Elastic Application Container: A Lightweight Approach for Cloud Resource Provisioning

Sijin He, Li Guo, Yike Guo, Moustafa Ghanem, Rui Han, Chao Wu

Department of Computing
Imperial College London
South Kensington Campus
London, United Kingdom

{sh107, liguo, yg, m.ghanem, rhan10, c.wu09}@doc.ic.ac.uk

Abstract—Virtual machine (VM) based virtual infrastructure has been adopted widely in cloud computing environment for elastic resource provisioning. Performing resource management using VMs, however, is a heavyweight task. In practice, we have identified two scenarios where VM based resource management is less feasible and less resource-efficient. In this paper, we propose a lightweight resource management model that is called Elastic Application Container (EAC). EAC is a virtual resource unit for delivering better resource efficiency and more scalable cloud applications. We describe the EAC system architecture and components, and also present an algorithm for EAC resource provisioning. We also describe an implementation of the EAC-oriented platform to support multi-tenant cloud use. To evaluate our approach and implementation, we conducted experiments and collected performance data by comparing VM-based and EAC-based resource management with regards to their feasibility and resource-efficiency. The experiment results show that our proposed EAC-based resource management approach outperforms the VM-based approach in terms of feasibility and resource-efficiency.

Keywords-component; Architecture; Cloud Computing; Lightweight; Multi-tenancy; Resource-efficiency;

I. INTRODUCTION

Driven by the rapid growth of the demand for efficient and economical computational power, cloud computing, has led the world into a new era. By enabling virtualisation technology on physical machines (PMs), it not only gives immense benefits in terms of reliability, efficiency, and scalability, but also provides virtual computational services, such as computing power, storage and network, in such a way end-users are able to consume them over the Internet as utilities.

The VM-based virtual infrastructure has been adopted widely in cloud computing environment for elastic resource provisioning. A VM allows the resources of a single PM to be shared across multiple VMs for maximum efficiency. However, performing resource management using VMs as a collection of resources is a heavyweight task. In practice, we have identified two scenarios where VM based resource management is less feasible and less resource-efficient.

Heavyweight VM Migration - VM migration over LAN is one of the most common VM resource management operations for cloud providers. However, the VM migration over LAN is a heavyweight task. In a shared-storage environment, a VM live migration requires transferring the working state and memory from one PM to another over LAN. It consumes a large amount of I/O and network traffics in the LAN environment [1]. In a WAN environment, mechanisms for migrating VMs remain elusive. The VM-based migration across IDCs over the Internet [2] also requires a huge amount of I/O for both IDCs and costs a great amount of time for replicating a VM from one IDC to another. This is because a VM generally consists of a guest OS and applications (as shown Figure 1) in which the size of the VM can vary from hundred megabytes to 50 gigabytes or more depending on the sizes of the OS and the applications. This makes the VM resource management in WAN infeasible.

Overhead in the VM-based Approach – A VM generally consists of a guest operating system (OS) for running applications. The guest OS in the VM always occupies a considerable amount of VM resources, this cause the overhead in a VM. As number of VMs increases, the total amount of overhead in VMs increases in the cloud system. Therefore, the overhead in VMs consumes a great amount of cloud resources. This leads to resource-inefficient in the VM-based approach.

In this paper, we propose a lightweight resource management model that is called Elastic Application Container (EAC). EAC is a virtual resource unit for delivering better resource efficiency and more scalable cloud
applications. Each EAC also has its own resource capacity (such as number of CPUs, size of RAM, etc.) to run an instance of an application. The EAC system provides a lightweight resource management solution in comparison with the VM-based approach. Further, it natively supports auto-scale of applications which allows resource use by each end-user is tracked and adjusted as needed.

The rest of the paper is organised as follows: Section II reviews the related work of the VM-based and the EAC-based approach. Section III presents the model and architecture of the EAC system. Section IV presents the implementation of the EAC system. Experiments and evaluation are presented in Section V and discussion of the work is in Section VI, and we conclude the paper and suggest some future work in Section VII.

II. RELATED WORK

Different cloud virtualisation technologies exist in the cloud computing industry. Zing [3] is JVM-based virtualisation technology that allows Java applications to be executed in its Zing JVM which is elastic and scalable. However, the mainstream of virtualisation technologies nowadays is based on VMs. VM-based virtualisation projects producing VM hypervisor software [4; 5; 6; 7; 8] have enabled resource provisioning mechanisms for end-users. Among various machine virtualisation technologies nowadays, Xen and KVM are popular due to they have a combination of features that make them uniquely well suited for many important applications.

Natis et.al [9] propose eleven characteristics of multi-tenancy for Cloud-enabled application platforms (CEAP). The following are the commercial products offering CEAP. Smartfrog platform automates the configuring, deploying and scaling process of software components written in Java [10]. Similar to Smartfrog, RollBase components are also developed with pure Java and they can manage applications in a list of mature enterprise servers such as Apache, Tomcat and MySQL [2]. This platform further integrates with the Google and supports its services including Gmail, Google Calendar and Google Docs. Appenda and Expanz platforms are built to support .NET developers to develop, deploy and scale their applications [11]. These applications’ servers can be automatically scaled from just one to a huge number using the Microsoft stack. GigaSpaces eXtreme Application Server (XAP) [12] is developed for the elastic scaling of e-commerce applications in different domains such as big-data analytics and finance. Finally, LongJump [13] is an on-line database that not only enables the dynamic scaling of data but can also adapt the business process according to demand. Hence this product can be applied to a wide range of applications.

III. ELASTIC APPLICATION CONTAINER

The architecture of the EAC-based approach has some similarities with the architecture of VM-based approach as shown in Figure 1 and 2. These two architectures both support multi-tenancy in the cloud environment. The VM-based approach requires a hypervisor to manage VM resources whilst the EAC-based approach requires an Elastic Application Server to manage EAC resources.

However, they still have some differences in terms of system architecture. From an internal infrastructure perspective, a VM requires a guest OS to run applications whilst an EAC can run an instance of an application without an OS. From the deployment environment perspective, the EAC-based approach gives more flexibility than the VM-based approach. This is because VM hypervisor software can only be deployed to PMs of IDCs whilst the EACs can be hosted in either PMs or VMs from IaaS. From scaling perspective, the auto-scale of an application in the VM approach is not natively supported whilst the auto-scale of an application in the EAC approach is a native functionality for exhibiting the elasticity of applications. With auto-scale of an application, end-users can concentrate on their application developments without considering underlying resource management of their applications. From end-users’ perspective, the access point of virtual resources in the VM approach is a virtual computer, the end-users need to setup the system before running their applications. However, the access point in the EAC approach is an application. The end-users can run their applications without setting up the system.

A. Model of an EAC

The model of an EAC in a node (PM or VM) is shown in Figure 3. On top of the hardware, there is a conventional OS where an Elastic Application Server (EAS) is installed. The EAS which is similar to the concept of VM hypervisor software is an EAC manager that allows multiple EACs to run concurrently in a node. Each EAC has its own capacity and can execute one instance of an application at a time. The EAS not only provides an environment for running applications, but is also responsible for EAC resource management. The type of application running in an EAC is
specifically for a certain domain of applications, such as web applications.

Definition 1 - An Elastic Application Container (EAC) is an isolated virtual container for hosting one instance of an application at a time. It has $R$ types of resources, $r_1, r_2, \ldots, r_R$, where $R \in \mathbb{N}$. Each EAC is required to specify the maximum capacity for each resource type.

B. Architecture of the EAC system

The architecture of the EAC system is flexible and modular with a hierarchical design which is common in many cloud architecture [14; 15] as shown in Figure 4. Using this design is beneficial for scaling the EAC system as clusters can be added or removed dynamically in response to real-time demands of the system. In this way, different clusters in different IDCs can easily join or leave the EAC system. This makes cross-IDC resource management feasible.

There are four high-level components in this hierarchical system, each with its own Web-service interface. The EAC Controller controls the life cycle of an EAC, the fundamental computing resource unit of the EAC system. On top of EAC Controllers, there are Node Controllers which are designed for managing the node and monitor application instances running in EACs. Cluster Controllers are responsible for monitoring all statuses in all nodes in the cluster and make appropriate resource management on the nodes in response to the demands of the nodes. The overview of the main system components in the EAC system is shown in Figure 5.

1) EAC Controllers

An EAC Controller, a component in the Elastic Application Server, controls the life cycle of an EAC as demonstrated in Figure 6. Upon verifying the authorisation, only the owner of an EAC or an administrator is allowed to control the life cycle. To start deploying an application in an EAC, an end-user must firstly create an EAC, and then upload an application to the EAC. On receiving a request of creating an EAC, the EAS notifies the Virtual Resource Controller to acquire the server resources for the EAC, and then it instantiates a new EAC Controller for controlling the life cycle of the EAC. While an application instance is being deployed to an EAC, the monitor keeps track of the application instance resource usage, such as CPU cycles, memory used, data transfer, etc., and then reports to the Cloud Controller. The Cloud Controller will make decisions on scaling the application according to its application resource usage. Note that we present an algorithm for scaling the application in the next section. To destroy an EAC, the EAC Controller firstly stops the application instance and then deletes the files and its data associated with the application. Finally, the Virtual Resource Controller is notified to release the EAC resource.

2) Node Controller

A Node Controller controls the execution and termination of a node (PM or VM) on the host where it runs and has a monitor which is one of the most important components in the EAC system. The main functionalities of the monitor are as follow:

- **Node usage monitoring** – It serves for two purposes: monitoring PMs and VMs. For monitoring PMs, the monitor keeps track of the node resource usage including CPU time, memory, I/O operations and network traffic etc. For
monitoring VMs, the monitor calls the IaaS provider’s external monitoring APIs to obtain the resource usage of the node which are then propagated to the Cluster Controller.

- **Application instance usage monitoring** – The monitor keeps recording all the application instance resource usages and then propagates the information to the Cloud Controller.

3) **Cluster Controller**

The Cluster Controller generally executes on a machine that has network connectivity to the machine running the Cloud Controller and to the nodes running the Node Controllers. It has a scheduler module which is responsible for resource management of its nodes and EACs. The main functionalities of the scheduler are shown as follow:

- **Gather node information from Node Controllers** – It gathers information about nodes from the Node Controllers in a cluster.
- **Scale the cluster** – It is responsible for adding or removing nodes from the cluster which can be manually scaled by the administrator or auto-scaled in response to the total resource usage of nodes in the cluster. We apply time series [16] on the total resource usage history of nodes in the cluster to forecast the resource usage in the future for auto-scale.
- **EAC resource allocation to a node** – When it receives a request for an EAC creation from the Cloud Controller, it searches for a node that has enough free resources to host the EAC. We use vector-based approach for EAC resource allocation [17].

4) **Storage Controller**

The storage controller provides a database storage service for applications running in EACs. Application developers can store their application data through the interface (Data Storage portal) provided by the storage controller to access the database clusters. The scheduler of the Storage Controller is responsible for scaling up or down the database cluster in response to the resource usage of the database. We also apply time series on the database resource usage history to forecast the resource usage in the database for auto-scale. In addition to that, the scheduler is also responsible for provisioning database resource for an application, such as total data storage, number of concurrent database requests for each application.

5) **Cloud Controller**

It is the entry-point into the EAC system for end-users and administrators. The underlying resources that comprise the EAC system are managed by the Cloud Controller. The Cloud Controller is a collection of web services which are grouped by their roles into two categories.

a) **Interface Services**

The Interface Services present user-visible interfaces (EAC, User Account and Data Storage Portals) to end-users and administrators, handling user authentication, providing system statistics and management tools.

b) **Resource Services**

The Resource Services has a scheduler which is responsible for managing the following main functionalities in the EAC system.

- **Gather application information from Node Controllers** – It gathers information about applications running in EACs from all Node Controllers in all clusters.
- **Scale applications** – The Cloud Controller can scale up an application by replicating an application instance from an EAC to another newly created EAC, and scale down an application by removing a replicated application instance from an EAC. The EAC system comes with the auto-scaling functionality which is to assure efficient use of shared resources. With auto-scale enabled in an application, the Cloud Controller closely monitors each application instance in an EAC and properly scales in response to the change of the application instance usage. Algorithm 1 is used for scaling applications in the EAC system.
- **Load-balancing for application instances in EACs** – Each application has an application load-balancer which can forward application requests to the same application instance in different EACs in different clusters as shown in Figure 7. Different load-balancing strategies can be applied. For simplicity, we use round-robin [18] to distribute Http requests to the application instances.
- **EAC migrations** – It has a migration utility that can easily migrate application instances in an EAC to another EAC in another node over LAN or WAN. It makes the cross-IDC resource management become possible. The EAC migration requires the application instance to be stopped, transferring to another EAC in a target node, and then restart the application instance in the target node. We use the vector-based approach for finding the most suitable target node for migration.
- **EAC Placement to a cluster** – Upon receiving an EAC creation request from an end-user or the EAC system, the Cloud Controller requires deciding a suitable cluster for hosting an EAC which is based on the following factors: resource usage of the cluster, geographical locations of the cluster and incoming requests, etc. We apply reactive placement algorithms [19] to find the most suitable cluster for EAC placement.

C. **Resource Provisioning of the EAC system**

Multi-tenancy implies that the same physical application server is used to support execution of multiple instances of one or more applications, each as a separate logical tenant. The EAC system supports multi-tenancy which allows end-users using EACs operate in virtual isolation from one
another and manage an application instance as though they each have a separate VM, yet their data remain secure and insulated from the activity of all other end-users. An end-user allows having multiple applications depending on the number of EACs the one has. However, one EAC can only be managed by one end-user. Furthermore, each application securely hosted in an EAC is a complete and independent unit. EAC separates the application in its own protected and reliable environment which is independent of the operating system and hardware.

Definition 2 - An EAC is full when an application in the EAC consuming any of its individual resources exceeds the corresponding maximum capacity of the resource in the EAC.

There are two levels of managers in the EAC system: Global Manager (GM) and Application Load Balancing Manager (LBM). Every application has its LBM, while there is only one GM in the Cloud Controller of the EAC system. The GM dispatches requests it receives, denoted as rq, to the LBMs of the corresponding APP. The requests received in a LBM are placed in its local waiting queue $\text{queue}_{\text{app}} = \{rq_1, rq_2, ..., rq_n\}$ and executed on the First-come-First-Served (FCFS) basis. The LBM of an application further dispatches the requests in its local waiting queue to one of the application instances of the application.

With auto-scale enabled, an end-user firstly define the resource requirement of an EAC for hosting an application instance, and then specify the maximum number of EACs required for an application, denoted as $\text{MAXEAC}_{\text{app}}$. The application resource provisioning of EACs are shown in Algorithm 1.

Algorithm 1: Resource_Provision($\text{queue}_{j}$)

1. $\Omega = \text{set of EACs that are not full for an application } j$;
2. $\Theta = \text{set of all EACs for an application } j$;
3. $\Psi = \text{set of all nodes in the EAC system};$
4. while ($\text{queue}_{j} \neq \emptyset$)
5. 
6. 
7. 
8. 
9. 
10. 
11. 
12. 
13. 
14. 
15. 
16. 

IV. IMPLEMENTATION

All the system components in the EAC system are implemented as web services in Java, which makes the system platform independent. We modified open-source Tomcat Application Server [20] as Elastic Application...
V. EXPERIMENTS AND EVALUATION

In this section, we evaluate the VM-based and the EAC-based resource management approaches with respect to the feasibility and resource-efficiency.

A. Feasibility of Resource Management

In order to investigate the feasibility of resource management between the VM-based and the EAC-based approaches, we conduct several experiments on VM and EAC migrations over LAN and WAN. This is because migration is one of the most frequently used resource management operations.

1) Experimental Setup

We have conducted the migration experiments based on the EAC system and IC Cloud [15] (a Xen-based IaaS). There are three types of migrations are performed over LAN and WAN in the experiments: EAC migrations, VM static migrations [1] and VM live migrations [21]. Migrations over LAN are conducted in London with 100 Mbps bandwidth whilst migrations over WAN are performed between London and Shanghai, China. Before conducting the EAC migration experiments, we firstly setup the EAC system in both IDCs of London and Shanghai, and then we deploy a 250-megabyte application to an EAC in a node. Note that the 250-megabyte application is a very large web application nowadays. For VM migration experiments, we setup the IC Cloud environment in both London and Shanghai, and then we use CentOS to run an unmodified Tomcat server where the same 250-megabyte application is also deployed. Note that the size of the VM is 1 GB which is the smallest size to be capable to run the OS and the applications. After setting the environment for experiments, we perform every type of migrations 50 times over LAN and WAN.

2) Evaluation

In a LAN environment, an EAC migration and a VM static migration have a slight difference in migration time as shown in Figure 10. The EAC migration over LAN requires the instance to be stopped, transferring an application instance in the EAC to another EAC in a target node, and then restarting the application instance in the target node whilst the VM static migration over LAN requires a complete shut-down of a VM in a node, transferring configuration files to a target node and restarting the VM in the target node in the shared-storage infrastructure [1]. However, a VM live migration takes approximately three times longer than the other two methods as shown in TABLE I. The live migration requires transferring VM working states and memory from one node to a target node in the shared-storage cloud infrastructure while it is running. The time of live migration over LAN is still acceptable for resource management, but it costs a large amount of I/O and network traffics. Therefore, the VM live migration is a heavyweight resource management operation.

Figure 10. Migrations over LAN

Figure 11. EAC Migrations over LAN and WAN

Figure 12. VM Static Migrations over LAN and WAN

Figure 13. VM Live Migrations over LAN and WAN
In a WAN environment, it is easy to see that all types of migrations take longer to finish than migrations over LAN as shown in Figure 11, 12 and 13. VM static and live migrations take much longer over WAN as the time differences between LAN and WAN are 763 and 743 respectively as shown in TABLE I. The EAC migration is still feasible resource management since the time difference between LAN and WAN is 158 seconds which is 4.8 times faster than both VM static and live migrations. This is because a VM static or live migration over WAN requires transferring the whole image of a VM including guest OS and its applications to a target node. This costs a huge amount of I/O, network traffics and time. In contrast, the EAC migration only requires transferring an application instance to a target node which costs far more less I/O and network traffics. In practice, a regular sized VM is normally greater than 1GB, migrating such a large VM over WAN is an infeasible resource management operation as it consumes a great amount of resources which cause negative effect to the cloud system.

### TABLE I. AVERAGE TIME FOR MIGRATIONS IN SECONDS

<table>
<thead>
<tr>
<th></th>
<th>Application Migration</th>
<th>VM Static Migration</th>
<th>VM Live Migration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Over LAN</td>
<td>27</td>
<td>40</td>
<td>94</td>
</tr>
<tr>
<td>Over WAN</td>
<td>185</td>
<td>803</td>
<td>837</td>
</tr>
<tr>
<td>Difference between LAN and WAN</td>
<td>158</td>
<td>763</td>
<td>743</td>
</tr>
</tbody>
</table>

### B. Resource-efficiency

In this section, we compare the EAC system and IC Cloud in terms of resource efficiency. To measure the resource efficiency for both systems, we examine the change of Http response time as the number of VMs or EACs increases in a PM.

1) **Experimental Setup**

We setup the experiment by using two PMs with exactly the same hardware configuration, i.e. 2 cores of CPUs and 4GB RAM. The IC Cloud platform was installed in PM 1 and the EAC system was installed in PM 2. For each VM we create in the IC Cloud platform, we also use CentOS to run an unmodified Tomcat server where a computing-intensive application is deployed. The computing-intensive application is used for calculating π by using Gregory’s series [22] which slowly converges to π:

$$\pi = 4 \sum_{k=0}^{n} \frac{(-1)^k}{2k + 1}$$

We decide to calculate 100,000 terms for the series, where n = 100,000. For each EAC we create, we also deploy the same application for calculating π.

In order to conduct the experiment for VMs, at the first iteration, step 1 is to start with creating one VM, denoted as V M_{11} where the first 1 represents the first iteration and the second 1 represents the index of the VM. The capacity of V M_{11} is equivalent to the current remaining capacity of the PM, i.e. after taking away the overhead of the hypervisor and the hosted OS, and other applications, and then start the VM_{11} to run the target application. Step 2 is to invoke an Http request dispatcher to send 1000 Http requests to the target application in order to obtain the Http response time per 1000 requests for the target application in V M_{12}, denoted as RT_{VM_{11}}. Step 3 is to repeat the step 2 for 10 times and calculate the average Http response time per 1000 requests for the target application in V M_{11}, denoted as \overline{RT}_{VM_{11}}. We then set it as the average response time at iteration 1, denoted as p_1. At the second iteration, we delete V M_{11} and then we create and start two new VM, V M_{21} and V M_{22}. Each VM has the resource capacity equivalent to \frac{1}{n} of the current remain capacity of the PM, where n represents the number of the VMs in this iteration. We invoke the Http request dispatcher to send 1000 Http requests to the target applications in those two VMs respectively. We then repeat the Http dispatching process for 10 times in order to calculate \overline{RT}_{VM_{21}} and \overline{RT}_{VM_{22}}. We then calculate p_2 which is the mean of \overline{RT}_{VM_{21}} and \overline{RT}_{VM_{22}}. At the third iteration, we delete V M_{21} and V M_{22} and create three VMs, and so on. We finally obtain a list of average response time (p_1, p_2, ..., p_n) from iteration 1 to n. The experiment for EACs was also conducted in a similar fashion. We have performed 13 iterations for VMs and EACs, the results are shown in Figure 14 and 15.

2) **Evaluation**

Figure 14 shows the Http response remains relatively steady as the number of VMs increases from 1 to 10, but there is a sudden increase in the http response time as the number of VMs increases from 10 to 13. This is due to the fact that there is insufficient resource in a VM to process the incoming Http requests. As we mention earlier, each VM consists of a guest OS and applications. In this experiment, a VM consumes 294 MB RAM overhead in average for running CentOS and Tomcat Server. As the number of VMs increases, the resource capacity of a VM decreases provided the PM resources remain constant. Hence, the remaining resource capacity in a VM for processing Http requests decreases given that the overhead is constant. This causes insufficient resources for processing the Http requests and leads to slower Http response time. Figure 15 shows the Http response time stays steady as the number of EACs increases. This is because an EAC doesn’t need an OS to run an application. It only consumes a small amount of overhead for each application running in an EAC. In the experiment, an EAC consumes only 10 MB RAM in average which is 29.4 times smaller than the VM. Each EAC has sufficient resource to run its application. Therefore, the EAC-based approach is more resource-efficient than the VM-based approach for running the same number of applications.
Even though VM-based approach offers VMs which allows a variety of applications to be executed, we stress that the EAC-based approach is not a replacement solution for VM-based approach. The EAC-based approach is an efficient approach specifically for a certain domain of applications, such as web applications. The VM-based approach is not suitable for web applications this is because web applications require a more scalable and flexible environment, but the resource management of the VM-based approach is heavyweight as demonstrated in the previous section. Currently, the EAC system only supports one type of programming language, Java. Using our model in Section III, application servers for different programming languages can be modified for supporting multi-tenant cloud use.

VII. CONCLUSION AND FUTURE WORK

In this paper, we proposed a lightweight resource management model that is called Elastic Application Container (EAC) for delivering scalable cloud applications. We presented the architecture of the EAC system and an algorithm for EAC resource provisioning. We then described an implementation of the EAC system based on modifying a Tomcat Server to support multi-tenant cloud use. Our evaluation included experiments to compare the VM-based and EAC-based approaches in terms of feasibility and resource-efficiency of resource management. The experiments show that EAC system offers lightweight resource management operations and more resource-efficiency than the VM-based approach. Furthermore, we have made an EAC system available for trials to all who wish to try the system in our computing department. The feedbacks so far have been very positive.

Future research will expand on several dimensions of our work here. On the theoretical side, we intend to more deeply explore efficiency and pricing mechanisms for EACs. We are also studying some alternatives algorithms which perform best for resource management operations. On the implementation side, we will carry on supporting more programming languages using different application servers for multi-tenant cloud use.

REFERENCES