often, these platforms are architected in a hierarchical manner with a master or controller node oversees the activities of the worker nodes that host applications. In the OpenNebula platform we use, the master node is called the *Front-end Node*, and the worker nodes are called the *Cluster Nodes.*

Hypervisors, such as Xen [8] and KVM [30], offer server virtualization that enables multiple applications to execute within isolated virtual machines. The hypervisor manages the virtual machines and ensures both performance and security isolation between different virtual machines hosted on the same physical server. To ensure that our solution can be adopted in a range of hypervisors, we use the *libvirt* [31] software suite that provides a portable approach to manage virtual machines. By providing a common API, libvirt is able to interoperate with a range of hypervisors and virtualization technologies.

3.2. Remus High Availability Solution

Remus [6] is a software system built for the Xen hypervisor that provides OS- and application-agnostic high-availability on commodity hardware. Remus provides seamless failure recovery and does not require lock step-based, whole-system replication. Instead, the use of speculative execution in the Remus approach ensures that the performance degradation due to replication is kept to a minimum. Speculative execution decouples the execution of the application from state synchronization between replica VMs by interleaving these operations and, hence, not forcing synchronization between replicas after every update made by the application.

Remus uses a pair of replica VMs: a primary and a backup. Since Remus provides protection against single host fail-stop failures only, if both the primary and backup hosts fail concurrently, the failure recovery will not

be seamless; however, Remus ensures that the system's data will be left in a consistent state even if the system crashes. Additionally, Remus is not concerned with where the primary and backup replicas are placed in the data center. Consequently, it cannot guarantee any performance properties for the applications. The VM placement is the responsibility of the user, which we have shown to be a significant complexity for the user. Our VM failover solution leverages Remus while addressing these limitations in Remus.

3.3. OMG Data Distribution Service

The OMG DDS [9] supports anonymous, asynchronous and scalable data-centric pub/sub communication model [32] where publishers and subscribers exchange topic-based data. OMG DDS specifies a layered architecture comprising three layers – two of these layers make DDS a promising design choice for use in the scalable and timely dissemination of resource usage information in a cloud platform. One layer, called the Data Centric Publish/Subscribe (DCPS), provides a standard API for data centric, topic-based, real-time pub/sub [33]. It provides efficient, scalable, predictable, and resource-aware data distribution capabilities. The DCPS layer operates over another layer that provides a DDS interoperability wire protocol [34] called Real-Time Publish/Subscribe (RTPS).

One of the key features of DDS when compared to other pub/sub middleware is its rich support for QoS offered at the DCPS layer. DDS provides the ability to control the use of resources, such as network bandwidth and memory, and non-functional properties of the topics, such as persistence, reliability, timeliness, and others [35]. We leverage these scalability and QoS capabilities of DDS to support real-time resource monitoring in the cloud.

4. Middleware for Highly Available VM-hosted Soft Real-time Applications

This section presents our three contributions that collectively offer a high availability middleware architecture for soft real-time applications deployed in virtual machines in cloud data centers. We first describe the architecture and then describe the three contributions in detail.

4.1. Architectural Overview

The architecture of our high-availability middleware, as illustrated in Figure 1, comprises a Local Fault Manager (LFM) for each physical host, and

a replicated Global Fault Manager (GFM) to manage the cluster of physical machines. The inputs to the LFMs are the resource information of physical hosts and VMs gathered directly from the hypervisor. We collect information for resources such as the CPU, memory, network, storage, and processes.

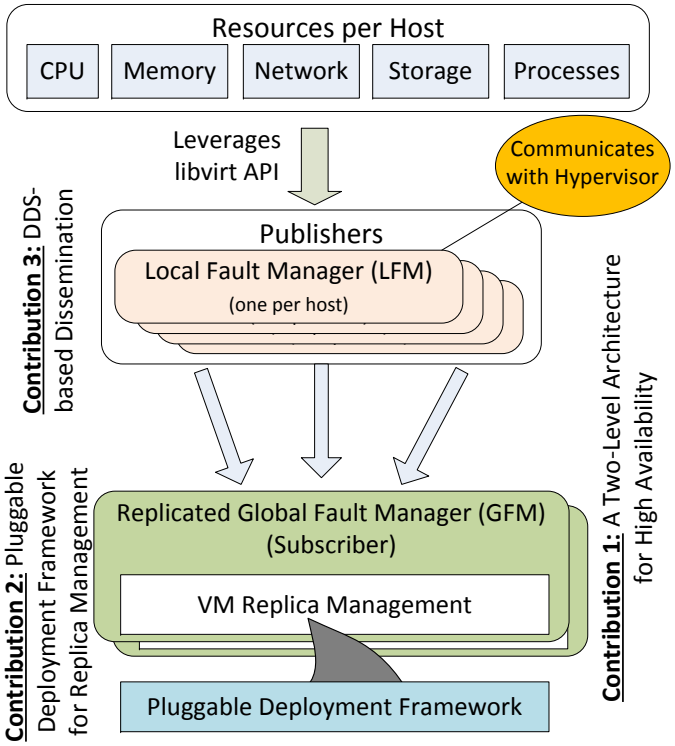

Figure 1: Conceptual System Design Illustrating Three Contributions

The GFM is responsible for making decisions on VM replica management including the decisions to place a replica VM. It needs timely resource utilization information from the LFMs. Our DDS-based framework enables scalable and timely dissemination of resource information from LFMs (the publishers) to the GFM (the subscriber). Since no one-size-fits-all replica placement strategy is appropriate for all applications, our GFM supports a pluggable replica placement framework.

4.2. Roles and Responsibilities

Before delving into the design rationale and solution details, we describe how the system will be used in the cloud. Figure 2 shows a use case diagram for our system in which roles and responsibilities of the different software components are defined. A user in the role of a system administrator will configure and run a GFM service and the several LFM services. A user in the role of a system developer can implement deployment algorithms to find and use a better deployment solution. The LFM services periodically update resource information of VMs and hosts as configured by the user. The GFM service uses the deployment algorithms and the resource information to create a deployment plan for replicas of VMs. Then, the GFM sends messages to LFMs to run a backup process via high-availability solutions that leverages Remus.

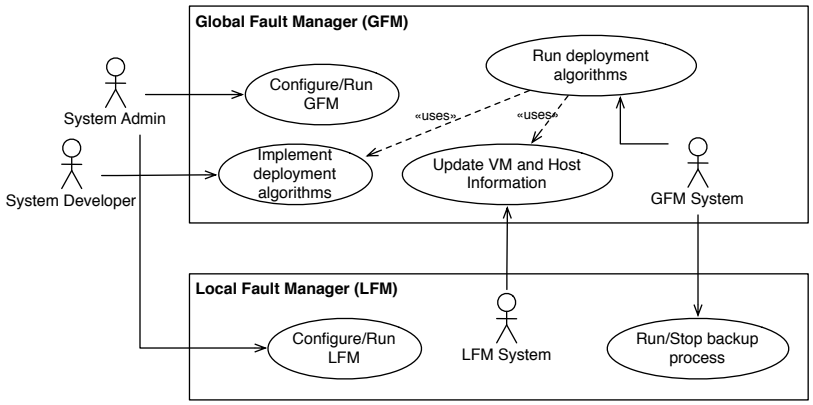

Figure 2: Roles and Responsibilities

4.3. Contribution 1: High-Availability Solution

This section presents our first contribution that deals with providing a high availability middleware solution for VMs running soft real-time applications. Our solution assumes that a VM-level fault recovery is already available via solutions, such as Remus [6].

4.3.1. Rationale: Why a Hierarchical Model?

Following the strategy in Remus, we host the primary and backup VMs on different physical servers to support the fault tolerance.

In a data center with hundreds of thousands of physical servers, a Remus-based solution managing fault tolerance for different applications may be deployed on every server. Remus makes no effort to determine the effective placement of replica VMs; it just assumes that a replica pair exists. For our solution, however, assuring the QoS of the soft real-time systems requires effective placement of replica VMs. In turn, this requires real-time monitoring of the resource usage on the physical servers to make efficient placement decisions.

A centralized solution that manages faults across an entire data center is infeasible. Moreover, it is not feasible for some central entity to poll every server in the data center for resource availability and their usage. Thus, an appropriate choice is to develop a hierarchical solution based on the principles of separation of concerns. At the local level (*i.e.*, host level), a fault management logic can interact with its local Remus software while also being responsible for collecting the local resource usage information. At the global level, a fault management logic can decide effective replica placement based on the timely resource usage information acquired from the local entities.

Although a two-level solution is described, for scalability reasons, multiple levels can be introduced in the hierarchy where a large data center can be compartmentalized into smaller regions.

4.3.2. Design and Operation

Our hierarchical solution is to utilize several Local Fault Managers (LFMs) associated with a single Global Fault Manager (GFM) in adjacent levels of the hierarchy. The GFM coordinates deployment plans of VMs and their replicas by communicating with the LFMs. Every LFM retrieves resource information from a VM that is deployed in the same physical machine as the LFM, and sends the information periodically to a GFM. We focus on addressing the deployment issue because existing solutions such as Remus delegates the responsibility of placing the replica VM onto the user. An arbitrary choice may result in severe performance degradation for the applications running in the VMs.

The replica manager is the core component of the GFM and is responsible for running the deployment algorithm provided by a user of the framework. This component determines the physical host machine where the replica of a VM should be replicated as a backup. The location of the backup is then supplied to the LFM running on the host machine where the VM is located to take the required actions, such as informing the local Remus of its backup

copy.

The LFM runs a High-Availability Service (HAS) that is based on the Strategy pattern [36]. This interface includes starting and stopping replica operations, and automatic failover from a primary VM to a backup VM in case of a failure. The use of the strategy pattern enables us to use a solution different from Remus, if one were to be available. This way we are not tightly coupled with Remus. Once the HAS is started and while it is operational, it keeps synchronizing the state of a primary VM to a backup VM. If a failure occurs during this period, it switches to the backup VM making it the active primary VM. When the HAS is stopped, it stops the synchronization process and high-availability is discontinued.

In the context of the HAS, the job of the GFM is to provide each LFM with backup VMs that can be used when the HAS is executed. In the event of failure of a primary VM, the HAS ensures that the processing switches to the backup VM and it becomes the primary VM. This event is triggered when the LFM informs GFM of the failure event and requests additional backup VMs on which a replica can start. It is the GFM's responsibility to provide resources to the LFM in a timely manner so that the latter can move from a crash consistent state to seamless recovery fault tolerant state as soon as possible thereby assuring average response times of performance-sensitive soft real-time applications.

In the architecture shown in Figure 3, replicas of VMs are automatically deployed in hosts assigned by a GFM and LFMs. The following are the steps of the system described in the figure.

1. A GFM service is started, and the service waits for connections from LFMs.
2. LFMs will join the system by connecting to the GFM service.
3. The joined LFMs periodically send their individual resource usage information of VMs hosted on their nodes as well as that of the physical host, such as CPU, memory, and network bandwidth to the GFM using the DDS solution described in Section 4.5.
4. Based on the resource information, the GFM determines an optimal deployment plan for the joined physical hosts and VMs by running a deployment algorithm, which can be supplied and parametrized by users as described in Section 4.4.
5. The GFM will notify LFMs to execute HAS in LFMs with information of source VMs and destination hosts.

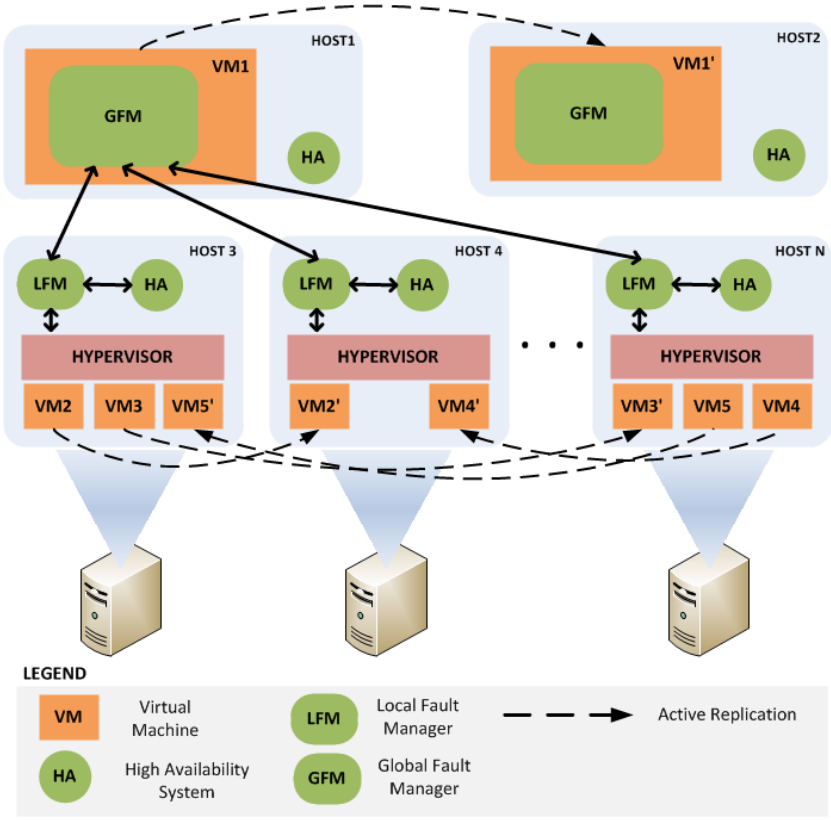

Figure 3: System Architecture

A GFM service can be deployed on a physical host machine or inside a virtual machine. In our system design, to avoid a single point of failure of a GFM service, a GFM is deployed in a VM and a GFM's VM replica is located in another physical host. When the physical host where the GFM is located fails, the backup VM containing the GFM service is promoted to primary and the GFM service continues to its execution via the high availability solution.

On the other hand, LFMs are placed in physical hosts used to run VMs in data centers. LFMs work with a hypervisor and a high availability solution (Remus in our case) to collect resource information of VMs and hosts and to replicate VMs to other backup hosts, respectively. Through the high availability solution, a VM's disk, memory, and network connections are actively replicated to other hosts and a replication of the VM in a backup host is instantiated when the primary VM is failed.

4.4. Contribution 2: Pluggable Framework for Virtual Machine Replica Placement

This section presents our second contribution that deals with providing a pluggable framework for determining VM replica placement.

4.4.1. Rationale: Why a Pluggable Framework?

Existing solutions for VM high availability, such as Remus, delegate the task of choosing the physical host for the replica VM to the user. This is a significant challenge since a bad choice of a heavily loaded physical host may result in performance degradation. Moreover, a static decision is also not appropriate since a cloud environment is highly dynamic. To provide maximal autonomy in this process requires online deployment algorithms that make decisions on VM and replica VM placement.

Deployment algorithms determine which host machine should store a VM and its replica in the context of fault management. There are different types of algorithms to make this decision. Optimization algorithms, such as bin packing, genetic algorithms, multiple knapsack, and simulated annealing are some of the choices used to solve similar problems in a large number of industrial applications today. Moreover, different heuristics of the bin packing algorithm are commonly utilized techniques for VM replica placement optimization, in particular.

Solutions generated by such algorithms and heuristics have different properties. Similarly, the runtime complexity of these algorithms is different. Since different applications may require different placement decisions and may also impose different constraints on the allowed runtime complexity of the placement algorithm, a one-size-fits-all solution is not acceptable. Thus, we needed a pluggable framework to decide VM replica placement.

4.4.2. Design of a Pluggable Framework for Replica VM Placement

In bin packing algorithms [37], the goal is to use minimum number of bins to pack the items of different sizes. Best-Fit, First-Fit, First-Fit-Decreasing, Worst-Fit, Next-Fit, and Next-Fit-Decreasing are the different heuristics of this algorithm. All these heuristics will be part of the middleware we are designing, and will be provided to the framework user to run the bin packing algorithm.

In our framework, we view VMs as items and the host machines as the bins. Resource information from the VMs, are utilized as weights to employ

the bin packing algorithm. Resource information is aggregated into one single scalar value, and one dimensional bin packing is employed to find the best host machine where the replica of a VM will be stored. Our framework uses the Strategy pattern to enable plugging in different VM replica placement algorithms. A concrete problem we have developed and used in our replication manager is described in Section 5.

4.5. Contribution 3: Scalable and Real-time Dissemination of Resource Usage

This section presents our third contribution that deals with using a pub/sub communication model for real-time resource monitoring. Before delving into the solution, we first provide a rationale for using a pub/sub solution.

4.5.1. Rationale: Why Pub/Sub for Cloud Resource Monitoring and Dissemination?

Predictable response times are important to host soft real-time applications in the cloud. This implies that even after a failure and recovery from failures, applications should continue to receive acceptable response times. In turn this requirement requires that the backup replica VMs be placed on physical hosts that will deliver the application-expected response times.

A typical data center comprises hundreds of thousands of commodity servers that host virtual machines. Workloads on these servers (and hence the VMs) demonstrates significant variability due to newly arriving customer jobs and the varying number of resources they require. Since these servers are commodity machines and due to the very large number of such servers in the data center, failures are quite common.

Accordingly, any high-availability solution for virtual machines that supports real-time applications must ensure that primary and backup replicas must be hosted on servers that have enough available resources to meet the QoS requirements of the soft real-time applications. Since the cloud environment is a highly dynamic environment with fluctuating loads and availability of resources, it is important that real-time information of the large number of cloud resources be available to the GFM to make timely decisions on replica placement.

It is not feasible to expect the GFM to pull the resource information from every physical server in the data center. First, this will entail maintaining a TCP/IP connection. Second, failures of these physical servers will disrupt the operation of the GFM. A better approach is for resource information

to be asynchronously pushed to the GFM. We surmise therefore that the pub/sub [32] paradigm has a vital role to play in addressing these requirements. A solution based on the "push" model, where information is pushed to the GFM from the LFMs asynchronously, is a scalable alternative. Since performance, scalability, and timeliness in information dissemination are key objectives, the OMG Data Distribution Service (DDS) [9] for data-centric pub/sub is a promising technology that can be adopted to disseminate resource monitoring data in cloud platforms.

4.5.2. Design of a DDS-based Cloud Resource Monitoring Framework

A solution based on the "push" model where information can be pushed to the GFM asynchronously lends itself to a more scalable alternative.

The GFM is consuming information from the LFMs and is a subscriber. The resources themselves are the publishers of information. Since LFMs are hosted on the physical hosts from which the resource utilization information is collected, the LFMs can also play the role of a publisher. The roles are reversed when a decision from GFM is pushed to the LFMs.

Figure 4 depicts the DDS entities used for our framework. Each LFM node has its domain participant containing a DataWriter and a DataReader. A DataWriter in LFM is configured to periodically disseminate resource information of VMs to a DataReader in GFM via a LFM Topic. The LFM obtains this information via the libvirt APIs. A DataReader in LFM is to receive command messages from a DataWriter in GFM to start and stop a HAS via GFM Topic when a decision is made by algorithms in GFM.

5. Experimental Results and Case Study

In this section we present results to show that our solution can seamlessly and effectively leverage existing solutions for fault tolerance in the cloud.

5.1. Rationale for Experiments

Our high-availability solution for cloud-hosted soft real-time applications leverages existing VM-based solutions, such as the one provided by Remus. Moreover, it is also possible that the application running inside the VM itself may provide its own application-level fault tolerance. Thus, it is important for us to validate that our high availability solution can work seamlessly and effectively in the context of existing solutions.

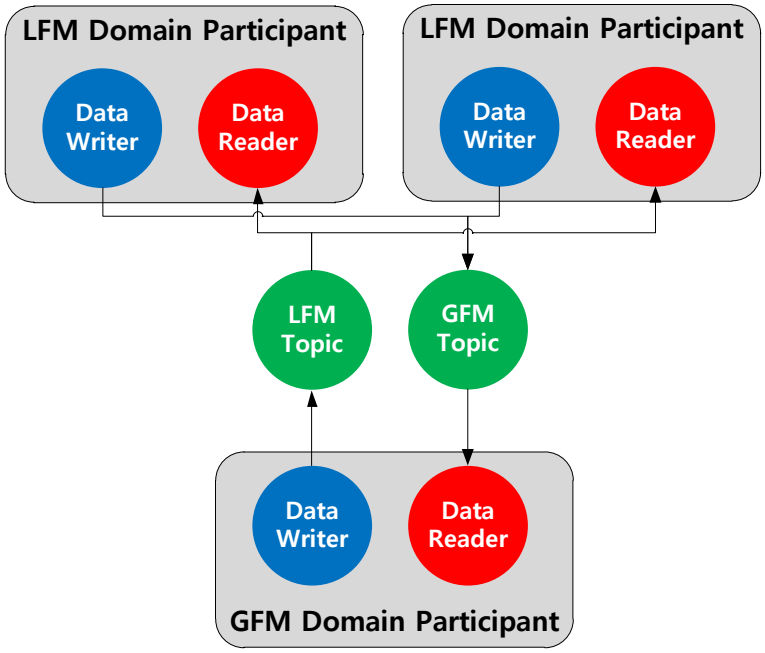

Figure 4: DDS Entities

Moreover, since we provide a pluggable framework for replica placement, we must validate our approach in the context of a concrete placement algorithm that can be plugged into our framework. To that end, we have developed a concrete placement algorithm, which we describe below and used it in the evaluations.

5.2. Representative Applications and Evaluation Testbed

To validate both the claims: (a) support for high-availability soft real-time applications, and (b) seamless co-existence with other cloud-based solutions, we have used two representative soft real-time applications. For the first set of validations, we have used an existing benchmark application that has the characteristics of a real-time application [38]. To demonstrate how our solution can co-exist with other solutions, we used a *word count* application that provides its own application-level fault-tolerance. We show how our solution can co-exist with different fault-tolerance solutions.

Our private cloud infrastructure for both the experiments we conducted

comprises a cluster of 20 rack servers, and Gigabit switches. The cloud infrastructure is operated using OpenNebula 3.0 with shared file systems using NFS (Network File System) for distributing virtual machine images. Table 1 provides the configuration of each rack server used as a clustered node.

Table 1: Hardware and Software specification of Cluster Nodes

Processor	2.1 GHz Opteron
Number of CPU cores	12
Memory	32 GB
Hard disk	8 TB
Operating System	Ubuntu 10.04 64-bit
Hypervisor	Xen 4.1.2
Guest virtualization mode	Para

Our guest domains run Ubuntu 11.10 32-bit as operating systems, and each guest domain has 4 virtual CPUs and 4GB of RAM.

5.3. A Concrete VM Placement Algorithm

Our solution provides a framework that enables plugging in different user-supplied VM placement algorithms. We expect that our framework will compute replica placement decisions in an online manner in contrast to making offline decisions. We now present an instance of VM replica placement algorithm we have developed. We have formulated it as an Integer Linear Programming (ILP) problem.

In our ILP formulation we assume that a data center comprises multiple hosts. Each host can in turn consist of multiple VMs. We also account for the resource utilizations of the physical host as well as the VMs on each host. Furthermore, not only do we account for CPU utilizations but also memory and network bandwidth usage. All of these resources are accounted for in determining the placement of the replicas because on a failover we expect our applications to continue to receive their desired QoS properties. Table 2 describes the variables used in our ILP formulation.

We now present the ILP problem formulation shown below with the defined constraints that need to be satisfied to find an optimal allocation of VM replicas. The objective function of the problem is to minimize the number of

Table 2: Notation and Definition of the ILP Formulation

Notation	Definition
x_{ij}	Boolean value to determine the i^{th} VM to the j^{th} physical host mapping
x'_{ij}	Boolean value to determine the replication of the i^{th} VM to the j^{th} physical host mapping
y_j	Boolean value to determine usage of the physical host j
c_i	CPU usage of the i^{th} VM
c'_i	CPU usage of the i^{th} VM's replica
m_i	Memory usage of the i^{th} VM
m'_i	Memory usage of the i^{th} VM's replica
b_i	Network bandwidth usage of the i^{th} VM
b'_i	Network bandwidth usage of the i^{th} VM's replica
C_j	CPU capacity of the j^{th} physical host
M_j	Memory capacity of the j^{th} physical host
B_j	Network bandwidth of the j^{th} physical host

physical hosts by satisfying the requested resource requirements of VMs and their replicas. Constraints (2) and (3) ensure every VM and VM's replica is deployed in a physical host. Constraints (4), (5), (6) guarantee that the total capacity of CPU, memory, and network bandwidth of deployed VMs and VMs' replicas are packed into an assigned physical host, respectively. Constraint (7) checks that a VM and its replica is not deployed in the same physical host since the physical host may become a single point of failure, which must be prevented.

$$\text{minimize} \sum_{j=1}^{m} y_j \tag{1}$$

$$\text{subject to} \sum_{j=1}^{m} x_{ij} = 1 \qquad \forall i \tag{2}$$

$$\sum_{j=1}^{m} x'_{ij} = 1 \qquad \forall i \tag{3}$$

$$\sum_{i=1}^{n} c_i x_{ij} + \sum_{i=1}^{n} c'_i x'_{ij} \leq C_j y_j \qquad \forall j \tag{4}$$

$$\sum_{i=1}^{n} m_i x_{ij} + \sum_{i=1}^{n} m'_i x'_{ij} \leq M_j y_j \qquad \forall j \tag{5}$$

$$\sum_{i=1}^{n} b_i x_{ij} + \sum_{i=1}^{n} b'_i x'_{ij} \leq B_j y_j \qquad \forall j \tag{6}$$

$$\sum_{i=1}^{n} x_{ij} + \sum_{i=1}^{n} x'_{ij} = 1 \qquad \forall j \tag{7}$$

$$x_{ij} = \{0,1\},\ x'_{ij} = \{0,1\},\ y_j = \{0,1\} \tag{8}$$

5.4. Experiment 1: Measuring the Impact on Latency for Soft Real-time Applications

To validate our high-availability solution including the VM replica placement algorithm, we used the RTI DDS Connext latency benchmark. [2] RTI Connext is an implementation of the OMG DDS standard [9]. The RTI Connext benchmark comprises code to evaluate the latency of DDS applications, and the test code contains both the publisher and the subscriber.

Our purpose in using this benchmark was to validate the impact of our high- availability solution and replica VM placement decisions on the latency of DDS applications. For this purpose, the DDS application was deployed inside a VM. We compare the performance between an optimally placed VM

[2]For the experiment, our application is using DDS and is not to be confused with our DDS-based resource usage dissemination solution. In our solution, the DDS approach is part of the middleware whereas the application resides in a VM.

replica using our algorithm described in Section 5.3 and a potentially worse case scenario resulting from a randomly deployed VM. In the experiment, average latency and standard deviation of latency, which is a measure of the jitter, are compared for different settings of Remus and VM placement. Since a DDS application is a one directional flow from a publisher to a subscriber, the latency measurement is estimated as half of the round-trip time which is measured at a publisher. In each experimental run, 10,000 samples of stored data in the defined byte sizes in the table are sent from a publisher to a subscriber. We also compare the performance when no high-availability solution is used. The rationale is to gain insights into the overhead imposed by the high-availability solution.

Figure 5 shows how our Remus-based high-availability solution along with the effective VM placement affects latency of real-time applications. The measurements from the experimental results for the case of *Without Remus*, where VM is not replicated, shows consistent range of standard deviation and average of latency compared to the case of *Remus with Efficient Placement*. When Remus is used, average latency does not increase significantly, however, a higher fluctuation of latency is observed by measuring standard deviation values between both cases. From the results we can conclude that the state replication overhead from Remus incurs a wider range of latency fluctuations.

However, the key observation is that significantly wider range of latency fluctuations are observed in the standard deviation of latency in *Remus with Worst Case Placement*. On the contrary, the jitter is much more bounded using our placement algorithm. our framework guarantees that the appropriate number of VMs are deployed in physical machines by following the defined resource constraints so that contention for resources between VMs does not occur even though a VM or a physical machine has crashed. However, if a VM and its replica is randomly placed without any constraints, unexpected latency increases for applications running on the VM could occur. The resulting values of latency's standard deviation in *Remus with Worst Case Placement* demonstrate how the random VM placement negatively influences timeliness properties of applications.

5.5. Experiment 2: Validating Co-existence of High Availability Solutions

Often times the applications or their software platforms support their own fault-tolerance and high-availability solutions. The purpose of this experiment is to test whether it is possible for both our Remus-based high availability solution and the third party solution could co-exist.

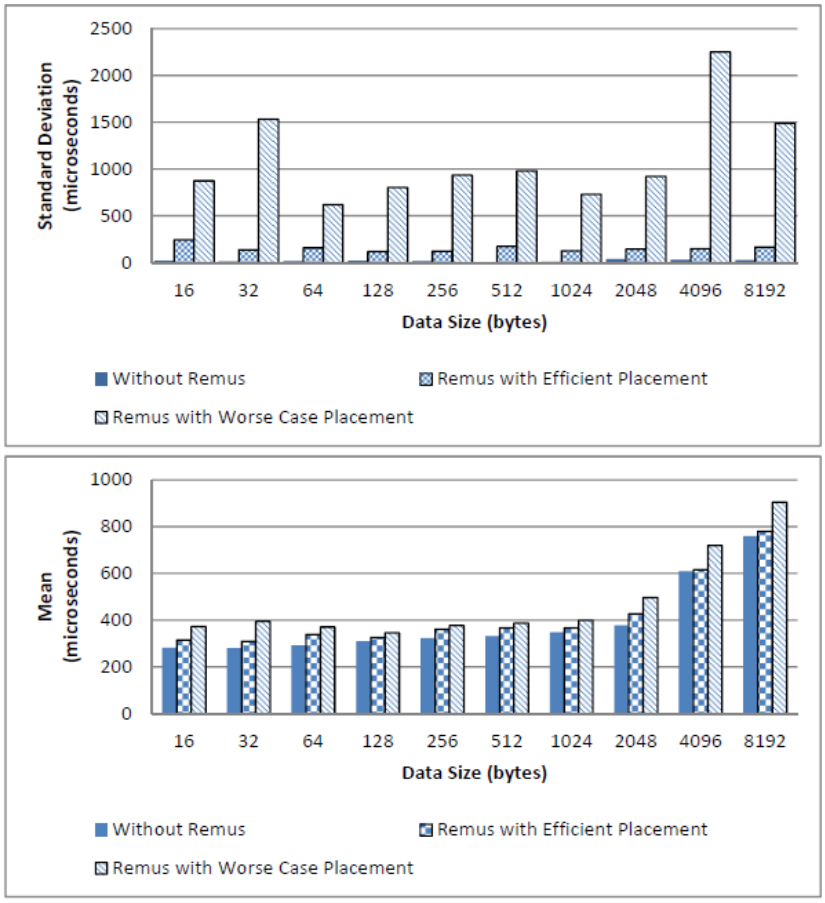

Figure 5: Latency Performance Test for Remus and Effective Placement

To ascertain these claims, we developed a *word count* example implemented in C++ using OMG DDS. The application supports its own fault tolerance using OMG DDS QoS configurations as follows. OMG DDS supports a QoS configuration called *Ownership Strength,* which can be used as a fault tolerance solution by a DDS pub/sub application. For example, the application can create redundant publishers in the form of multiple data writers that publish the same topic that a subscriber is interested in. Using the *OWNERSHIP_STRENGTH* configuration, the DDS application can dictate who the primary and backup publishers are. Thus, a subscriber receives the topics only from the publisher with the highest strength. When a failure occurs, a data reader (which is a DDS entity belonging to a subscriber) au-

tomatically fails over to receive its subscription from a data writer having the next highest strength among the replica data writers.

Although such a fault-tolerant solution can be realized using the ownership QoS, there is no equivalent method in DDS if a failure occurs at the source of events such as a node that aggregates multiple sensors data and a node reading a local file stream as a source of events. In other words, although the DDS ownership QoS takes care of replicating the data writers and organizing them according to the ownership strength, if these data writers are deployed in VMs of a cloud data center, they will benefit from the replica VM placement strategy provided by our approach thereby requiring the two solutions to co-exist.

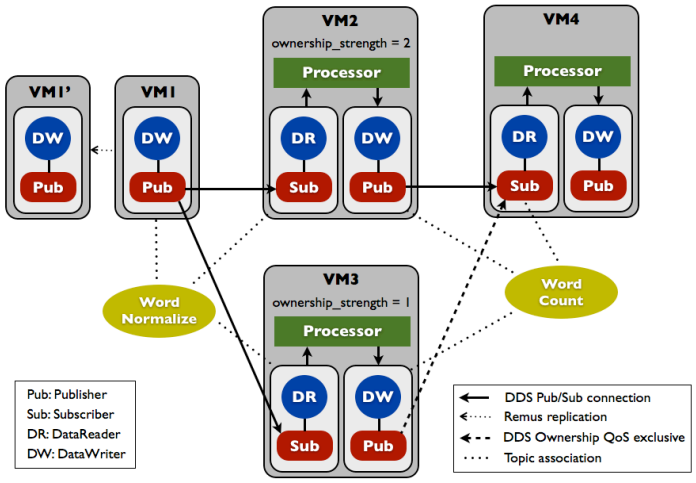

Figure 6: Example of Real-time Data Processing: Word Count

To experiment with such a scenario and examine the performance overhead as well as message missed ratio (*i.e.*, lost messages during failover), we developed a DDS-based "word count" real-time streaming application. The system integrates both the high availability solutions. Figure 6 shows the deployment of the *word count* application running on the highly available system. Four VMs are employed to execute the example application. VM1 runs a process to read input sentences and publishes a sentence to the next processes. We call the process running on the VM1 as the *WordReader*. In the next set of processes, a sentence is split into words. These processes are called *WordNormalizer*. We place two VMs for the normalizing process and

each data writer's Ownership QoS is configured with the exclusive connection to a data reader and the data writer in VM3 is set to the primary with higher strength. Once the sentences get split, words are published to the next process called the *WordCounter*, where finally the words are counted. In the example, we can duplicate processes for *WordNormalizer* and *WordCounter* as they process incoming events, but a process for *WordReader* cannot be replicated by having multiple data writers in different physical nodes as the process uses a local storage as a input source. In this case, our VM-based high availability solution is adopted.

Table 3: DDS QoS Configurations for the Word Count Example

DDS QoS Policy	Value
Data Reader	
Reliability	Reliable
History	Keep All
Ownership	Exclusive
Deadline	10 milliseconds
Data Writer	
Reliability	Reliable
Reliability - Max Blocking Time	5 seconds
History	Keep All
Resource Limits - Max Samples	32
Ownership	Exclusive
Deadline	10 milliseconds
RTPS Reliable Reader	
MIN Heartbeat Response Delay	0 seconds
MAX Heartbeat Response Delay	0 seconds
RTPS Reliable Writer	
Low Watermark	5
High Watermark	15
Heartbeat Period	10 milliseconds
Fast Heartbeat Period	10 milliseconds
Late Joiner Heartbeat Period	10 milliseconds
MIN NACK Response Delay	0 seconds
MIN Send Window Size	32
MAX Send Window Size	32

Table 3 describes the DDS QoS configurations used for our *word count* application. The throughput and latency of an application can be varied by different DDS QoS configurations. Therefore, our configurations in the table can provide a reasonable understanding of our performance of experiments described below. In the *word count* application, since consistent word counting information is critical, *reliable* rather than *best effort* is designated as the Reliability QoS. For reliable communication, history samples are all kept in the reader's and writer's queues. As the Ownership QoS is set to *exclu-*

sive, only one primary data writer among multiple data writers can publish samples to a data reader. If a sample has not arrived in 10 milliseconds, a deadline missing event occurs and the primary data writer is changed to the one which has the next highest ownership strength.

The results of experimental evaluations are presented to verify performance and failover overhead of our Remus-based solution in conjunction with DDS Ownership QoS. We experimented six cases shown in the Figure 7 to understand latency and failover overhead of running Remus and DDS Ownership QoS for the word count real-time application. The experimental cases represent the combinatorial fail over cases in an environment selectively exploiting Remus and DDS Ownership QoS.

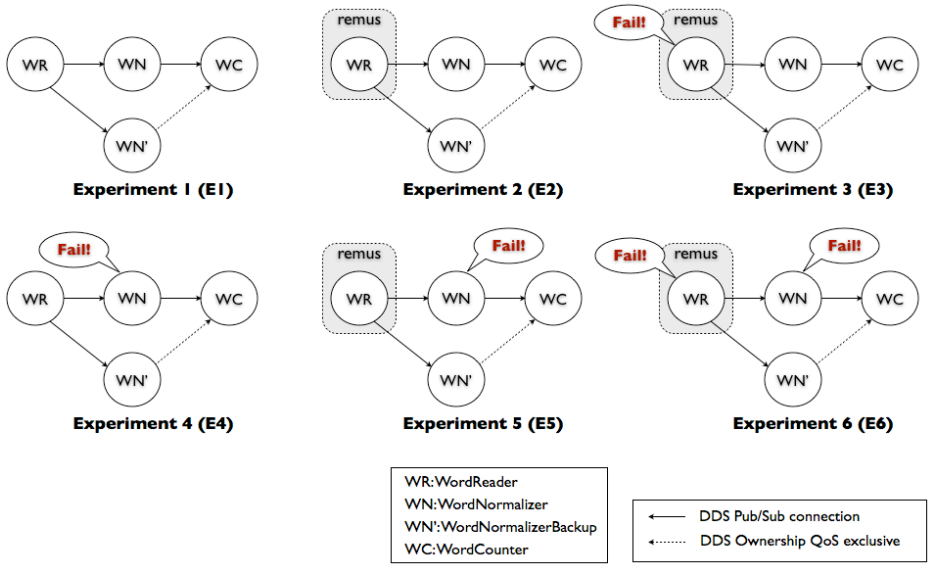

Figure 7: Experiments for the Case Study

Figure 8 depicts the results of Experiment E1 and E2 from Figure 7. Both the experiments have Ownership QoS setup as described above. Experiment E2 additionally has VM1 running the *WordReader* process, which is replicated to VM1' whose placement decision is made by our algorithm. The virtual machine VM1 is replicated using Remus high availability solution with the replication interval set to 40 milliseconds for all the experiments. This interval is also visibly the lowest possible latency for all the experiments, which has ongoing Remus replication. All the experiments depicted in Figure 7 involved a transfer of 8,000 samples from *WordReader* process on VM1

to *WordCount* process running on VM4. In the experiments E1 and E2, *WordNormalizer* processes run on VM2 and VM3 and incur the overhead of DDS Ownership QoS. In addition, experiment E2 has the overhead of Remus replication.

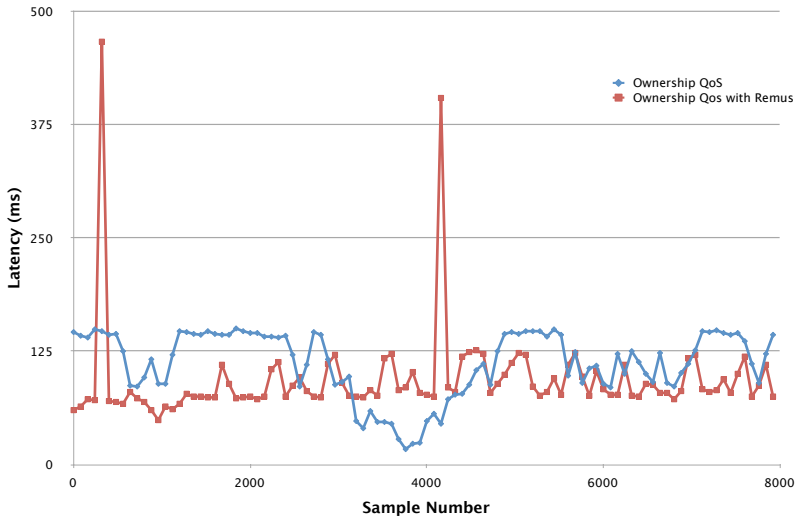

Figure 8: Latency Performance Impact of Remus Replication

The graph in Figure 8 is a plot of average latency for each of the 80 samples set for a total of 8,000 samples transfer. For experiment E1 with no Remus replication, it was observed that the latency fluctuated within a range depending upon the queue size of *WordCounter* and each of *WordNormalizer* processes. For experiment E2 with Remus replication, the average latency for sample transfer did not have much deviation except for a few jitters. This is because of the fact that Remus replicates at a stable, predefined rate (here 40 ms), however, due to network delays or delay in checkpoint commit, we observed jitters. These jitters can be avoided by setting stricter deadline policies in which case, some samples might get dropped and they might need to be resent. Hence, in case of no failure, there is very little overhead for this soft real-time application.

Figure 9 is the result for experiment E3 where *WordReader* process on VM1 is replicated using Remus and it experienced a failure condition. Before

the failure, it can be observed that the latencies were stable with few jitters due to the same reasons explained above. When the failure occurred, it took around 2 seconds for the failover to complete during which a few samples got lost. After the failover, no jitters were observed since Remus replication has not yet started for VM1', but the latency showed more variation as the system was still stabilizing from the last failure. Thus, the high availability solution works for real-time applications even though a minor perturbation is present during the failover.

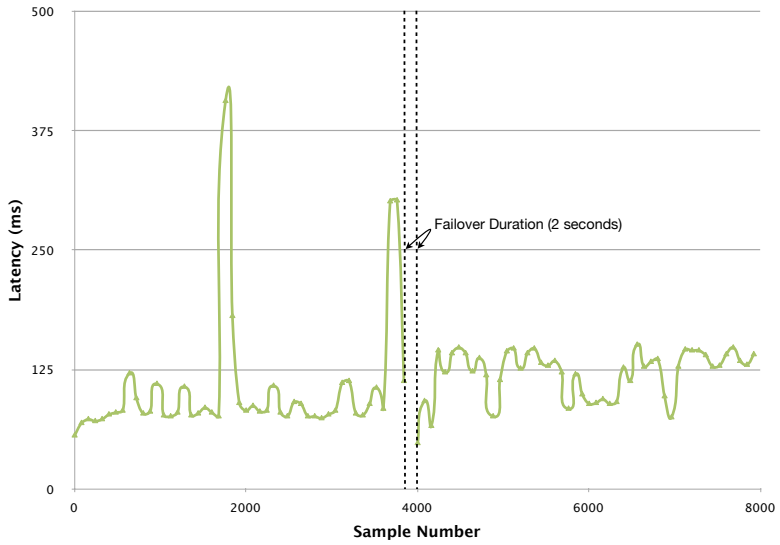

Figure 9: DDS Ownership QoS with Remus Failover

Table 4 represents the missed ratio for different failover experiments performed. In experiments E4 and E5, VM2 failed and the WordNormalizer process failed over to VM3. Since the DDS failover relied on publisher/subscriber mechanism, the number of lost samples is low. The presence of Remus replication process on WordReader process node VM1 did not have any adverse effect on the reliability of the system. However, in case of experiments E3 and E6, where Remus failover took place, the number of lost samples was higher since the failover duration is higher in case of Remus replication than DDS failover. These experiments show that ongoing Remus replication does not affect the performance of DDS failover, even though Remus failover is

slower than DDS failover. However, since DDS does not provide any high availability for the source, infrastructure-level high availability provided by our Remus-based solution must be used.

Table 4: Failover Impact on Sample Missed Ratio

	Missed Samples (total of 8000)	Missed Samples Percentage (%)
Experiment 3	221	2.76
Experiment 4	33	0.41
Experiment 5	14	0.18
Experiment 6	549	6.86

6. Conclusion

As real-time applications move to the cloud, it becomes important for cloud infrastructures and middleware to implement algorithms that provide the QoS properties (*e.g.*, timeliness, high-availability, reliability) of these applications. In turn this requires support for algorithms and mechanisms for effective fault-tolerance and assuring application response times while simultaneously utilizing resources optimally. Thus, the desired solutions require a combination of algorithms for managing and deploying replicas of virtual machines on which the real-time applications are deployed in a way that optimally utilizes resources, and algorithms that ensure timeliness and high availability requirements.

This paper presented the architectural details of a middleware framework for a fault-tolerant cloud computing infrastructure that can automatically deploy replicas of VMs according to flexible algorithms defined by users. Finding an optimal placement of VM replicas in data centers is an important problem to be resolved because it determines the QoS delivered to performance-sensitive applications running in the cloud. To that end this paper presents an instance of an online VM replica placement algorithm we have formulated as an ILP problem.

The work presented in this paper addresses just one dimension of a number of challenges that exist in supporting real-time application in the cloud. For example, scheduling of virtual machines (VMs) on the host operating system (OS) and in turn scheduling of applications on the guest OS of the

VM in a way that assures application response times is a key challenge that needs to be resolved. Scheduling alone is not sufficient; the resource allocation problem must be addressed wherein physical resources including CPU, memory, disk and network must be allocated to the VMs in a way that will ensure that application QoS properties are satisfied. In doing so, traditional solutions used for hard real-time systems based on over-provisioning are not feasible because the cloud is an inherently shared infrastructure, and operates on the utility computing model. Autoscaling algorithms used in current cloud computing platforms must be such that response times are not adversely impacted when resources are scaled up or down, and applications must be migrated.

The gamut of the problem space described above is vast. Addressing these needs forms the bulk of our future work. Our ongoing research is focusing on refining the presented architecture. Additionally, substantial validation of the solutions is necessary. To that end we are seeking to test a range of performance-sensitive applications hosted in the cloud. We are leveraging a private cloud testbed we have deployed at our institution where we have access to a variety of latest hardware and network switches, as well as a variety of open-source cloud infrastructure platforms, such as OpenStack and OpenNebula as well as hypervisors, such as Xen and KVM.

Even though we have experimented on a small private cloud, we believe our solution is scalable enough to deal with a large cloud environment. We surmise this on the basis that our underlying technology is Remus, which works between a pair of replicas. Hence, multiple instances of Remus will be active to deal with multiple such pairs of replicas. The impact of message exchanges on the network due to multiple independent Remus replicas needs to be investigated, however. We believe that traffic isolation solutions may be used to alleviate the impact on network bandwidth. The VM replica placement algorithm requires real-time monitoring, which is provided by DDS. Clearly, we cannot expect to use a single VM placement engine for a large data center nor can be use an entire data center as one single DDS domain within which the resource monitoring is performed. Rather, a large data center can be partitioned into multiple regions and have the DDS monitoring capability restricted to individual regions by using the concept of a DDS domain. Consequently, the DDS traffic is now limited to within individual domains and each region can have its own VM replica placement engine. Hierarchical solutions can also be built. To test our hypothesis, however, in future, we would need to work with cloud providers to gain access to large

clusters and validate the scalability of our solution.

References

[1] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, M. Zaharia, A View of Cloud Computing, Communications of the ACM 53 (4) (2010) 50–58.

[2] A. Corradi, L. Foschini, J. Povedano-Molina, J. Lopez-Soler, DDS-enabled Cloud Management Support for Fast Task Offloading, in: IEEE Symposium on Computers and Communications (ISCC '12), 2012, pp. 67–74. doi:10.1109/ISCC.2012.6249270.

[3] T. M. Takai, Cloud Computing Strategy, Tech. rep., Department of Defense Office of the Chief Information Officer (Jul. 2012).
URL `http://www.defense.gov/news/DoDCloudComputingStrategy.pdf`

[4] M. Chippa, S. M. Whalen, S. Sastry, F. Douglas, Goal-seeking Framework for Empowering Personalized Wellness Management, in: (POSTER) in Workshop on Medical Cyber Physical Systems, CPSWeek, April, 2013.

[5] S. Xi, J. Wilson, C. Lu, C. Gill, RT-Xen: Towards Real-time Hypervisor Scheduling in Xen, in: Proceedings of the Ninth ACM International Conference on Embedded Software, EMSOFT '11, ACM, New York, NY, USA, 2011, pp. 39–48. doi:10.1145/2038642.2038651.
URL `http://doi.acm.org/10.1145/2038642.2038651`

[6] B. Cully, G. Lefebvre, D. Meyer, M. Feeley, N. Hutchinson, A. Warfield, Remus: High Availability via Asynchronous Virtual Machine Replication, in: Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation, USENIX Association, 2008, pp. 161–174.

[7] J. Fontán, T. Vázquez, L. Gonzalez, R. Montero, I. Llorente, Opennebula: The open source virtual machine manager for cluster computing, in: Open Source Grid and Cluster Software Conference, 2008.

[8] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, A. Warfield, Xen and the Art of Virtualization, in: ACM SIGOPS Operating Systems Review, Vol. 37, ACM, 2003, pp. 164–177.

[9] Object Management Group, Data Distribution Service for Real-time Systems Specification, 1.2 Edition (Jan. 2007).

[10] D. J. Scales, M. Nelson, G. Venkitachalam, The design of a practical system for fault-tolerant virtual machines, Operating Systems Review 44 (4) (2010) 30–39.

[11] Y. Tamura, K. Sato, S. Kihara, S. Moriai, Kemari: Virtual machine synchronization for fault tolerance, In USENIX 2008 Poster Session.

[12] K.-Y. Hou, M. Uysal, A. Merchant, K. G. Shin, S. Singhal, Hydravm: Low-cost, transparent high availability for virtual machines, Tech. rep., HP Laboratories (2011).

[13] X. Zhang, E. Tune, R. Hagmann, R. Jnagal, V. Gokhale, J. Wilkes, Cpi2: Cpu performance isolation for shared compute clusters, in: Proceedings of the 8th ACM European Conference on Computer Systems, EuroSys '13, ACM, New York, NY, USA, 2013, pp. 379–391.

[14] X. Pu, L. Liu, Y. Mei, S. Sivathanu, Y. Koh, C. Pu, Understanding performance interference of i/o workload in virtualized cloud environments, in: Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on, IEEE, 2010, pp. 51–58.

[15] O. Tickoo, R. Iyer, R. Illikkal, D. Newell, Modeling virtual machine performance: challenges and approaches, ACM SIGMETRICS Performance Evaluation Review 37 (3) (2010) 55–60.

[16] R. Nathuji, A. Kansal, A. Ghaffarkhah, Q-clouds: managing performance interference effects for qos-aware clouds, in: Proceedings of the 5th European conference on Computer systems, ACM, 2010, pp. 237–250.

[17] C. Hyser, B. McKee, R. Gardner, B. Watson, Autonomic virtual machine placement in the data center, Hewlett Packard Laboratories, Tech. Rep. HPL-2007-189.

[18] S. Lee, R. Panigrahy, V. Prabhakaran, V. Ramasubrahmanian, K. Talwar, L. Uyeda, U. Wieder, Validating heuristics for virtual machines consolidation, Microsoft Research, MSR-TR-2011-9.

[19] A. Beloglazov, R. Buyya, Energy efficient allocation of virtual machines in cloud data centers, in: Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on, Ieee, 2010, pp. 577–578.

[20] M. Massie, B. Chun, D. Culler, The ganglia distributed monitoring system: design, implementation, and experience, Parallel Computing 30 (7) (2004) 817–840.

[21] W. Barth, Nagios: System and network monitoring, No Starch Pr, 2008.

[22] I. Foster, Y. Zhao, I. Raicu, S. Lu, Cloud computing and grid computing 360-degree compared, in: Grid Computing Environments Workshop, 2008. GCE'08, Ieee, 2008, pp. 1–10.

[23] C. Huang, P. Hobson, G. Taylor, P. Kyberd, A study of publish/subscribe systems for real-time grid monitoring, in: Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International, IEEE, 2007, pp. 1–8.

[24] M. García-Valls, I. Rodriguez-Lopez, L. Fernandez-Villar, iland: An enhanced middleware for real-time reconfiguration of service oriented distributed real-time systems.

[25] F. Han, J. Peng, W. Zhang, Q. Li, J. Li, Q. Jiang, Q. Yuan, Virtual resource monitoring in cloud computing, Journal of Shanghai University (English Edition) 15 (5) (2011) 381–385.

[26] S. De Chaves, R. Uriarte, C. Westphall, Toward an architecture for monitoring private clouds, Communications Magazine, IEEE 49 (12) (2011) 130–137.

[27] D. Guinard, V. Trifa, E. Wilde, A resource oriented architecture for the web of things, in: Internet of Things (IOT), 2010, IEEE, 2010, pp. 1–8.

[28] openstack.org (Sep. 2013).
URL `http://www.openstack.org`

[29] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, D. Zagorodnov, The Eucalyptus Open-source Cloud-computing System, in: Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, IEEE Computer Society, 2009, pp. 124–131.

[30] A. Kivity, Y. Kamay, D. Laor, U. Lublin, A. Liguori, kvm: the Linux virtual machine monitor, in: Proceedings of the Linux Symposium, Vol. 1, 2007, pp. 225–230.

[31] libvirt (Sep. 2013).
URL http://libvirt.org/

[32] P. T. Eugster, P. A. Felber, R. Guerraoui, A.-M. Kermarrec, The Many Faces of Publish/Subscribe, ACM Computer Survey 35 (2003) 114–131. doi:http://doi.acm.org/10.1145/857076.857078.
URL http://doi.acm.org/10.1145/857076.857078

[33] A. Corsaro, 10 reasons for choosing opensplice dds (2009).
URL http://www.slideshare.net/Angelo.Corsaro/10-reasons-for-choosing-opensplice-dds

[34] D. Schmidt, H. van't Hag, Addressing the challenges of mission-critical information management in next-generation net-centric pub/sub systems with opensplice dds, in: Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on, IEEE, 2008, pp. 1–8.

[35] A. Corsaro, L. Querzoni, S. Scipioni, S. Piergiovanni, A. Virgillito, Quality of service in publish/subscribe middleware, Global Data Management (2006) 1–19.

[36] E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, Reading, MA, 1995.

[37] J. Berkey, P. Wang, Two-dimensional finite bin-packing algorithms, Journal of the operational research society (1987) 423–429.

[38] RTI Connext DDS Performance Benchmarks - Latency and Throughput, http://www.rti.com/products/dds/benchmarks-cpp-linux.html.

Appendix A. Glossary

Integer Linear Programming (ILP) is a mathematical method to achieve the best outcome (lowest cost) where all the unknown variables are integers.

Continuous check-pointing is a high availability solution in which at frequent intervals of time, the execution of primary VM is paused to capture its state.

Lock step execution is a redundant execution paradigm where both the primary and secondary VMs execute the same set of instructions.

Deterministic replay is defined as reenactment of the program state from one VM to the other.

Speculative execution is an optimization technique in which primary VM continues to execute next set of instructions without waiting for the response from secondary VM. However, the results are discarded if any error occurs at the secondary VM.

Fail-stop failure is a failure condition where the failed host stops working and all associated data are lost.

Stable storage is a data storage technique where data is preserved even after host failure.