# Auto-Scaling Web Applications in Hybrid Cloud Based on Docker

Yunchun Li
Sino-German Joint Software Institute,
School of Computer Science and Engineering,
Beihang University, Beijing, China
lych@buaa.edu.cn

Yumeng Xia
Sino-German Joint Software Institute,
School of Computer Science and Engineering,
Beihang University, Beijing, China
xym@buaa.edu.cn

***Abstract*—The rapid development of web applications has greatly improved people's work and life. Cloud computing can provide resources on-demand, hybrid cloud is a popular solution. In this paper, we design a platform which can auto-scale web applications in hybrid cloud based on docker. We use docker container technology to deploy web applications on hybrid cloud hosts, and add or delete docker containers to adjust web applications resource. We build a hybrid scheduling controller, and use a combination of prediction and reaction algorithms to scale up or scale down. The experimental results show that docker containers can be deployed in private and public cloud, and our system can dynamically adjust the number of containers in a few seconds in order to meet the resource requirements.**

***Keywords—Hybrid cloud; Web application; Auto-scaling; Docker; Prediction***

## I. INTRODUCTION

Web applications, due to their convenience, have become the main entrance for accessing the Internet services, as occupying a larger share in all the internet application. The requests' number of many Web applications is normally small, but sometimes may increase rapidly. To meet applications peak demand, one could purchase more servers, which may lead to higher waste of resources when the application requirements decline.

Cloud computing is an on-demand resource model to deliver high performance and reliability. Cloud computing has three main forms: private clouds, public clouds, and hybrids clouds. A hybrid cloud includes multiple private cloud or public cloud platforms provided by multiple cloud providers.

Hybrid cloud can dynamically increase the response and processing capacity of the Web applications by dynamically extending the Web applications to the nearly unlimited public cloud resources. After the peak of the burst access requests, the public cloud resources can be released dynamically, thus effectively dealing with the sudden increase in requests quantity of the scene, taking into account the scalability provided by the applications and low cost.

Docker is a lightweight virtualization technology. Using docker, we can package web applications as docker images, and run them on any cloud hosts with docker running environment. The web applications' deployment become more convenient and flexible. We needn't rely on a single provider, and we can migrate web applications between different cloud providers easily, prevent lock-in by a cloud provider, and take advantage of a competitive pricing market.

In the paper, we use Docker containers to deploy and auto-scale web applications. First, we will review the related work. Then we design the architecture and introduce the algorithms. Finally, we provide an experimental evaluation of the prototype.

## II. RELATED WORK

### A. Use Docker for Deploying Application

Deploying web applications requires a runtime environment which includes web servers, language support, databases and other components, rather than the whole operation system. By using docker, we can package a web application and its running environment in docker images, and develope, test, ship and deploy it anywhere. Docker containers can start up more quickly and make use of resource more efficiently than VMs[1]. Docker's lightweight packaging and deploying of applications is an amazing function, and it is quickly being accepted by many companies[2].

Many researchers package a whole web application and its running environment in a single container, which is similar to deploying applications in VMs. Docker official suggested that one container runs a single service, compared to the fat container mode in which all services are placed in a container. It will lead to increased coupling between the containers. This coupling needs to use the docker run options to be accomplished. The contents of the container become independent and secure due to the partitioning of the entire structure by the containers. Each container can be flexibly upgraded or disassembled without impacting other container applications. Fig.1 shows the example of deploying a web application in three containers. At the same time, we can achieve the fine-grained resource scheduling according to the load of different application containers. When a container runs only one service, the size of a single image is small, and migration and scheduling expenses are small. In order to run efficiently, interdependent containers should run within the same host.

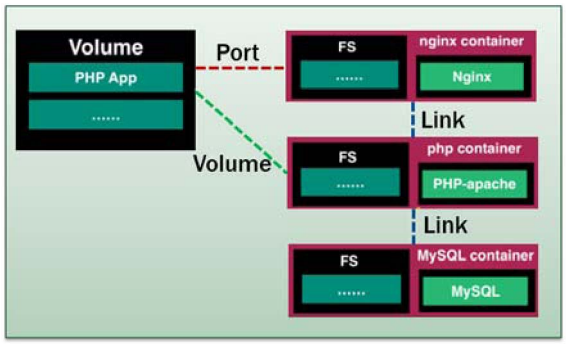


Fig. 1. Example of deploying a web application in three containers

### B. Scale Applications on Hybrid Cloud

Deploying and scaling the application in hybrid cloud is a hot cloud computing research area. Many researchers now are focusing on this field. Now, applications' scale in hybrid cloud is usually based on virtual machines (VMs). VMs support either or both of the following mechanisms to scale capacity: 1) horizontal scaling: more VM replicas are added on demand 2)vertical scaling: allocate more resources such as more CPU cores and more memories to an application's VM. Compared with the horizontal scaling, vertical scaling algorithm is more complex[3].

Docker containers are lighter than virtual machines, but the mechanism of scaling is similar. Horizontal scaling is easier and more efficient than vertical scaling.

We assume that there is a workload forecaster which can predict whether an application is becoming overloaded or not. It needs full consideration to auto-scale web applications in time and efficiently. There are two commonly used methods (i.e. proactive method and reactive method) to solve this problem. We predict following resource requirements of web applications based on historical data, and increase or decrease the allocation of resources in advance.

In [4] , a new on-line prediction system is proposed, which uses historical data to predict the number of requests based on machine learning techniques and time series analysis. Queuing theory and multi-objective optimization are used to determine the optimal number of VMs. [5] proposed a hybrid elastic controller based on control theory, which can dynamically adjust the required resources under the condition of uncertain workload, and have good robustness.

In this article, we will use docker to deploy web applications in hybrid clouds. An elastic controller is used in combination with the positive mode and the reactive mode to auto-scale web application containers.

## III. System Design

### A. Architecture Design

We designed the following system architecture. The goal is to quickly and automatically increase or decrease the docker application containers, to quickly adjust the web application resources to respond to the rapid changes in the load, without breaking QoS.

The overall structure of the system is shown in Fig.2, mainly including a load balancer, management server, and private docker registry.

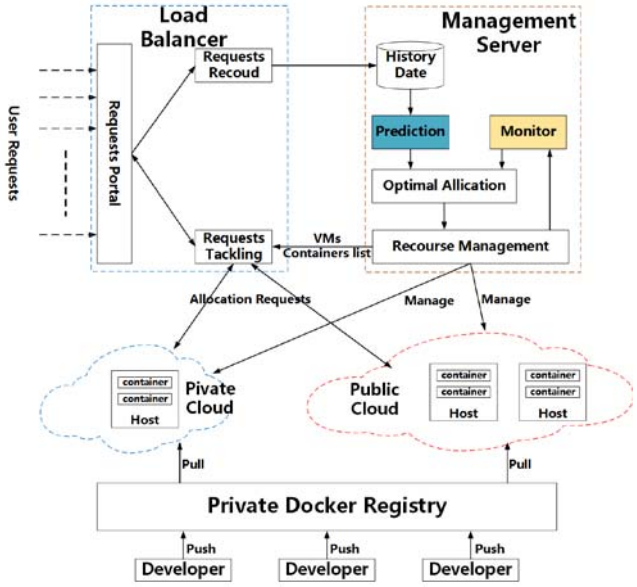


Fig. 2. Architecture of system

### B. Load Balancer

Load balancer is the entrance of system. It receives all users' requests, routes them to docker containers which have been installed web applications and then sends back the responses to users.

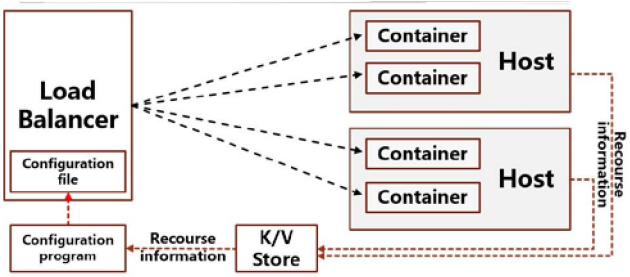


Fig. 3. Structure of Load Balancer

The system must be able to dynamically and automatically adjust the web application resource to meet the changing needs of users, which brings the change of the number of docker containers. Load balancer must update the resource information and adjust the load balancing strategy to meet the user's requests and the change in the number of containers in the case of varying workloads.

Obviously load balancer for web applications is very important, which requires load balancer to have a very good performance and robustness. We use nginx as a load balancer because of its good performance and stability. We also use *Keepalived* to provide simple and efficient high availability. We use the *K / V* store to store resource information, and use the configuration program to dynamically update the configuration of the load balancer using resource information

when system increases or decreases the web application containers[6].

One notable feature of hybrid clouds is the large variation in the geographical location of different cloud providers. There will be a significant delay in the access through the Internet which brings a great variation of response to the web applications deployed in hybrid cloud.

To address the problem above, we adopt a relatively simple balancing strategy. We route users' requests to corresponding web application containers based on their source IPs. For example, to access college applications, users outside the campus firstly visit the public cloud, while campus users first visit private clouds. This can bring a better access experience.

### C. Management Server

The management server mainly includes the prediction subsystem, monitoring subsystem and the resource management subsystem.

The prediction subsystem calculates the number of application containers needed by predicting the number of user requests in the next time period[7]. The main steps are given as follows.

1) Collecting request number of each time period as history data;

2) Analyzing the history data and predicting the request number of the next time period;

3) Calculating the required web application resources and container number.

Monitoring subsystem collects the current resource utilization of each container and host, and uses the data to decide whether the number of containers is needed to be added or reduced.

The resource management subsystem uses the data obtained by the prediction subsystem and monitoring subsystem, calculates the number of containers and hosts needed, and then decides whether the number of current containers and hosts is needed to be scaled up or scaled down. Resource management subsystem interacts with docker daemon with docker remote api, to start or stop the containers and keep the containers in a correct quantity.

### D. Private Docker Registry

Docker uses images to package applications, so developers can use docker images to run docker container in any docker cloud host which has been installed docker. Docker registry is a server-side application that stores some docker images and enables developers to distribute their docker images. Developers can ship and share images easily by pushing and pulling images. The official docker registry is docker hub. Everyone can pull images from it. However, pulling images from the docker hub will be very slow and may take a long time. If we build a private docker registry, we can significantly increase the pull and push speed.

By using docker technology, when we need to migrate web applications between private and public clouds, we only need to pull docker images that are needed, rather than having to move containers between the private and public clouds via the internet, in this way we can save a lot of time. We use nginx as the front-end proxy to ensure the security of our private docker registry, in which case internal users can pull and deploy images without authentication, and provide basic security authentication and HTTPS access for external users.

## IV. Algorithm Design

[8] proposed a classification of auto-scaling techniques for applications in cloud environments into five main categories: static threshold-based rules, control theory, reinforcement learning, queuing theory and time series analysis.

In this article, we use the time series analysis to predict the demand for application resources, set the hosts and containers resource threshold-based rules, and monitor them. We build a hybrid scheduling controller, use a combination of prediction and response algorithms to scale up or scale down containers and hosts.

### A. Prediction Model

The prediction model is used to predict the number of user requests and calculate the number of containers and hosts required.

In order to predict the web application requests number, we define web request numbers as time series: $\left\{ X_{(t)}; t \in T \right\}$ . $T$ is the index of the time period.

Predictive problems can be defined as follows: given the history data $\left( X_{(t-i)}, X_{(t-i+1)}, ... X_{(t-1)} \right)$ , predict the next time period of data $X_{(t)}$ . We use autoregressive moving average method [9][10]. The equation is :

$$X_{(t)} = \sum_{i=1}^{N} \omega_i X_{(t-i)} + \varepsilon_{(t)} \tag{1}$$

where $\omega_{(i)}$ is the weight of $X_{(t-i)}$ , and $N$ is the number of related time periods. Not too many related time periods are needed, and the top key terms that determine the predicted value should be selected.

We calculate the correlation coefficient between $X_{(t)}$ and $X_{(t-1)}$ by

$$\rho_{t,t-i} = \frac{E\left\{ \left[ X_{\{t\}} - \mu \right] \left[ X_{(t-i)} - \mu \right] \right\}}{\sigma_{(t)} \sigma_{(t-i)}} \tag{2}$$

Then we chose the top $K$ high correlation coefficient items to improve regression equation as follows:

$$x_{(t)} = \sum_{t'=1}^{K} \omega_{t'} X_{(t')} + \varepsilon_{(t)} \tag{3}$$

We use this method to predict workload of Beihang SNS website. Fig.4 shows the predicted request number compared to the actual request number.

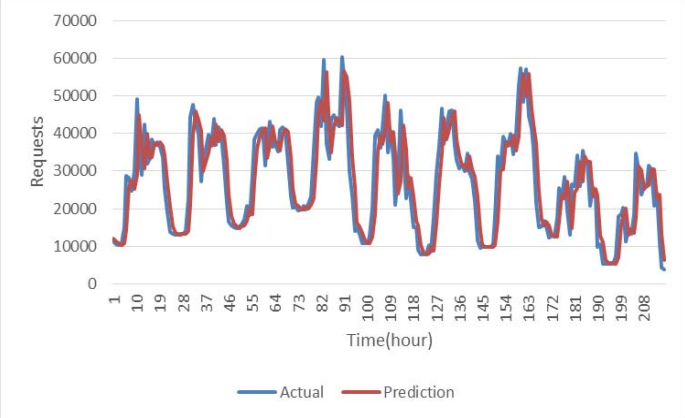


Fig. 4. Actual Workload vs Predicted Workload

With the predicted request number, next step is to estimate the number of containers on demand. We adjust the docker containers runtime constrains resource and test the number (*f*) of requests that a single container can handle per second. The *r* is the predicted request number, the container number $N_p$ is:

$$N_p = r / f \tag{4}$$

As we defined the container resources (CPU, memory), it is easy to calculate the number of required hosts.

### B. Reactive Model

There may be a bias in the number of user requests prediction, so we need a reactive model to adjust the resource allocation. We set the host and container resource utilization upper threshold, the monitoring subsystem regularly collects every host and container resource utilization periodically. If the resource utilization of some containers exceeds the given upper threshold $T_{threshold}$, we need to start some containers and add them to the load balancer list. [11]give a simple algorithm to reactive scaling containers, the details are described by the following pseudo procedure:

1. function reactive ()
2. $N_{exceed} \leftarrow 0, N_{reactive} \leftarrow 0$
3. for container *i* in all running containers $N_{instance}$
4. if $(R_i >= T_{threshold})$ then
5. $N_{exceed}++$
6. end if
7. end for
8. $N_{reactive} \leftarrow \lceil N_{exceed} * (1 - T_{threshold}) / T_{thresshold} \rceil$
9. return $N_{reactive}$
10. end function

The host reactive model algorithm is similar to the container algorithm. Because the host preparation time is longer than the container, the upper threshold of the host should be lower than the container.

### C. Scaling Algorithm

In order to meet the user's experience, scaling up should adjust the number of containers fast enough. When the Prediction model or the reactive model decides to increase the number of containers, management server just starts more containers and adds them to the load balancer immediately.

However, it should not scale down the container number too early, otherwise it may cause oscillations in the number of containers if users' requests increase quickly just after scaling down[12]. Scaling down should only occur when the web application needn't so many containers any more in the near future. We only scale down the container when the number of containers in the prediction model is less than the number of containers currently running in *k* consecutive periods.

The host scaling algorithm should be similar to the container scaling algorithm. There are two points to note: First, in order to reduce costs, we should give priority to the use of private cloud resources, and give priority to the release of public cloud resources. The other point is that, in order to improve the efficiency of web applications, containers with connection relationships should be scheduled on the same host.

## V. Experimental Evalution

In this section, we simulate three different scenarios and monitor the container numbers. We have deployed the SNS web application UCenter Home in the docker container, using the nginx container and the php container, and the database uses a specialized mysql server. In order to simplify the experiment, we use two cloud platforms, and use one virtual machine per cloud platform.

In experiment 1, the workload has three stages, in the first stage workload grows linearly, in the second stage the workload maintains stable, in the third stage workload declines linearly. We can see that the numbers of containers can change quickly as workload changes. When the private cloud resources are insufficient, the public cloud starts the containers, and when the workload drops, stop the public cloud containers prior.

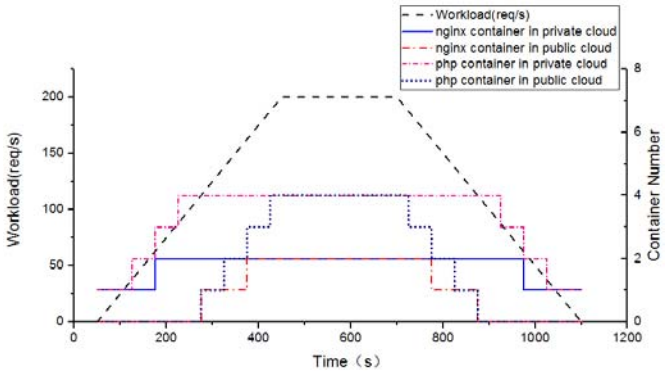


Fig. 5. Containers' numbers change as workload changes

Experiment 2 shows that when the workload changes rapidly in a short time, the numbers of containers can remain stable.

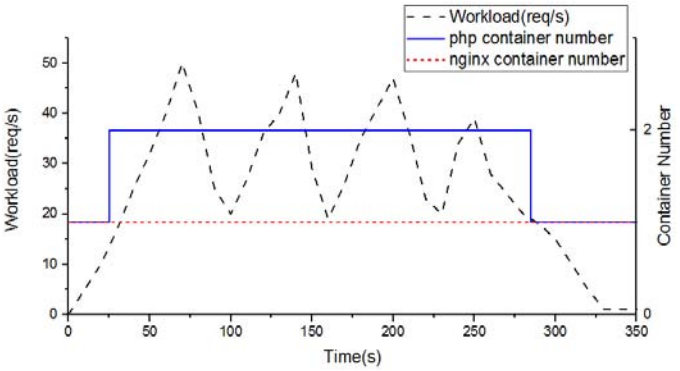


Fig. 6. Containers' numbers keep stable when workload shakes

In experiment 3, we selected some of the data in Fig.1 and scale it down. Our results show that our system can handle real workload.

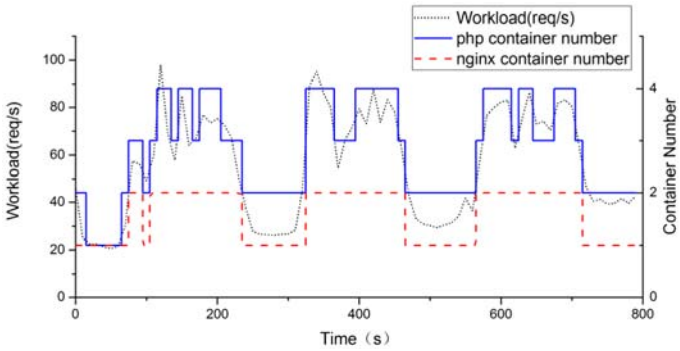


Fig. 7. Containers' numbers can change with real workload

## VI. Conclusion & Future Works

In this paper, we use docker to deploy web applications in hybrid cloud, and design an auto-scaling system. The experiments show our system can dynamically scale-up or scale-down the number of containers in a few seconds to meet the web applications' requirement. The consider of more types of applications to extend this work will be conducted in the future.

## Acknowledgment

This work is partly supported by the National High-tech R&D Program ("863" Program) of China (No.2015AA01A202) and research project of Beijing Municipal Education Commission.

## References

[1] Felter, W., Ferreira, A., Rajamony, R., Rubio and J. "An updated performance comparison of virtual machines and linux containers"technology, pp. 28-32, 2014

[2] Merkel, Dirk. "Docker: lightweight linux containers for consistent development and deployment." Linux Journal 2014.239 (2014): 2.

[3] Guo, Tian, et al. "Seagull: intelligent cloud bursting for enterprise applications." Presented as part of the 2012 USENIX Annual Technical Conference (USENIX ATC 12). 2012.

[4] Jiang J, Lu J, Zhang G, et al. Optimal cloud resource auto-scaling for web applications[C]//Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on. IEEE, 2013: 58-65.

[5] Jamshidi P, Ahmad A, Pahl C. Autonomic resource provisioning for cloud-based software[C]//Proceedings of the 9th international symposium on software engineering for adaptive and self-managing systems. ACM, 2014: 95-104.

[6] https://github.com/kelseyhightower/confd

[7] Jiang, Jing, et al. "Optimal cloud resource auto-scaling for web applications." Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on. IEEE, 2013.

[8] Lorido-Botran, Tania, Jose Miguel-Alonso, and Jose A. Lozano. "A review of auto-scaling techniques for elastic applications in cloud environments." Journal of Grid Computing 12.4 (2014): 559-592.

[9] Roy N, Dubey A, Gokhale A. Efficient autoscaling in the cloud using predictive models for workload forecasting[C]//Cloud Computing (CLOUD), 2011 IEEE International Conference on. IEEE, 2011: 500-507.

[10] Calheiros, Rodrigo N., et al. "Workload prediction using arima model and its impact on cloud applications' qos." IEEE Transactions on Cloud Computing 3.4 (2015): 449-458.

[11] Kan, Chuanqi. "DoCloud: An elastic cloud platform for Web applications based on Docker." 2016 18th International Conference on Advanced Communication Technology (ICACT). IEEE, 2016.