Elastic Resource Provisioning for Cloud Based on Docker

Shi-da QIU¹,², Ming-fa ZHU¹,², Guang-jun QIN¹,², Li-min XIAO¹,², Bin SONG¹,², Shou-xin WANG³ and Rui LIU⁴

¹School of Computer Science and Engineering, Beihang University, Beijing 100191, China
²State Key Laboratory of Software Development Environment, Beihang University, Beijing 100191, China
³Space Star Technology Co., Ltd, Beijing 100086, China
⁴National Engineering Research Center for Science & Technology Resources Sharing Service, Beijing 100191, China

{qiushida, zhumf}@buaa.edu.cn

Keywords: Elastic resource provisioning, Cloud computing, Benefit analysis, Docker.

Abstract. Elastic resource provisioning is one of the key techniques for cloud computing, as it can make resources to be organized and provided on demand, resolve the contradiction between resource utilization and user satisfaction, and ensure higher service level agreement. Considering the deficiency of the present cloud resource managements, we propose an innovative elastic resource provisioning approach with a loosely coupled and scalable architecture for cloud based on docker. The elastic resource provisioning method combines both dynamic virtual clustering techniques and economic benefit analysis theories. The results show that our approach can meet the resource supply requirements of elastic cloud with excellent scalability, great agility, strong robustness and high resource utilization.

Introduction

As one of the most significant revolutions in the world, cloud computing[1,2] has changed the IT service model, as well as the way people using network and computer. Cloud computing strengthens agility, scalability, feasibility of the distributed computing and lowers potential cost. However, the resource management technologies are still not mature at present, especially in the aspect of elastic resource provisioning for cloud.

First, due to the diversity and uncertainty of the tasks, the cloud load is fluctuating, and the resource requirements are also constantly changing. The resource supply mode based on static pool is difficult to meet the demand of elastic scaling. Second, the typical resource allocation method based on fixed templates is short of flexibility, and the resources wasting phenomenon is too serious. In fact, the average data center server utilization is only 18% [3]. Last but not least, system load cannot directly reflect the degree of resource demand. Load prediction technology is difficult to give the right suggestions for resource provisioning, especially for the tasks with special resource preferences.

In view of the above problems, we have designed an elastic scalable resource provision approach which combines dynamic virtual clustering techniques and benefit analysis model. It has been proved that the approach can effectively meets the resource supply requirements for cloud.

System Design and Implementation

Scaling Design

In order to improve the reliability and scalability of the system, we introduce zookeeper[4] as the coordination control center of the distributed system. Fig. 1 shows the high scalable distributed architecture of the elastic cloud system.

Unlike centralized control mode, we decouple the strong dependences between different modules by introducing a message sharing mechanism. In the system, zookeeper cluster acts as a global
negotiation center and is responsible for synchronizing all messages and statuses about resource provisioning and task scheduling. All the modules can not only push their own statuses to the zookeeper, but also watch and take action based on the data drawn from the zookeeper. For instance, when the client pushes a new task submission to the zookeeper, the corresponding cluster will detect the request and run the task proactively.

By introducing zookeeper, the dynamic management of clusters becomes very easy. In our system, all nodes are dynamically generated based on docker[5]. According to the running state, we can divide the nodes into three categories: newborn node, surviving node and dying node. When more computing resources are needed, the container manager can just create a proper amount of newborn nodes, and register them to the zookeeper. The cluster resource manager will detect the changes and incorporate the newborn nodes into existing or newly built clusters, while changing their status to surviving. After the completion of the tasks, the corresponding nodes will be transferred to the dying state and eventually be removed to release the occupied resources. All the state changes of the nodes are synchronized to the zookeeper in real time, and this provides support for the resource manager to make appropriate resource provisioning decisions.

The main advantages of the structure can be summarized as follows:

High scalable. Thanks to the high concurrency support capability of the zookeeper cluster, our system can be extended to a large scale, which can effectively meet the multi-cluster computing needs for cloud.

High available. As all the key information is stored in zookeeper, even if a component failure, it will not cause the failure of the entire cloud system. Based on the message sharing techniques, our system can quickly recover from failure.

High flexible. Compared to the traditional virtual machine, the container is more flexible in resource organization. We use docker containers to achieve the dynamic generation of computing clusters, with the advantages of short startup time and high resource utilization.

Elastic Cloud Implementation

Our elastic resource provisioning system is built on Apache Hadoop YARN[6] platform, as an enhancement of the cloud computing resource management. Fig. 2 shows the overall scaling framework of the elastic cloud.

We propose a hierarchical resource provisioning model for cloud based on docker. In order to ensure the segregation of resources and tasks among tenants, the cluster is generated on demand and assigned to each tenant independently. Considering that the system load is always fluctuating in open cloud service platform, the resource provisioning in our system is multidirectional, including horizontal and vertical scaling.

In terms of horizontal scaling, we adopt multi-cluster scheme to break the size limit of single centralized cluster. Besides, the problems of dynamic load balancing and tenant segregation are also addressed by the way. In our elastic cloud system, all the clusters are organized upon virtual
containers and assigned to the tenants automatically. In terms of vertical scaling, instead of load monitoring and prediction, we add a resource interceptor module into the resource manager to collect the key communication messages about resource allocation between master and slaves in the cluster. According to the resource request messages, the elastic cloud can increase or decrease the number of computing nodes of the cluster to meet the requirements of the running tasks.

As shown in Fig. 2, the elastic resource provisioning cloud system consists of three components: client, outside and inside resource manager. As the task submitter, the client provides API to the developers who use the elastic cloud platform. The outside resource manager is the core module of cluster dynamic resource provisioning. In this component, we use the benefit analysis model to balance the size and quantity of the clusters, and apply zookeeper as a coordinate service to realize cluster’s self-organization. In the physical layer, all the resources are dynamically packaged by the virtual containers. And this work is done by the container manager which built on docker. The inside resource manager is mainly responsible for vertical scaling. Based on the master-slave architecture of YARN, a resource interceptor is injected into the master node to collect resource request information and a local agent is embedded in the slave nodes to synchronize status with zookeeper.

**Resource Provisioning Algorithm**

Based on the above architecture, we propose a benefit analysis[7] model which uses economic efficiency to measure resource value to guarantee the elasticity of our cloud. The main objective of this model is to balance the size and quantity of the dynamic clusters.

Define $T_r(t) = \{t_{r1}(t), t_{r2}(t), \ldots, t_{rn}(t)\}$ as the resource requirements at time $t$ of the tasks numbered from 1 to $n$, $T'_r(t) = \{t'_{r1}(t), t'_{r2}(t), \ldots, t'_{rn}(t)\}$ as the resource provisioning for the tasks at time $t$, $T_p = \{t_{p1}, t_{p2}, \ldots, t_{pn}\}$ as the resource purchase price of the tasks, $c(t)$ as the selling price of the resources at time $t$, and $T_e = \{t_{e1}, t_{e2}, \ldots, t_{en}\}$ as the expected execution time of the tasks. Then, we can draw the benefit expression, as Eq. (1).

$$\text{Benefit} = \sum_{i=1}^{n} \left( \int_{0}^{t_{ei}} (t_{pi} \cdot t_{r1}(t) - c(t) \cdot t'_{r1}(t)) \right).$$

In the centralized distributed system, with the continuous growth of the scale, the master node will accumulate more and more management pressure. Larger clusters mean higher management costs. Define $\text{size}(s, t)$ as the size of cluster $s$ at time $t$, $g(x)$ as the management cost when the cluster size is $x$, $N$ as the cost of building a new cluster, $d$ as the cluster size increment caused by the new task, and $c_b$ as the resource base price.
When using an existing cluster, the cost can be expressed as:

$$c(t) = c_b + g(size(s, t) + d).$$  \hspace{1cm} (2)$$

On the contrary, the cost of creating a new cluster to perform the task is:

$$c(t) = c_b + g(0 + d) + N.$$ \hspace{1cm} (3)$$

In order to maximize the benefits, we should make $$c(t)$$ as small as possible. Define $$B_1$$ as the result of Eq. (1) and (2), and $$B_2$$ as the result of Eq. (1) and (3). Then the objective function can be expressed as:

$$Obj = \max\{B_1, B_2, 0\}.$$ \hspace{1cm} (4)$$

Based on the above description and discussion, the benefit analysis resource provisioning algorithm can be described in Table 1.

<table>
<thead>
<tr>
<th>Table 1. Benefit analysis resource provisioning algorithm description.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input</strong>: Task List L1, Cluster List L2</td>
</tr>
<tr>
<td><strong>Output</strong>: Resource Provisioning Strategy</td>
</tr>
<tr>
<td>For each task T in L1 Do</td>
</tr>
<tr>
<td>$$B_1 \leftarrow \text{horizontalScaleBenefit}(T, L2)$$</td>
</tr>
<tr>
<td>$$B_2 \leftarrow \text{verticalScaleBenefit}(T, L2)$$</td>
</tr>
<tr>
<td>Obj $\leftarrow \max{B_1, B_2, 0}$</td>
</tr>
<tr>
<td>If Obj $= B_1$ Then</td>
</tr>
<tr>
<td>[L1, L2] $\leftarrow$ doHorizontalScale(T, L2)</td>
</tr>
<tr>
<td>Elif Obj $= B_2$ Then</td>
</tr>
<tr>
<td>[L1, L2] $\leftarrow$ doVerticalScale(T, L2)</td>
</tr>
<tr>
<td>Else</td>
</tr>
<tr>
<td>[L1, L2] $\leftarrow$ delaySchedule(T, L2)</td>
</tr>
<tr>
<td>Fi</td>
</tr>
<tr>
<td>End For</td>
</tr>
</tbody>
</table>

For each job, the benefit analysis model calculates the expected return of the two resource allocation methods respectively. If the new cluster schema is more profitable, the system will be expanded in the horizontal direction. On the contrary, if the existing cluster scheme is better, the system will perform vertical resource allocation. Particularly, when the task cannot meet the demand for profit, it will be delayed to wait for the next schedule.

In a word, the benefit analysis is always choosing the most profitable way to do the cluster scaling. Through the benefit analysis model, the system can maintain the balance between the quantity and the size of the clusters. And in this way, the resources are allocated under the objective of the optimal benefit.

**Experimental Evaluation**

We designed a task submission scenario which is extremely common in cloud environments. In this scenario, all of the tasks are created according to Poisson distribution. And the tasks are independent of each other.

In order to verify the cluster expansion capability, the Poisson distribution function adopts different $$\lambda$$ parameters to simulate the effect of the varying load in cloud. Eq. (5) shows the form of the Poisson distribution function.

$$P(X = k) = \frac{\lambda^k}{k!} e^{-\lambda}, \lambda = \begin{cases} 2, t \in [0,50) \\ 5, t \in [50,100) \\ 1, t \in [100,150) \end{cases}.$$ \hspace{1cm} (5)$$

The experimental results are shown in Fig. 3 and Fig. 4.
Fig. 3 shows that our resource provisioning approach has a strong ability to adapt to the changes of the work load. With the increase of the running tasks, the system can automatically create new clusters to balance the system load. When the scale of cluster A is too large, cluster B can expand its own scale quickly to avoid the continuous influx of tasks to cluster A.

We also compared the differences in resource utilization between the load trigger method and ours. As shown in Fig. 4, the benefit analysis resource provisioning method based on message interceptor technique and container technique can realize a more granular, accurate and flexible resource assignment, and it has a higher resource utilization rate than the load trigger method. Besides, because the message interceptor technique overcomes the problem of poor accuracy of the load trigger method, the resource utilization of our approach is more stable than the load trigger’s.

**Related Work**

The elastic resource management in cloud computing has been a hot research topic. Roy[8] proposes a resource allocation method based on load forecasting model, which use historical data to predict future resource needs. Salah[9] uses Markov chain to predict resource demand and reduce the heavy dependence on historical information. According to the characteristics of system real-time state, Gandhi[10] proposes a resource management method for energy efficiency based on control theory. Urgaonkar[11] proposes a resource allocation technology based on queuing model for multi-tier applications. Quasar[12] uses the classification technology of machine learning to analyze the impact of resource allocation on the service level, and resource assignment is carried out under the guidance of the classification results. In industry, as a typical representative of elastic cloud, Amazon EC2 [13] use a dynamic resource provisioning scheme based on load trigger. However, Amazon Auto Scaling is not fully automatic, which depend on the user to define the conditions for scaling up or down resources. And it is usually difficult for the user to figure out the proper scaling conditions.

From the above research, we can see that the load forecasting plays an important role in elastic resource management, but the implementation of load prediction is often very complicated, and it is usually difficult to translate work load into resource requirements. Furthermore, the existing cloud platform mostly use virtual machine as the basic unit of resource allocation, which can lead to the problem of serious waste of resources.

**Conclusions**

In this paper, we proposed an innovative elastic resource provisioning approach for cloud based on docker. The elastic resource provisioning method combines both dynamic virtual clustering techniques and economic benefit analysis theories. We introduce zookeeper to coordinate the key information, which enhance the system robustness, reliability and scalability. At the same time, we use docker container instead of virtual machine, not only improve system resource utilization, but also improve system flexibility. Through theoretical analysis and experimental verification, our approach is highly reliable, scalable and flexible, and can effectively meet the requirements of elastic resource provisioning in large-scale cloud environment.
Although demonstrated efficient in experiments, our resource provisioning approach has several limitations which we plan to address in our future work. First, it has not considered the impact of the local distribution of the stored data, which may result in a large number of IO load pressure. What's more, our resource provisioning method does not pay much attention to the relationship between tasks, there is no specific optimization for this kind of tasks.

Acknowledgements

This work is supported by the fund of the State Key Laboratory of Software Development Environment under Grant No. SKLSDE-2016ZX-13, the Open Research Fund of The Academy of Satellite Application under Grant No. Y20A-E03, the Open Project Program of National Engineering Research Center for Science & Technology Resources Sharing Service (Beihang University), the National Natural Science Foundation of China under Grant No. 61370059, Beijing Natural Science Foundation under Grant No. 4152030 and the Fundamental Research Funds for the Central Universities under Grant NO. YWF-14-JSJXY-14.

References


