# DoCloud: An Elastic Cloud Platform for Web Applications Based on Docker

Chuanqi Kan*
*School of Electronic Information and Electrical Engineering
Shanghai Jiao Tong University, 800 Dongchuan Road, Shanghai, China
**kanchuanqi@gmail.com**

***Abstract*—Internet is growing at an alarming rate, and Web applications have permeated every aspect of people's life. Cloud computing provides a powerful computing model that allows users to access resources on-demand and pay as they use. Cloud computing attracts an increasing number of developers to migrate their Web applications to cloud platforms. Cloud platforms should provide elasticity ability to change the amount of resources allocated to a Web application in order to meet the actual varying demands because of the changing workload. In this paper, we design and implement DoCloud which is an elastic cloud platform based on Docker. In DoCloud, we adopt adding or removing Docker containers to change a Web application's resource and we build a hybrid elasticity controller that incorporates proactive model and reactive model for scale out coupled with proactive model for scale in. Our experiments show that DoCloud can dynamically allocate resources to applications within seconds and maintain higher resource utilization in a single container.**

***Keywords—cloud computing; autoscale; Docker; elasticity; hybrid controller***

## I. INTRODUCTION

Nowadays, with the concept of SaaS (software as a service), Web applications have developed a lot, many companies such as Google, Amazon etc. have achieved great success from Web applications. Web application providers should keep the application meeting the quality of service (QoS) requirements specified in the SLA agreements. The load of Web application usually vary drastically along with time. Flash crowds are also very common in today's Web Applications world. Figure 1 shows workload logs of the FIFA 1998 world cup website in the number of incoming requests from day50 to day57. If we maintain sufficient resources to meet peak requirements can be costly, which will increase developers' cost. Conversely, if the developers cuts costs by maintaining only minimal or medium computing resources, there will not be sufficient resources to meet peak requirements and cause bad performance which may lead to losing customers. Cloud computing is an on-demand computing model with a usage-based payment structure. With the help of cloud computing, developers can scale up or scale down the applications' resource manually or by APIs provided by cloud platform within hours or minutes. Autoscaling supported in Cloud computing can solve this problem totally. Autonomous elastic cloud dynamically allocate resources according to the current actual load. When load of Web applications grows up, elastic cloud automatically add more computing resource to the applications and reduce the computing resource while the load drops.

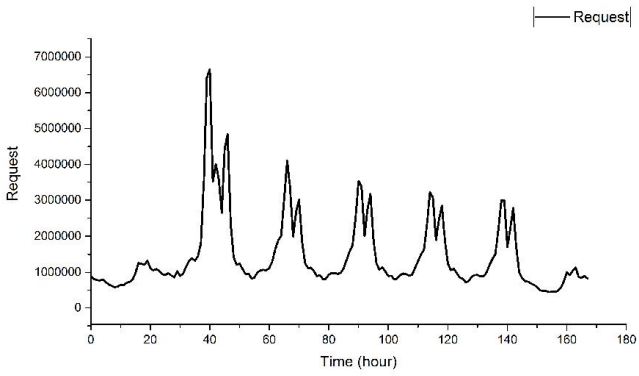



Fig.1 Traces Logs of World Cup Soccer 1998

Elastic cloud is a hot cloud computing research area, many researchers now are focusing on this thesis. Now, elastic cloud is usually based on virtual machines (VMs). VM is too heavy for Web applications, because all need of Web applications is applications' running environment which includes Web server (Apache, Nginx), language support, databases and other components, not the whole guest operation system in VMs. In this point, Deploying Web applications in VMs results in waste of resource and reduced performance. In more and more "flash purchase" scenarios, clients' requests suddenly surge, this require that elastic cloud should scale up resource within seconds to avoid breaking QoS. Elastic cloud based on VMs usually can't achieve this goal.

Docker is a new lightweight virtualization technology. With the help of Docker, we can package a Web application and all its running environment into one standardized unit for software developing, testing, shipping and deploying. Docker containers running on the same host share the same linux kernel so containers can start up instantly and make more efficient use of resource.

In this paper, we will use Docker container instead of VM as the unit of resource adjustment to design and implement an elastic cloud for Web applications. First, we will review the relative work. Then we will focus on the design of architecture and elasticity controller which can ensure the efficiency and scalability. In section IV, we use Tsung to simulate different types of load to provide an experimental evaluation of the prototype. In the last section, we draw the conclusion and point out the future research directions.

## II. Related Work

### A. Docker Containers vs Virtual Machines

Fig.2 shows the difference between Docker container and virtual machine on architecture. Each VM includes the application, binaries, libraries and an entire guest OS which cost a lot of CPU, memory and storage. In a container, there are only the application and its dependencies. Container runs as an isolated process in userspace on the host OS. Container can start up within 2 or 3 seconds while VMs' startup cost minutes, this is very suitable for handling flash crowds. In [1], researchers from IBM compare the performance of virtual machines with Docker containers. They use a suite of workloads that stress the CPU, memory, storage and networking resources, and the results show that containers result in equal or better performance than VM in almost all cases.

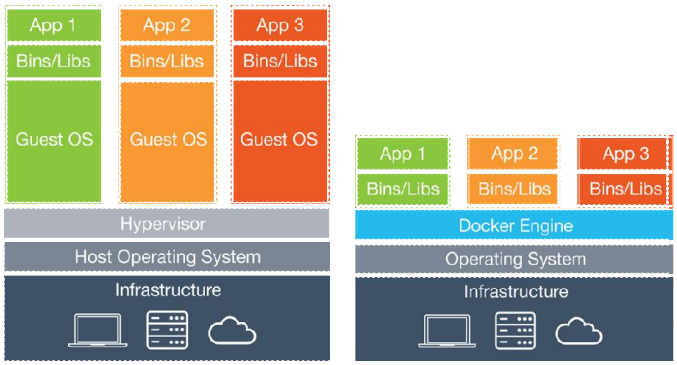



Fig.2 VM vs Docker Container [2]

### B. Elastic Cloud

There has been a large amount of research on elastic cloud, but almost all of them are based on VMs. Reviewing these work will have a certain reference value in the design of DoCloud. In [3, 4], researchers adopt horizontal scalability to adjust the amount of resource allocated to a specific application. Horizontal scalability means changing the number of instances of resource unit (VMs). Cloud platforms using this method need a load balancer component to route the requests to all the instances. Horizontal scalability can achieve high availability of Web applications, because one instance breaks down, others can take charge of the requests belonging to it. On the contrary, [5, 6] use vertical scalability to build elastic clouds. Vertical scalability adds or reduces the amount of specific resource such as CPU, memory and network within a single instance (VM). Vertical scalability can dynamically adapt the resource more quickly, but it need to interact with hypervisor with higher authority. Higher authority means more complex and insecure.

Elasticity controller is the core of elastic cloud. How to scale the resource just in time and efficiently calls for our premier consideration. There are two frequently-used approaches which are proactive approach and reactive approach to solve this problem. With the first approach, elasticity controller will initiatively predict the demands of resource in the near future based on historical data, and allocate or deallocate resource in advance. [7] uses an online prediction system which includes a fast analytical predictor and an adaptive machine learning based predictor to solve the translation problem from service-level metrics to resource-level metrics. [8] presents a resource prediction model based on double exponential smoothing. The reactive approach is based on threshold-based rules set by application developer. When the conditions are reached, the actions of resource adjustment will take place. Amazon applies the reactive approach to its EC2, developers can configure thresholds of resource utilization to let EC2 scale out/in automatically.

In this paper, we will apply horizontal scalability to DoCloud and build an hybrid elasticity controller incorporating proactive approach and reactive approach.

## III. System Design

### A. Architecture Design

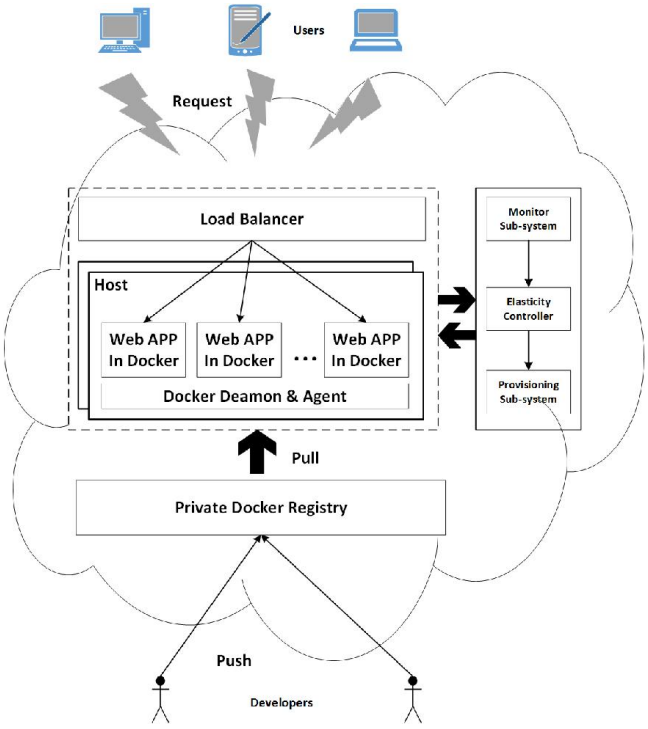



Fig.3 Architecture of DoCloud

We design DoCloud to let Web applications automatically adjust to the varying load without breaking QoS by growing or shrinking the amount of Docker containers on demand quickly. Figure 3 shows the general architecture of DoCloud. The architecture design includes a load balancer, a number of web application Docker containers, a monitor sub-system, and a provisioning sub-system with an elasticity controller. In DoCloud, we also consider the requirement of frequent upgrades of Web applications, because more and more developers are using agile software development to develop Web applications. We add a private Docker registry to DoCloud, which will make developers feel convenient to ship, deploy and upgrade Web applications in DoCloud. And also DoCloud provides hot-upgrade function to make the application still available without downtime during upgrading the application.

### B. Load Balancer

Load balancer is the entrance of a Web application, it receive all incoming requests, route them to real application servers in containers and then send back the responses to clients. It can be seen that load balancer is very important to Web applications, and this requires the load balancer must have good performance and robustness. We use HAProxy as the load balancer in DoCloud because of its first-class performance and stability. We also use keepalived to provide simple and robust facilities for high-availability. Keepalived maintains two running HAProxy instances, when master instance failed the backup one will take in charge. The structure of load balancer is shown in Figure 4. Confd [9] is adopted to automatically update HAProxy's configuration when DoCloud dynamically adds or removes containers.

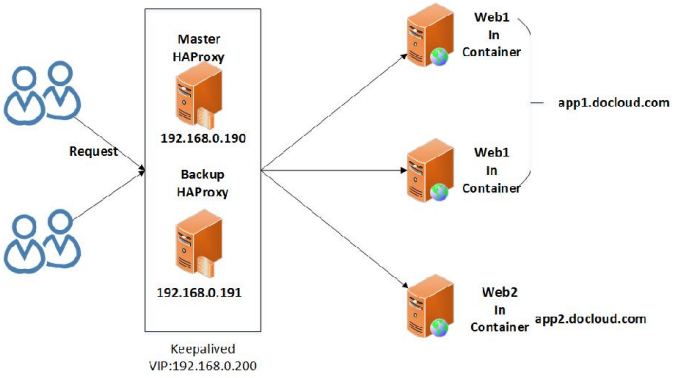



Fig. 4 Structure of Load Balancer

### C. Monitor and Provisioning Sub-systems

Monitor sub-system collects current resource utilization in every container from Docker daemons on each host via Docker remote RESTful API (GET /containers/ (id)/stats), and reactive model in elasticity controller will use this data to decide whether need add containers. Additionally, monitor sub-system gathers request rate (requests per second) from HAProxy stats page and store this data to database. Proactive model will use this data to predict the load in the near future.

Provisioning sub-system is also based on Docker remote RESTful API. Provision sub-system interact with Docker daemon to start or stop containers and maintains the exact number of containers which elasticity controller determines. When some container stops unexpectedly, provisioning sub-system should try to restart it or start a new container to replace it. Provisioning sub-system also take responsibility for hot-upgrade. After developers take upgrade actions, provisioning sub-system will start new containers from newer application image when elasticity controller decide to grow the number of containers, provisioning sub-system will first stop old containers that come from older image when elasticity controller determines to shrink the number of containers. Then the running old containers will be replaced by new container one by one (start a new container then stop one old container) until all the running containers come from the newer image.

### D. Private Docker Registry

Docker registry is a stateless, highly scalable server side application that stores and lets developers distribute Docker images. Developers can ship and share images easily via pushing and pulling images. The official Docker registry is Docker Hub (https://hub.docker.com), every developer can pull official images from it. Docker registry itself also is packaged as an image. We build a private Docker registry from that image to integrate image storage and deployment tightly into DoCloud for developers' convenience. We use an nginx as the frond-end proxy to provide basic authentication and HTTPS access for external developers. With the basic authentication only legal developers can push and pull images which may be business secrets. For internal components in DoCloud, the private Docker registry opens port 5000 without authentication, components can pull and deploy images more quickly via this port.

## IV. Elasticity Controller

In [10], Ahmed Ali-Eldin et al. explore nine different ways to build a hybrid controller with proactive model and reactive model. In DoCloud, the elasticity controller incorporates proactive model and reactive model for scale out and uses proactive model for scale in.

### A. Proactive Model

Proactive model will estimate the incoming workload (requests per second) after a short period $\Delta T$ and then translate the workload to the number of containers. To predict the workload in the near future, we use a second order ARMA (autoregressive moving average) method [11]. The equation is

$$y_{t+1} = \beta \times y_t + \gamma \times y_{t-1} + (1-(\beta+\gamma)) \times y_{t-2} \quad (1)$$

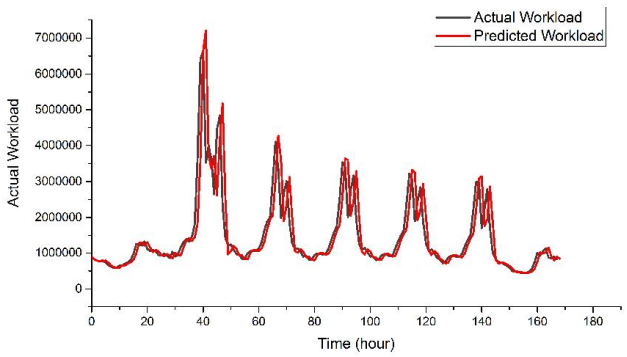



Fig.5 Predicted Workload vs Actual Workload

We use this method to predict workload shown in figure 1, and figure 5 shows the predicted workload compared to the actual workload. With the predicted workload, next step is to estimate the amount of containers on demand. We think the number is in direct proportion to the workload, so we translate the predicted workload to the number of containers by the following equation. The $f$ in the equation can be simply explained as the number of requests a single container can

handle per second. The $f$ can be set manually or set by reactive model. The $l_t$ is the predicted workload.

$$N^t_{proactive} = l_t / f \quad (2)$$

### B. Reactive Model

Developers set upper threshold for some resource utilization, the monitor sub-system collect the resource utilization data in every container periodically. If some containers' resource utilization above the given upper threshold $T_{upper}$ , some containers will be started and added to the load balancer. The detailed reactive model algorithm is described by the following pseudo procedure.

Algorithm 1 Reactive Model Scaling

```
1. function reactiveScaling()
2.   N_exceed ← 0 ,  N_reactive ← 0
3.     for container i in all running containers N_instance do
4.       if( R_i >= T_upper ) then
5.         N_exceed ++
6.       end if
7.     end for
8.     N_reactive ← ⌈ N_exceed × (1 − T_upper) / T_upper ⌉
9.     return N_reactive
10. end function
```

Algorithm 1 output $N_{reactive}$ how many containers DoCloud should add according to reactive model.

### C. Scaling Algorithm

While elasticity controller adjusts the amount of containers, scale out should be quick enough to avoid breaking the QoS and affecting user experience. When proactive or reactive model determines to increase the number of containers, elasticity controller will invoke provisioning sub-system to start more containers immediately.

On the other hand, scale in should not be premature, otherwise it may cause oscillations in the number of containers if clients' requests flood in quickly just after scale in takes place. Scale in should only occur when the Web application does not need the containers any more in the near future. In our elasticity controller, only during the following continuous $k$ periods, the numbers of containers predicted by the proactive model are all below current running containers, then some containers will be stopped.

The total scaling algorithm in the elasticity controller is shown as the following pseudo procedure. The algorithm outputs the total containers Web application should have during the next period. The elasticity period should be set short but longer than the container's startup time, this can make DoCloud sensitive to flash crowds.

Algorithm 2 Total Scaling

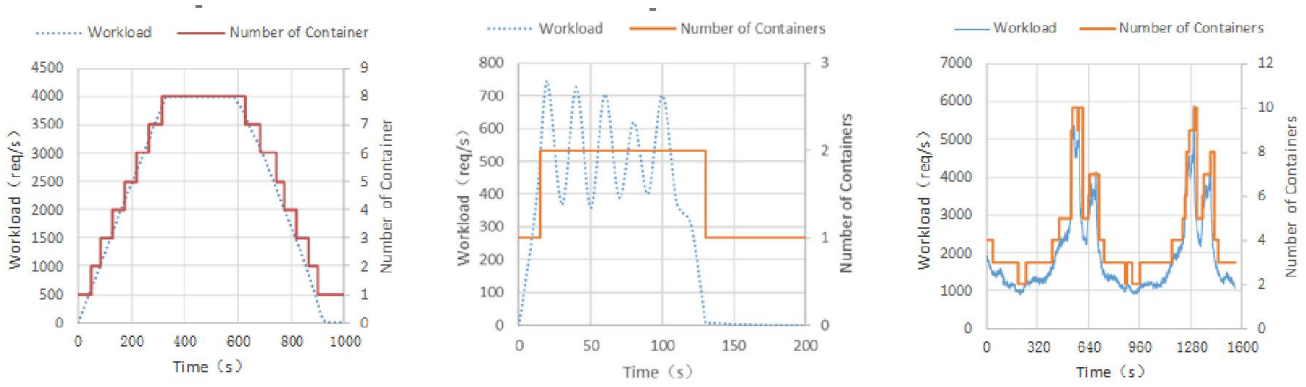
```
1.  lastTimes ← 0  // global variable for delay scale in
2.  function totalScaling( N_instance, k )
3.  // N_instance is the running containers now
4.    N_proactive ← proactiveScaling()
5.    N_reactive ← reactiveScaling()
6.    if N_reactive > 0 then
7.      if f in proactiveScaling() is not set
8.        f ← (R_t + R_{t−1}) / 2  // R_t means load at t period
9.      end if
10.     lastTimes ← 0
11.     return N_reactive + Max(N_instance, N_proactive)
12.   else if N_proactive >= N_instance
13.     lastTimes ← 0
14.     return N_proactive
15.   else if lastTimes >= k  // delay scale in
16.     lastTimes ← 0
17.     return N_proactive
18.   else
19.     lastTimes ++
20.   end if
21. end function
```

## V. Experimental Evaluation

In this section, we use Tsung to simulate three types of load in different scenarios and monitor the number of containers during the experiments. In the experiments, we set the prediction period $\Delta T$ =5s to take advantage of container's short startup time, and with this setting DoCloud can adapt containers to the changing load more quickly. We set the upper threshold $T_{upper}$ =0.8 for CPU and memory utilization. The results are shown in figure 6.

In experiment 1, the load has three stages, on the first stage the load grows smoothly, on the middle stage load keeps stable and on the last stage load drops slowly. We can see the number of containers varies quickly with the changing load (within seconds). In experiment 2, Tsung produces the shaking load and DoCloud does not decrease containers hastily until the load tends to be stable. In experiment 3, we simulate the real load shown in figure 1. To shorten the time of the experiment and narrow the gap in the Internet between 1998 and today, we choose part of the load and compress two minutes to one second. We also scale down the original data in consideration of the capability of our experimental equipment. The result shows that DoCloud can handle real workload including flash crowds.

We also evaluate the resource utilizations in a single container during the experiment 1, and figure 7 shows the results. From the results, we can find that DoCloud can maintain the resource utilizations of containers to a high level, which will ensure high efficiency of resources on the hosts.




(a) Experiment 1 (b) Experiment 2 (c) Experiment 3

Fig 6 The Number of Containers vs Changing Load

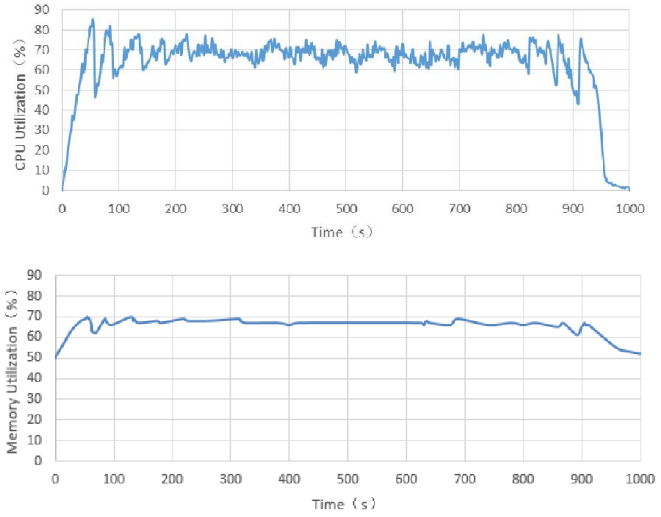



Fig. 7 Resource Utilizations

During another experiment 1, we perform a hot-upgrade operation, the number of new containers grows step by step as shown in figure 8.

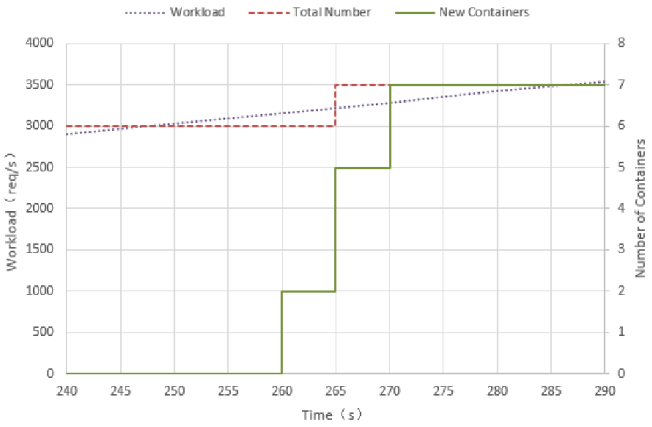



Fig. 8 Hot-Upgrade during Experiment 1

## VI. Conclusion

In this paper, we design and implement an elastic cloud platform for Web applications based on Docker and we build a hybrid elasticity controller to dynamically grow or shrink the number of containers within seconds. The experiments show DoCloud has good efficiency and scalability. For developers' convenience, we integrate a private Docker registry into DoCloud. DoCloud also supports hot-upgrade in consideration of high availability of Web applications.

## References

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

[2] Docker: https://www.docker.com/what-docker .

[3] Li, Yunchun, and Cheng Lv. "The Elastic Cloud Platform for the Large-Scale Domain Name System." Practical Applications of Intelligent Systems. Springer Berlin Heidelberg, 2014. 305-316.

[4] Tighe, Michael, and Matthias Bauer. "Integrating cloud application autoscaling with dynamic VM allocation." Network Operations and Management Symposium (NOMS), 2014 IEEE. IEEE, 2014.

[5] Shen, Z., Subbiah, S., Gu, X., & Wilkes, J. (2011, October). Cloudscale: elastic resource scaling for multi-tenant cloud systems. In Proceedings of the 2nd ACM Symposium on Cloud Computing (p. 5). ACM.

[6] Blagodurov, Sergey, et al. "Maximizing server utilization while meeting critical SLAs via weight-based collocation management." Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on. IEEE, 2013.

[7] Reig, G., Alonso, J., & Guitart, J. (2010, July). Prediction of job resource requirements for deadline schedulers to manage high-level SLAs on the cloud. In Network Computing and Applications (NCA), 2010 9th IEEE International Symposium on (pp. 162-167). IEEE.

[8] Huang, J., Li, C., & Yu, J. (2012, April). Resource prediction based on double exponential smoothing in cloud computing. In Consumer Electronics, Communications and Networks (CECNet), 2012 2nd International Conference on (pp. 2056-2060). IEEE.

[9] Confd: https://github.com/kelseyhightower/confd.

[10] Ali-Eldin, A., Tordsson, J., & Elmroth, E. (2012, April). An adaptive hybrid elasticity controller for cloud infrastructures. In Network Operations and Management Symposium (NOMS), 2012 IEEE (pp. 204-212). IEEE.

[11] Roy, N., Dubey, A., & Gokhale, A. (2011, July). Efficient autoscaling in the cloud using predictive models for workload forecasting. In Cloud Computing (CLOUD), 2011 IEEE International Conference on (pp. 500-507). IEEE.

**Chuanqi Kan** was born in Xuzhou city, China, on November 8, 1990. He received the B.E. degree from Xidian University in 2013. He is currently a M.S. candidate in School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, Shanghai, China. His research interests include adaptive cloud computing, light-weight virtualization, artificial intelligence and Big Data.