Recovery-Oriented Resource Management in Hybrid Cloud Environments

Yasser Aldwyan[1,2] and Richard O. Sinnott[1]
[1]*School of Computing and Information Systems, The University of Melbourne, Australia*
[2]*Department of Computer Science, Islamic University in Madinah, Saudi Arabia*

Keywords: Cloud Computing, Hybrid Cloud, Reliability, Recovery Oriented Computing (ROC), Fault Tolerance, Virtual Infrastructure Management, Resource Management.

Abstract: Cloud-based systems suffer from an increased risk of individual server failures due to their scale. When failures happen, resource utilization and system reliability can be negatively affected. Hybrid cloud models allow utilization of local resources in private clouds with resources from public clouds as and when needed through *cloudbursting*. There is an urgent need to develop cloudbursting approaches that are cognisant of the reliability and fault tolerance of external cloud environments. Recovery oriented computing (ROC) is a new approach for building reliable services that places emphasis on recovery from failures rather than avoiding them completely since even the most dependable systems will eventually fail. All fault tolerant techniques aim to reduce time to recover (TTR). In this paper, we develop a ROC-based fault tolerant approach for managing resources in hybrid clouds by proposing failure models with associated feedback control supporting a local resource-aware resource provisioning algorithm. We present a recovery-oriented virtual infrastructure management system (RVIMS). Results show that RVIMS is more reliable than those of single cloud environments even though TTR in the single cloud environments are about 10% less than those of RVIMS.

1 INTRODUCTION

Cloud computing has become an important paradigm in the field of Information Technology. It encompasses important aspects such as self-service, enhanced access to virtualized resources, such as compute, i.e. virtual machines (VMs) and storage resources, and on-demand capacity provisioning. While cloud computing has many benefits, its main advantage is the ability to provision and release resources based on workloads (Voorsluys et al., 2011). However, this ability poses a challenge when the traditional single cloud model is used and the data center overloaded. Hybrid cloud models can help to overcome such issues by making the cloud adaptive and allowing seamless utilization of resources of public clouds along with the local resources of private clouds.

As in other large-scale distributed systems, failures in cloud computing are unavoidable due to their scale (Javadi et al., 2012). When failures occur, the utilization of resources, reliability and availability can be adversely affected. Consider a web application running in a cloud. Figure 1 shows a typical architectural model of the application that consists of a load balancer running on a VM, (i.e., a front end tier) and a number of web servers deployed on other VMs in the cloud. The load balancer distributes the incoming (http) requests for web pages evenly across the back end servers, and the back end servers process the requests and send back the responses which can include web pages or other web resources. Now, assume in a given period, the load on these web servers increases dramatically. This could result in a failure in performance if the cloud management system was not aware of this increase and could not take appropriate action, e.g. by provisioning new VMs to run new web servers. This failure could cause the CPU utilization of the web servers to exceed predefined thresholds. The response time thus becomes high and thus the overall quality of service (QoS) will decrease. This situation exemplifies an undesirable form of utilization, i.e., overutilization of resources.

Furthermore, under-utilization of resources is another unwanted form of resource utilization. This situation occurs when the incoming requests decrease and the cloud management system does not

Aldwyan, Y. and Sinnott, R.
Recovery-Oriented Resource Management in Hybrid Cloud Environments.
DOI: 10.5220/0006359602530265
In *Proceedings of the 7th International Conference on Cloud Computing and Services Science (CLOSER 2017)*, pages 225-237
ISBN: 978-989-758-243-1

decrease the number of resources given to the cloud application. At any given time a VM crash can cause unavailability of the cloud application and affect the QoS. To overcome these issues, cloud management systems need to be fault tolerant and have the ability to provide cloud applications with appropriate features, such as auto-scaling and load balancing, across multiple clouds. This requires the cloud management system supports fault tolerant techniques in hybrid cloud environments.

Most existing fault tolerant approaches attempt to predict failures and avoid them before they occur. However, failures will inevitably happen. Thus, in our work we attempt to detect failures and recover from them as rapidly as possible i.e. our proposed fault tolerant techniques put effort to reduce time to recover (TTR). The *aim of our research* is to develop an approach for enabling fault tolerant techniques in hybrid cloud environments that will enable those environments to be recovery-oriented, adaptive and self-managed in order to optimize the utilization of resources; improve availability and reliability and minimize human interventions.

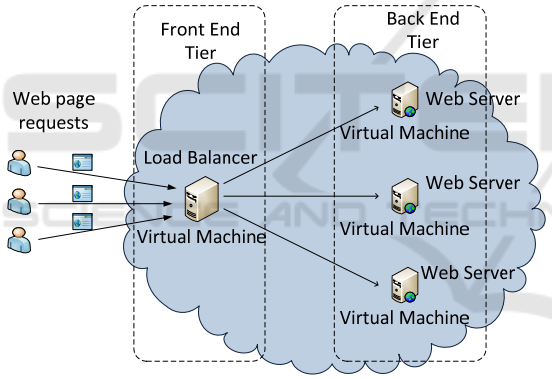

Figure 1: Example architectural model of cloud-based web application.

We adopt a recovery-oriented computing approach (ROC) (Berkeley, 2004). Specifically, we investigate failure models for hybrid clouds. To support this, we first explore failure models by identifying failures and their characteristics that can impact on TTR. We then develop a feedback control system. To support this, we propose a system model based on a control theory (Yixin et al., 2005), which provides a number of mechanisms for designing automated self-managing computing models. The model periodically monitors the health of resources, detects failures and recovers from failures. We use this to establish a recovery-oriented virtual infrastructure management system (RVIMS). We apply RVIMS in a hybrid cloud environment and show how it can subsequently be used to manage cloud services in fault tolerant hybrid cloud environments.

The rest of the paper is organized as follows. Section 2 presents background and an overview of cloud computing and ROC with focus on feedback control models for self-managing computing systems and virtual infrastructure management. In Section 3 we present failure models for hybrid clouds. Then in Section 4, the feedback control model is proposed, including a local resource aware hybrid cloud provisioning algorithm. In Section 5, the architecture of RVIMS is presented. The experiments and results are presented in Section 6. Finally, conclusions and future work are presented in Section 7.

2 BACKGROUND AND RELATED WORK

This section aims to provide an overview of cloud computing with particular focus on hybrid cloud models. We introduce relevant concepts and services and provide a brief overview of ROC; the control theory for self-managing computing systems, and virtual infrastructure management systems for hybrid clouds.

Cloud Computing is based on two main technologies: service oriented architectures (SOA) and virtualization technology (El-Refaey, 2011). SOA are an architectural model in which everything should be provided as a service, including processing power, networks, storage, IT infrastructure, software, hardware and other IT resources. El-Refaey (El-Refaey, 2011) defines virtualization technology as technology that provides an abstraction of computing resources: examples include CPUs, memory, storage and networks. This has led to the division of physical servers into multiple virtual machines (VMs). This technology is significant in cloud computing because it facilitates the management of resources and improves the utilization of those resources.

A **Hybrid Cloud**, also known as a Multi-Cloud in related literature (Grozev and Buyya, 2014) is a combination of two or more different cloud infrastructures: private, public or community. A major benefit of the hybrid cloud model is that it takes the attributes of both public and private clouds and combines them into a unified, automated, and well-managed cloud computing offering. A hybrid cloud takes advantage of a public cloud's scalability

and cost-effectiveness while also providing the control and high performance available in a private cloud. However, utilizing resources from both models in an optimized way is a major issue in hybrid clouds, and to ensure a minimum level of quality of service (QoS), providers must leverage strategies that fulfil potentially diverse QoS requirements.

Resource Provisioning is used for exercising control over VMs or other cloud resources in cloud systems. This is often used for launching, suspending and terminating VMs. A hybrid cloud resource provisioning service us used for provisioning resources from different clouds. For instance, if a user requests three VMs in a hybrid cloud, the resource provisioning can launch one VM from the private cloud and the other two from the public cloud. Many factors can be considered when provisioning resources in hybrid clouds including local resource awareness. Local resource awareness factors allows resource provisioning services to first launch local resources, e.g. resources from private clouds, and then, when local resources are at capacity, launch resources on public resources (Grozev and Buyya, 2014). This improves scalability however it incurs additional complexity and potential monetary cost.

Recovery-Oriented Computing (ROC) is an approach developed by Berkeley and Stanford for investigating innovative strategies and techniques for building highly-dependable Internet services (Berkeley, 2004). ROC places emphasis on recovery from failures rather than avoiding failures. The motivation behind this approach is that even long-lasting and healthy systems will periodically face failures. There are three assumptions considered in the ROC approach: Software and hardware will definitely fail; not all failures can be predicted in advance, and individuals can/do make mistakes.

Applying the ROC approach helps a system designer change their way of thinking from paying attention to failure avoidance to paying attention to reducing the time needed to recover from a failure. This shift of thinking can help create more robust cloud platforms (Microsoft, 2014a). In (Microsoft, 2014a), the authors propose an approach to design reliable cloud services based on ROC. They introduce mean time to recover (MTTR) (or only time to recover [TTR] (Microsoft, 2014b)) which is the time needed to re-establish a service after a failure. Minimizing TTR requires a system to be recovered to a fully functional state as quickly as possible.

A hybrid cloud model helps in overcoming scalability and availability issues. A better approach is to make such adaptations happen automatically (Tanenbaum and Steen, 2006). This is often known as autonomic computing or self-managing systems, such as IBM's Autonomic Computing and Microsoft's Dynamic Systems Initiative. A main goal of these systems is to minimize the costs of operation by increasing automation, i.e. making systems self-managing without any human interaction (Yixin et al., 2005). Making cloud systems self-managing allows them to recover from failures quickly and, subsequently the TTR can be reduced.

To make automatic adaptations, monitoring and adjustments of Cloud systems is required. One way to achieve this is to organize systems to include high-level feedback and control systems. These systems are typically based on **control theory** which gives a valuable set of methods for building self-diagnosis, self-repairing, self-healing, self-optimizing, self-configuring and ultimately self managing computing systems (Yixin et al., 2005, Tanenbaum and Steen, 2006).

A system that manages virtualized resources is known as a **Virtual Infrastructure Management System (VIMS)** or a virtual infrastructure (VI) manager (Sotomayor et al., 2009). When designing and implementing hybrid (or private) clouds, Sotomayor el al. (Sotomayor et al., 2009) outlined several features of public clouds that must be considered: a hybrid cloud must provide a consistent, identical, homogeneous view of all virtualized resources without consideration of the virtualization technology, e.g. Xen or VMware. It must have control over the entire lifecycle of VMs, such as VM disk image and software deployment. It must be adaptive to meet dynamic needs for resources, such as peak times where resources are not sufficient for the current demand. Resource provisioning in hybrid clouds must also be configurable to different policies in order to meet the systems' requirements such as server consolidation to save power and/or cost optimization or support high availability demands.

OpenNebula (OpenNebula, 2016) is an example of a VIMS. In our work, we propose a recovery oriented virtual infrastructure management system (RVIMS), which employs the proposed failure and feedback control models in hybrid cloud environments.

2.1 Related Work

A standard model of Cloud computing (i.e. a single

cloud) poses a number of challenges (Grozev and Buyya, 2014). In terms of availability and reliability, a data center outage can cause mass service unavailability and all cloud clients will not be able to access cloud resources (NIST, 2011, Armbrust et al., 2009, Laing, 2012, Google, 2010). Another challenge is scalability. This occurs when the cloud is overloaded. Hybrid clouds as a kind of Multi-Cloud (Grozev and Buyya, 2014) overcome these issues by making the cloud adaptive and utilize cloud resources from external public clouds.

Considerable work has been done in the development of hybrid cloud open-source libraries, e.g. Apache LibCloud (Libcloud, 2009). These libraries provide a unified API for managing and deploying cloud resources, such as VMs and storage (Grozev and Buyya, 2014). However, they are not concerned with resource provisioning. Likewise, cross-cloud management services, such as RightScale (RightScale, 2006) only offer (unified) user interfaces and tools for managing different clouds without implementing resource provisioning. In terms of application deployment, projects like Contrail (Cascella et al., 2012) aim at deploying applications in hybrid cloud environments. However, they only deal with provisioning and set-up and do not consider the distribution of workload and autoscaling of applications.

With regards to resource provisioning, Javadi et al. (Javadi et al., 2012) propose hybrid cloud resource provisioning policies in the presence of resource failures. These policies only consider resource failure correlations when redirecting user requests for resources to suitable cloud providers and not during deployment of VMs for user requests. Furthermore, in (Mattess et al., 2013), the authors propose a dynamic provisioning algorithm of MapReduce applications across hybrid clouds, however this does not handle failures. In contrast to others, our hybrid cloud resource provisioning approach considers: failures that may occur during the whole lifecycle of cloud applications; recovery mechanisms based on recovery-oriented computing (ROC) (Berkeley, 2004); local resource awareness issues (Grozev and Buyya, 2014) to reduce the cost, and offers a multi-tier architectural model suited for web-based applications.

Significant efforts have been made in the development of virtual infrastructure management systems (Sotomayor et al., 2009). These kind of management systems are often called Multi-Cloud services when they support multiple clouds (Grozev and Buyya, 2014). Amazon Elastic Compute Cloud (Amazon EC2) (Amazon, 2016) and Google Compute Engine (Google, 2016) provide single-cloud VIMS. On the other hand, Eucalyptus provides a VIMS across hybrid clouds, however the clouds have to be compatible with Amazon Web Services (AWS) (Eucalyptus, 2008). They also lack fault tolerant techniques to make them more reliable and resilient. In contrast, we propose a recovery-oriented virtual infrastructure management system (RVIMS) suitable for hybrid cloud environments that is failure-aware and leverages control theory. This offers self-managing features for systems (Yixin et al., 2005). We also facilitate the process of adding new clouds to the system. Lastly, we develop a vendor-independent cloud agent that provides RVIMS with feedback messages to monitor resources across multiple clouds.

3 FAILURE MODELS FOR HYBRID CLOUDS

In this section, we explore failure models for hybrid cloud systems. We identify possible failures and their characteristics and potential recovery solutions. Our failure models are adapted from Resilience Modeling and Analysis (RMA), i.e. an approach for improving resilience at Microsoft (Microsoft, 2014b). This approach adopts the main ideas behind ROC, i.e. failures will eventually occur and thus it is necessary to try to reduce TTR to minimize the impacts of such failures. There are six types of failures in the proposed models: *full private VM pool capacity, cloud outage, VM crash, VM slowdown, VM high load failures and VM low load failures* as illustrated in Table 1.

Table 1: Failures in hybrid clouds.

Failure	**Recovery**	
	Private Cloud Solution	Hybrid Cloud Solution
Full Private VM Pool Capacity	Request rejected	Launch VMs on public clouds
Cloud Outage	No solution	Launch VMs on healthy clouds
VM Crash	Launch a VM on private cloud; or reject	Launch new VM on private or public clouds
VM Slowdown	Launch a VM on private cloud; or reject.	Launch new VM on private or public clouds
VM High Load	Launch a VM to distribute the load	Launch a VM to distribute the load
VM Low Load	Decrease the running VMs	Decrease the number of running VMs

A **Full Private VM Pool Capacity Failure** occurs when the resources of a private cloud are fully utilized, i.e. when VMs are allocated to other cloud applications. Such a failure arises when the infrastructure of the private cloud is overloaded. Such a situation impacts both the cloud provider and cloud application providers. The former will not be able to meet one of its core needs, scalability while the latter will find that their needs are unmet.

To detect a full private VM pool capacity failure, it is necessary to monitor the number of idle VMs in the cloud. If no idle VMs in the private cloud are available (or a limited number) then mitigating steps should be taken. In terms of recovery, when only a private cloud is used, there is no solution for recovery from this type of failure. The cloud provider will often simply reject any request for VMs needed for new cloud applications until VMs become available. The cloud provider can solve this issue by scaling the infrastructure out (i.e. adding new physical resources), but this solution can cause optimization issues in the longer term, i.e. the infrastructure may subsequently be underutilized. On the other hand, a hybrid cloud solution is more efficient in terms of time and resource utilization. Cloud providers can scale their cloud infrastructure dynamically based on demand. This provides an opportunity to scale up and down based on need, so the utilization of resources can be optimized.

A **Cloud Outage Failure** has been known since the emergence of cloud computing (Google, 2010, Laing, 2012). In this failure, cloud application providers and users cannot access services and applications. The cause of this failure varies. It can be a network partition of the data center, an outage of the power supply or even a bug in cloud infrastructure software. This failure can have a major impact on cloud providers and end users because all running services in the data center become effectively unavailable. There are many detection mechanisms that can be used here, e.g. pinging where a dummy message is sent to a suspected machine and a reply expected. Recovering from this failure is a challenge. There is no solution in a single cloud model (i.e. private cloud), however, a hybrid cloud model can address it to some extent, by launching VMs from healthy clouds, or at least mitigate its impact on cloud applications and overall cloud systems (Grozev and Buyya, 2014). This cannot be guaranteed to be autonomously supported however. Thus if a private cloud experiences a total outage, then an automated process to launch new VMs on the public cloud via redirecting request from the private cloud may be impossible.

A **VM Crash Failure** can be caused by hardware failure, a virtual machine monitor issue (VMM), an operating system issue or indeed an application software issue. The impact of this failure is downtime of the VM and the inaccessibility of the cloud applications running on the VM. This can have major issues, especially when a cloud application is running on only the impacted VM. The situation is less risky when the application is running on two or more VMs, e.g. as shown in Figure 1 with a web application running on back end servers with a load balancer running at the front end. Like all distributed systems, detecting a VM failure in cloud computing is non-trivial (Tanenbaum and Steen, 2006). This is because, even if the suspected VM is running (apparently) healthily, there may be other issues such as network partitions or test messages getting lost due to network issues. As with cloud outage failures, the mechanism that can be used to detect a VM crash can be as simple as pinging.

Recovery from this failure in the private cloud solution can be achieved by launching a new VM from the single private cloud. However, the request for a new VM may be rejected if there are no available VM resources in the private resource pool. If the cloud application is running on only one VM, this will make the application unavailable for potentially unpredictable periods. In contrast, in a hybrid cloud solution this situation can be avoided if the system is able to launch new VMs to the public cloud. In this case, the amount of downtime is determined by how long it takes to launch a new VM and install and start the cloud application. Knowing the temporal thresholds for such re-establishment is a key aspect of TTR.

A **VM Slowdown Failure** is less problematic type of failure than other failure types because the application is still running and can respond, although the response time may be relatively high. The cause of this failure can be due to other VM issues. Another cause may be input/ output (I/O) sharing among multiple VMs running on a physical machine. In (Armbrust et al., 2009), Armbrust et al. introduce I/O sharing as an obstacle for cloud computing that can unpredictably affect the overall system performance. They claim that sharing CPUs and memory among different VMs results in improved performance in cloud computing but that I/O sharing is a problem.

The effects of a VM slowdown failure on cloud applications can include a delay in handling requests and QoS subsequent decrease. There are two possible methods for detecting a VM slowdown

failure. One is when a response time exceeds a predefined threshold. The other is when the number of requests per second exceeds a given threshold, i.e. the cloud application responds after a delay. Regarding failure recovery, the private cloud and the hybrid cloud solutions are similar to solutions involving the recovery mechanism of a VM crash failure. The only difference is that the failed VM will continue to serve, albeit with lower QoS, until the new VM is ready to use.

A **VM High Load Failure** occurs when the demands on a cloud application increase and consequently the load on the VM increases and it eventually becomes over-utilized. The cause of this failure is related to high demands on the cloud application itself, e.g. if it becomes very popular or the business running it offers a temporary discounted price on the offered services. For an application to be ready for unexpected bursts, it needs to be scalable dynamically and automatically. The impact of this failure is on the QoS of the cloud application and higher response times.

Detecting VM high load failures can be achieved by monitoring CPU utilization, main memory utilization and network traffic. A policy including a set of thresholds for each VM (e.g. CPU and memory) should be provided before launching the application in the cloud. A failure occurs when a resource exceeds a predefined threshold. To recover from this failure in the private cloud, a new VM can be launched in order to distribute the workload evenly on all running VMs for the cloud application. However, this will be problematic if the private cloud is overloaded. Hybrid cloud systems can overcome this issue by launching a new VM on a public cloud and thus distributing the load to VMs across multiple clouds.

In contrast to the previous failures a **VM Low Load Failure** can introduce other undesirable forms of resource utilization. This type of failure is caused when the cloud application encounters lower demands. As result, the resources will be underutilized. The failure detection mechanism for this failure is similar to the one for the VM high load failure. There is a need for lower load thresholds for cloud application resources. The failure happens when the use of a monitored VM resource is below a predefined threshold. In terms of failure recovery, there is only one solution, and it can be applied in either a private cloud or a hybrid cloud. This solution is decreasing the number of running VMs for the cloud application.

4 FEEDBACK CONTROL MODEL FOR HYBRID CLOUDS

To detect failures and recover from them rapidly to reduce TTR, there is an urgent need for a fault tolerant system model based on the failure models proposed in the previous section. As consequence, we propose a self-managed feedback control model in a hybrid cloud environment by organizing components in a way that enables monitoring resources and taking appropriate action in the presence of failures. We describe the components of the model and propose local resource-aware hybrid cloud provisioning, failure detection and failure recovery algorithms.

4.1 Components of Feedback Control Model

As shown in Figure 2, the proposed model consists of six components: the *cloud interface* component, *cloud services and policies* components, *provisioner* component, *monitoring* component and *fault tolerance* components.

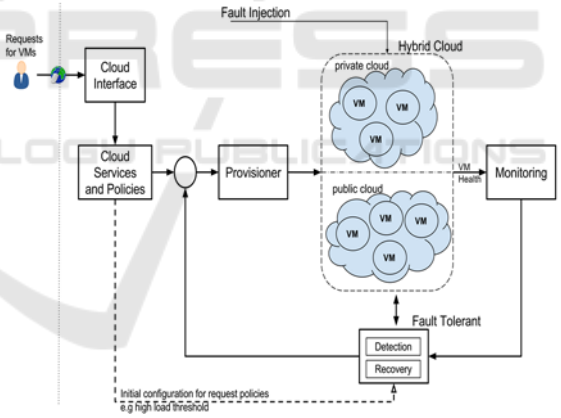

Figure 2: Feedback control model for hybrid clouds.

The **Cloud Interface Component** is an entry point for cloud users (e.g. SaaS/cloud application providers) to request cloud services upon which to deploy their cloud applications. This component receives requests for cloud services. After receiving requests, the cloud interface component passes them to the cloud services and policy component. Then, the cloud interface component waits until it receives a response from other components as to whether the cloud platform is able to handle a given request by provisioning the needed resources. If not, the request is rejected. In both cases, the response is forwarded to the cloud users.

The **Cloud Services and Policies Component** is responsible for creating and initializing appropriate cloud services and policy objects based on user requests and then passing them to the provisioner component for deployment. Cloud services can be VMs used to deploy cloud applications, load balancers used to distribute workloads across multiple back end VMs or autoscaling services used to scale up or down based on peak usage times. Cloud policies are a set of rules or conditions that help support failure detection and failure recovery and to take appropriate action when one or more conditions are met.

The **Provisioner Component** controls private and public resource pools in the hybrid cloud platform. It is responsible for provisioning VMs and other cloud resources on private and public clouds. This component provides a hybrid cloud provisioning service using the provisioning algorithm (see Algorithm 1). This service has a number of advantages for the proposed model, including providing awareness of local resources and thus reducing monetary costs (Grozev and Buyya, 2014). Furthermore, it has the ability to easily add more resource pools either from public or private clouds. Another benefit of this service is that it provides a higher level of abstraction by hiding the lower level communications and their implementation across clouds.

Algorithm 1: Local Resource Aware Hybrid Cloud Provisioning.

```
input: nReqVMs  //Number of VMs for a request
// VMs in unavailable clouds will not be considered
nIdleVMs ← getTotalHybridCloudIdleVMs();
listPrivateIdleVMs ← empty list;
listPublicIdleVMs ← empty list;
if nReqVMs ≤ nIdleVMs then
  nPrivateIdleVMs ←
                    getTotalPrivatCloudIdleVMs();
  if nPrivateIdleVMs ≥ 0 then
    // add idle VMs from private pool
    listPrivateIdleVMs.add(privateIdleVMs)
  nRemainingVMs ← nReqVMs –
                    length of listPrivateIdleVMs
  if nRemainingVMs > 0 then
    // add idle VMs from public pool
    listPublicIdleVMs.add(publicIdleVMs)
  if listPrivateIdleVMs is not empty then
    // Launch VMs from private pool
    launchVM( listPrivateIdleVMs)
  if listPublicIdleVMs is not empty then
    // Launch VMs from public pool
    launchVM(listPublicIdleVMs)
else
  reject the request
```

The provisioner component manages the whole life cycle of VMs in the model. Firstly it launches, suspends, migrates and terminates VM instances across multiple clouds. Secondly it runs customization scripts. Thirdly it installs cloud agent software on the top of VMs, to allow the cloud management system to monitor the health of the hybrid cloud resources. The provisioner component is also able to migrate VM instances from public clouds to private ones whenever VMs in the private cloud become free thereby reducing cost.

With regards to the local resource-aware hybrid cloud-provisioning algorithm shown in Algorithm 1, the primary parameter is the number of VMs required for user requests. The algorithm checks whether the total number of idle VMs in the hybrid cloud system is sufficient for the request. If the resources are not sufficient, then the request will be rejected and the algorithm will exit. Otherwise, the algorithm will first attempt to provision VMs from the resource pool of the private cloud. If there are not enough VMs at that time, the algorithm will provision VMs from public pools of available public clouds.

The **Monitoring Component** is designed to monitor VMs in both the private and public clouds. Each cloud agent running in a VM checks the health of the VM and periodically sends health messages to the monitoring component. This component listens on a (predefined) monitoring port and receives the VM health messages. It extracts the health information (e.g. CPU load) and sends it to the failure detection component so that any failures can be detected as early as possible.

The **Failure Detection Component** detects failures that occur during the lifetime of cloud applications running on VMs. It can detect the occurrence of a failure based on the VM health information coming from the monitoring component or on information obtained by direct communications with VMs (e.g. pinging and measuring response times). The failure detection component detects failures using two approaches: message based failure detection methods (MFDM) and direct failure detection methods (DFDM). In the MFDM method, the component receives a health message from a running VM, extracts the health information for resources from that VM and then compares this information with predefined thresholds provided by cloud policies that are received initially from the cloud services and policies component. These policies can vary from one cloud application to another based on user requests for the specific cloud services. For DFDM,

the failure detection component continuously (periodically) pings all running VMs across clouds in order to detect their liveness. If there is a reply, then the receiving VM is awake and healthy. If there is no reply, the cloud manager repeats the request to make sure the problem is not related to a sporadic network issue. If there is no reply after three attempts, the failure detector component detects a VM crash failure. Every time a VM crash failure is detected, the detector checks if other VMs in the resource pool have crashed, and if so a cloud outage failure is flagged and the resource pool of that cloud is marked as unavailable. This helps the provisioner to use live and available clouds only.

Another form of DFDM is a slowdown detection. In this form, the failure detection component periodically sends test messages to all VMs running a cloud application and measures their response time. If the response time is more than the predefined threshold in the policy of the cloud application, then a VM slowdown failure is identified. When such a failure occurs, the failure detection component puts a failure message into the failure recovery component's message queue, prompting the component to take appropriate action to recover from the failure as soon as possible.

The **Failure Recovery Component** reads failure messages in its message queue and handles the failure accordingly. The means of recovering from failures depends on the nature of failure and the cloud policy and autoscaling service associated with the cloud services and policies component of the cloud application. This component is used to reduce TTR and thus enhance the utilization of resources.

The input parameter of this component's algorithm is a failure message. The algorithm checks the failed VM object and the autoscaling service in which the VM is involved. After that, the algorithm takes the appropriate action based on the type of failure. If the failure is a VM crash failure, then the algorithm will unregister the failed VM from the load balancer, terminate the failed VM, launch a new VM and, finally, register the new VM with the load balancer. If the failure is a VM slowdown failure, then a new VM will be launched and registered with the load balancer and, finally, the failed VM will be released. If the failure is a VM high load failure, then the algorithm will launch and register a new VM with the load balancer. If the failure is a VM low load failure, the cloud platform will look first for a VM running on the public cloud to be unregistered from the load balancer and then released. Otherwise, any private VM for the cloud application will be chosen. The reason for doing this is to reduce the monetary cost since public VMs are typically not free. We note that this algorithm recovers from failures occurring during the lifetime of the cloud application, while the failures occurring before the deployment of the cloud application are handled implicitly by the provisioner component.

5 SYSTEM ARCHITECTURE

Employing the previously mentioned models in hybrid clouds requires the realization of a cloud management system. A central goal of this kind of system is to provide a high level of abstraction with which to facilitate the process of managing heterogeneous resources across different clouds including monitoring resources in VMs across different clouds. Ideally it should be possible to add new public clouds easily. In this section, we present a system architecture and implementation of a fault tolerant cloud management system for hybrid cloud environments supporting a **Recovery-Oriented Virtual Infrastructure Management System (RVIMS)**.

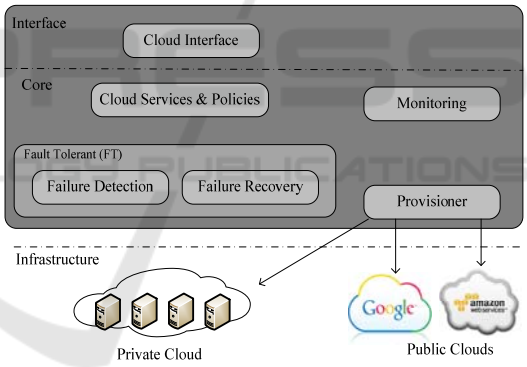

Figure 3: Architecture of RVIMS.

RVIMS has a number of key features. First, it deploys cloud services, such as creating instances of VMs and volumes, in different cloud platforms. At present the system supports Google Compute Engine (Google, 2016) and Amazon (AWS) (Amazon, 2016) clouds and the Australian National eResearch Collaboration Tools and Resources (Nectar) research cloud (Nectar, 2016). Secondly, it automates the process of installing cloud applications and their dependencies on VMs. Thirdly, it supports OpenStack based cloud platforms (e.g. Nectar) (OpenStack, 2010). Fourthly, it can easily support new cloud platforms. Essentially, it can detect and recover from all failures previously given in Table 1.

As shown in Figure 3, the architecture of RVIMS

consists of the same components found in the feedback control model section. RVIMS offers a cloud interface component, which acts as an entry point for cloud application providers. In the core part, we have four components: the cloud services and policies, monitoring, provisioner, failure detection, and failure recovery components. As noted, the provisioner component is responsible for providing higher levels of abstraction for managing heterogeneous resources more readily.

Figure 4 depicts the interaction between RVIMS and a VM instance. To understand the RVIMS and VM interactions, we consider the layered architecture of a VM. The first layer from the bottom is the operating system (OS) layer. The OS is chosen during the process of launching a VM. The layer above the OS layer is a cloud agent layer. This layer allows management and monitoring of running VMs regardless of the cloud platform from which they were launched. To achieve this, the cloud agent has to be in the application layer. On top of the cloud agent layer, we have the cloud application itself. With regard to the interaction between RVIMS and a VM, the cloud agent monitors VM resources (i.e., CPU, memory, hard disk and network traffic) and captures the behaviour of the cloud application. It periodically sends a health message containing this information to RVIMS. RVIMS reads the information, detects failures if they have occurred or failures that might subsequently arise and subsequently resolves them.

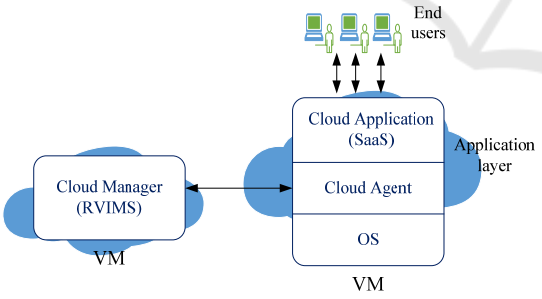

Figure 4: Interaction between RVIMS and a VM instance.

6 EXPERIMENTS AND RESULTS

This section aims to evaluate RVIMS by considering a recovery solution in a single (private) cloud as the baseline and then applying recovery solutions in two different hybrid cloud environments. The primary evaluation metric here is TTR, a measurement that starts when the system detects a failure and ends when the failure has been successfully resolved. Note that this work focuses on independent web services and inter-process communications between more complex applications and their deployment management across hybrid clouds remains an area for future work.

6.1 Experimental Testbed and Sample Application

The RVIMS system was deployed in a hybrid cloud environment consisting of three resource pools from different cloud providers: Nectar, Amazon and Google clouds. Nectar was used as a private cloud and the others were treated as public clouds. The private cloud was composed of four VMs, each of which was an *m1.medium* instance with two virtual CPUs (VCPUs) and 8 GB of RAM. Amazon cloud as a public cloud provider had a *t2.large* VM with two VCPUs and 8 GB of RAM. The other public cloud resource, which was provisioned was from Google cloud which included an *n1-standard-2* instance with two VCPUs and 7.5 GB RAM. Nectar instances were located in Melbourne, Australia. The Amazon instance was in Sydney, Australia, and the Google cloud was in Changhua County, Taiwan.

One VM instance in the private cloud was running the RVIMS while the others were used to deploy a sample application. The sample application for this evaluation was a simple two-tier web server. The first tier consisted of a load balancer running on a private VM instance, used to balance the load related to incoming http requests. The second tier was composed of back-end web servers running VM instances in different clouds.

A number of configuration parameters in the experiments were used to cause deliberate failures. This allowed us to observe the proposed system reactions in the presence of different kinds of failures. One configuration parameter was used to shut down a back-end server running on a VM instance. Another parameter slowed down the processing procedures of http requests in a back-end server by adding a sleep statement. Another configuration parameter increased the load on a VM instance by causing CPU utilization to exceed the threshold specified in the user request for VMs. The final configuration parameter acted to decrease the load on a VM instance.

User requests for cloud resources consisted of a number of VMs needed to run a cloud application. In our experiment, we limit the user request to a maximum of 3 VMs where one VM runs as a load balancer while the others run as back-end web servers. In all tests, all resources were provisioned

initially from the private cloud. When a failure was injected, there were three ways of handling it (see Table 2). The first is the private cloud solution, which handles failures in the private cloud only. The second is through the hybrid cloud (using the public cloud Amazon) for handling failures in the Amazon cloud. The third is through the cloud (using the public cloud Google), which handles failures in Google public cloud. In the second and the third recovery solutions, we assume the private cloud is overloaded.

Table 2: Types of failure recovery solutions.

Recovery Solution	Description
Private cloud Only (Nectar)	Failures resolved in the private cloud only.
Hybrid cloud (Nectar & Amazon)	Failures resolved in Amazon cloud.
Hybrid cloud (Nectar & Google)	Failures resolved in Google cloud.

6.2 Test Scenarios: Results and Discussion

Each test in the experiment involved the user request (described in the previous section) in the hybrid cloud environment. A configuration parameter was used to deliberately cause failures. We immediately began to measure TTR of RVIMS. We included a test scenario for each failure in the failure models except for the full private VM capacity failure—this exception was made because we were interested here in the failures that occurred during the run time of our user request. Five tests were conducted for each recovery solution.

6.2.1 VM Crash and Outage Failures Test Scenario

The VM crash failure and outage failure are similar. The impact of each failure is the only difference. The impact of an outage failure is complete service unavailability (downtime) while the effect of a VM crash failure is decreased QoS. As a result, we used the same test scenario for both. A recovery mechanism was used to launch new VMs. In the private cloud solution, the new VM was launched on the private cloud (Nectar), while in the hybrid cloud solution, it was launched on a public cloud. As shown in Figure 5, the results indicate that the private cloud had the lowest TTR, which means it was the fastest to recover from failures. The worst case showed that the TTR difference between the private cloud and the hybrid cloud (Google) ranged from 10-20 seconds. This may be an issue for critical applications. The Amazon hybrid cloud solution showed better TTRs than the Google hybrid cloud solution. However, in the case of cloud outage failure, there was no recovery at all in the private cloud solution. Also, when the private cloud was overloaded, the TTRs increased and became undermined.

6.2.2 VM Slowdown Failure Test Scenario

The VM slowdown failure can be triggered by using the VM slow down configuration parameter through adding a sleep statement. The results showed that TTRs in the private cloud solution were better than those of the hybrid cloud solutions (see Figure 6). In one case, a TTR for the Amazon hybrid cloud solution was the same as that for the private cloud solution. In another case, a TTR for the Amazon hybrid cloud solution was the worst (i.e. Test 2). Notice that TTRs in the private cloud solution were more stable than those of the other solutions. Additionally, the Google hybrid cloud solution was more stable than the Amazon one. Test 1 of the private cloud and the two other hybrid recovery solutions showed similar TTRs.

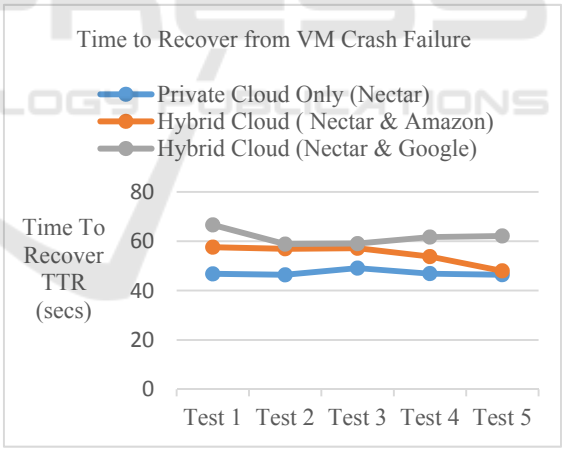

Figure 5: Time to recover from VM crash failure.

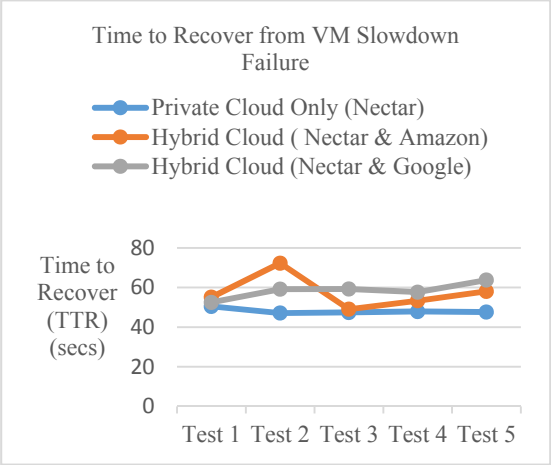

Figure 6: Time to recover from VM slowdown failure.

6.2.3 VM High Load Failure Test Scenario

When the requests on the application increase, the CPU and memory utilization subsequently increase. In our tests, the VM load increase configuration parameter was used to increase the CPU and memory utilization. This caused a VM high load failure to occur. The failure recovery mechanism here was to scale the cloud application by adding new VMs. As shown in Figure 7, the values of TTRs for the private cloud solution were the smallest of all of the tests. In terms of hybrid cloud solutions, the Google hybrid cloud solution showed shorter TTRs than Amazon. In addition, the Google hybrid cloud solution was more stable than Amazon. TTRs for the Google hybrid cloud solution were around 56 seconds while the average TTRs for the private cloud solution were of the order of 47 seconds.

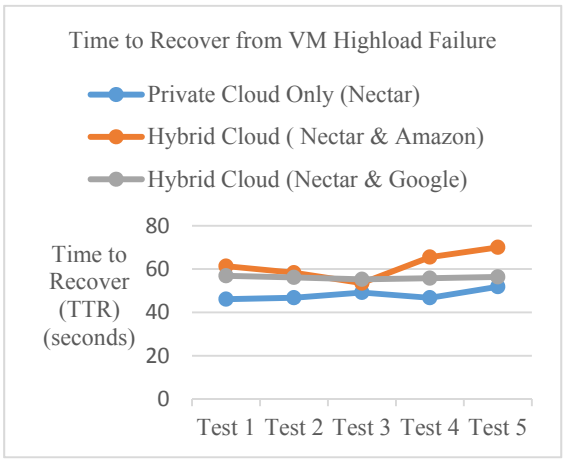

Figure 7: Time to recover from VM high load failure.

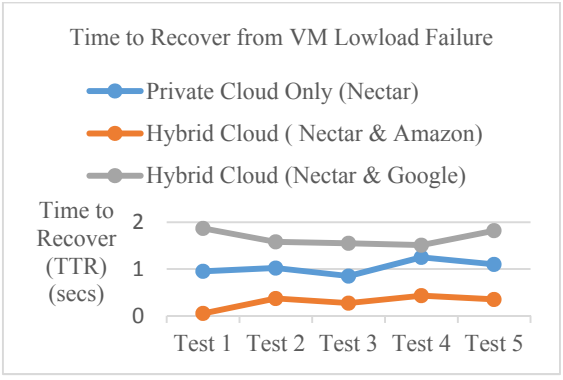

Figure 8: Time to recover from VM low load failure.

6.2.4 VM Low Load Failure Test Scenario

The VM low load failure is almost the opposite of the VM high load failure. Here, the application load decreases and thus the CPU and memory utilization are reduced. To avoid resources being underutilized, the cloud management systems need to scale down the applications immediately by terminating idle VMs. A VM load decrease configuration parameter can deliberately cause this kind of failure. Moreover, though this failure has no impact on the cloud application itself, it affects the resource utilization of the overall cloud platform. The results depicted in Figure 8 show that the Amazon hybrid cloud solution performed better with this kind of failure than the other recovery solutions. The average TTR for the Amazon hybrid cloud solution was about 0.3 seconds. The TTR values for the Google hybrid cloud solution were the highest. Overall, the TTRs for all solutions were small compared to other TTR failures.

7 CONCLUSIONS AND FUTURE WORK

We have presented a fault tolerant approach for efficiently managing cloud resources, improving reliability and availability and minimizing human intervention in hybrid cloud environments. The methodology behind this approach leverages ideas from recovery-oriented computing (ROC) approaches. A key assumption of the ROC approach is that failures will (eventually) occur. Instead of attempting to solely rely on predicting failures and attempting to avoid them completely, one should also focus on attempting to recover quickly from them. Thus, the intermediate goal of applying ROC

in our research was to reduce the recovery time after failures to meet QoS demands.

Our fault tolerant approach supports a range of failure models and a feedback control system model that leverages hybrid cloud services for autoscaling, resilience and load distribution. The RVIMS realizes the proposed models and heterogeneous cloud services. This fault tolerant system can overcome many scalability issues found in single cloud models.

In our research, we have taken into account awareness of failures and local resources when deciding upon resource provisioning. However, considering only those factors leaves a number of challenges unmet in achieving optimal resource use. Further studies are needed to determine how other factors influence the effectiveness of hybrid clouds.

Furthermore most resource provisioning algorithms suffer when dealing with big data including data transfer, limitations of network bandwidths and the topology awareness of clouds. It is imperative that such issues are addressed in order to make hybrid clouds more reliable and efficient. This is one focus of our future work.

Furthermore, augmenting our work with further (richer) models of fault tolerance and failure prediction is also an area of future consideration. Thus whilst ROC can help certain classes of application to recover, partial failures for long running applications can have unique requirements that need to be considered also.

Finally the challenge of bursting to the public cloud can often have implications on what applications and data can be recovered to external resources, e.g. due to privacy considerations of outsourcing. We shall also consider such demands as part of a more holistic approach to where and how RVIMS can be optimally applied.

ACKNOWLEDGEMENTS

The authors would like to express thanks to the Nectar Research Cloud (www.nectar.org.au) for the cloud resources used to perform this research.

REFERENCES

Amazon. 2016. *Amazon Elastic Compute Cloud* [Online]. Available: http://aws.amazon.com/ec2 [Accessed 22-09-2016].

Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R. H., Konwinski, A., Lee, G., Patterson, D. A., Rabkin, A., Stoica, I. & Zaharia, M. 2009. Above the Clouds: A Berkeley View of Cloud Computing. EECS Department, University of California, Berkeley.

Berkeley. 2004. *Recovery-Oriented Computing Overview* [Online]. Available: http://roc.cs.berkeley.edu/roc_overview.html [Accessed 03-11-2016].

Cascella, R. G., Morin, C., Harsh, P. & Jegou, Y. 2012. Contrail: a reliable and trustworthy cloud platform. *Proceedings of the 1st European Workshop on Dependable Cloud Computing.* Sibiu, Romania: ACM.

El-Refaey, M. 2011. Virtual Machines Provisioning and Migration Services. *Cloud Computing.* John Wiley & Sons, Inc.

Eucalyptus. 2008. *Eucalyptus Cloud Platform* [Online]. Available: www.eucalyptus.com [Accessed 25-09-2016].

Google. 2010. *Google. Post-mortem for February 24th, 2010 outage* [Online]. Available: https://groups.google.com/forum/#!topic/google-appengine/p2QKJ0OSLc8 [Accessed 02-10-2016].

Google. 2016. *Google Compute Engine* [Online]. Available: https://cloud.google.com/compute [Accessed 04-07-2016].

Grozev, N. & Buyya, R. 2014. Inter-Cloud architectures and application brokering: taxonomy and survey. *Software: Practice and Experience,* 44**,** 369-390.

Javadi, B., Abawajy, J. & Sinnott, R. O. 2012. Hybrid Cloud resource provisioning policy in the presence of resource failures. *Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on,* 10-17.

Laing, B. 2012. *Summary of Windows Azure Service Disruption on Feb 29th, 2012* [Online]. Available: https://azure.microsoft.com/en-us/blog/summary-of-windows-azure-service-disruption-on-feb-29th-2012 [Accessed 07-07-2016].

Libcloud. 2009. *Apach Libcloud* [Online]. Available: http://libcloud.apache.org [Accessed 05-09-2016].

Mattess, M., Calheiros, R. N. & Buyya, R. Scaling MapReduce Applications Across Hybrid Clouds to Meet Soft Deadlines. Advanced Information Networking and Applications (AINA), 2013 IEEE 27th International Conference on, 25-28 March 2013 2013. 629-636.

Microsoft. 2014a. *An Introduction to designing reliable cloud services* [Online]. Trustworthy Computing. Available: https://www.microsoft.com/en-us/twc/reliability.aspx [Accessed 08-01-2017].

Microsoft. 2014b. *Resilience by design for cloud services* [Online]. Trustwothy Computing. Available: https://www.microsoft.com/en-us/twc/reliability.aspx [Accessed 02-01-2017].

Nectar. 2016. *The Australian National eResearch Collaboration Tools and Resources (Nectar) Research Cloud* [Online]. Available: http://cloud.nectar.org.au [Accessed 13-10-2016].

NIST. 2011. *The NIST Definition of Cloud Computing* [Online]. Available: http://csrc.nist.gov/publications/PubsSPs.html [Accessed 9-10-2016].

OpenNebula. 2016. *OpenNebula Cloud Platform* [Online]. Available: http://opennebula.org [Accessed 13-11-2016].

OpenStack. 2010. *OpenStack Cloud Platform* [Online]. Available: https://www.openstack.org [Accessed 10-10-2016].

RightScale. 2006. *RightScale - A Cloud Management Solution* [Online]. Available: http://www.rightscale.com [Accessed 25-10-2016].

Sotomayor, B., Montero, R. S., Llorente, I. M. & Foster, I. 2009. Virtual Infrastructure Management in Private and Hybrid Clouds. *IEEE Internet Computing,* 13**,** 14-22.

Tanenbaum, A. S. & Steen, M. v. 2006. *Distributed Systems: Principles and Paradigms (2nd Edition)*, Prentice-Hall, Inc.

Voorsluys, W., Broberg, J. & Buyya, R. 2011. Introduction to Cloud Computing. *Cloud Computing.* John Wiley & Sons, Inc.

Yixin, D., Hellerstein, J. L., Parekh, S., Griffith, R., Kaiser, G. E. & Phung, D. 2005. A control theory foundation for self-managing computing systems. *Selected Areas in Communications, IEEE Journal on,* 23**,** 2213-2222.