future internet

MDPI

Article

High-Performance Elastic Management for Cloud Containers Based on Predictive Message Scheduling

Chengxin Yan, Ningjiang Chen * and Zhang Shuo

School of Computer and Electronic Information, Guangxi University, Nanning 530004, China; yanchengxin313@163.com (Y.C.); zhs1992@126.com (Z.S.)

* Correspondence: chnj@gxu.edu.cn; Tel.: +86-139-7718-2589

Received: 2 October 2017; Accepted: 7 November 2017; Published: 28 November 2017

Abstract: Containerized data centers can improve the computational density of IaaS layers. This intensive high-concurrency environment has high requirements for message scheduling and container processing. In the paper, an elastically scalable strategy for cloud containers based on predictive message scheduling is introduced, in order to reduce the delay of messages and improve the response time of services and the utilization of container resources. According to the busy degree of different containers, a management strategy of multiple containers at message-granularity level is developed, which gives the containers better elasticity. The simulation results show that the proposed strategy improves service processing efficiency and reduces response latency compared with existing solutions.

Keywords: message scheduling; cloud container; elastic management; predictive strategy

1. Introduction

Container technology, represented typically by Docker (dotCloud, Inc., San Franciso, CA, USA), has been widely used by a wide range of sandbox environments to deploy applications based on operating systems with a time-sharing multiplexing mechanism. Containers can be started quickly, with little resource consumption. The combination of container and micro-service architectures provides an application-oriented lightweight infrastructure that can be extended by creating replicas of bottleneck components [1]. In container-based micro-service architectures, there are a great deal of log data, such as user behavior and system logs, which default to real-time consumption, and are not suitable for asynchronous processing; additionally, there exists the problem of message coupling and response delay [2]. Therefore, dynamic adjustment of message queues is studied in this paper. In the micro-service architecture, a user's events are achieved by calling a number of micro-services; therefore, event-based micro-services can be decoupled by optimizing the messaging mechanism. In the container-based cloud environment [3], existing research mostly focuses on job scheduling, ignoring the impact of message scheduling on container performance. In order to enhance the adaptive of optimization micro-services framework in containers, highly elastic management for cloud containers is researched in this paper. This paper proposes an elastically scalable strategy for containers in cloud system, based on container resource prediction and message queue mapping. By establishing a multi-channel message queue-based processing model to control the scale of the container cluster, we can achieve a decentralized control to address the dynamic expansion of container resources in cloud containers.

The paper is organized as follows: Section 2 describes the background of our work and the related work; Section 3 describes the framework of the elastically scalable model; Section 4 describes the policy of elastic scalable strategy for containers; Section 5 shows the implementation of the prototype; Section 6 gives the experimental results; and Section 7 draws the conclusions.

2. Background

To date, there have been certain container systems, such as Docker and Rocket, which are based on the CoreOS project (CoreOS, Inc., San Franciso, CA, USA), LXC, OpenVZ (SWsoft, Inc., Beijing, China), and so on. Container technology is a lightweight virtualization technology that hosts applications by using small, concise containers. Micro-services separate the overall framework according to the functional structure. A micro-service framework is constructed by a number of small application units and the corresponding micro-database unit. Containers and micro-services are widely used in cloud-based systems. Much research has discussed how to guarantee the quality of services in the context of cloud containers and micro-service architectures. Docker has a representative application for data center purposes and for the construction of micro-service architectures, so our work is based on Docker.

A framework named MMGreen [4] proposes a joint computing-plus-communication optimization framework exploiting virtualization technologies to cope with the demands of the highly fluctuating workload that characterizes this type of service. The idea of using real-time multi-processing and embedding Docker containers into the workflow is presented in [5], and three kinds of embedding strategies with different levels of granularity are used to verify the performance. After comparing the performances under different conditions, the results show that the coarse-grained shared embedding strategy has a better response speed. A strategy proposed in [6] setup waiting queues and release queues for real-time message based on the working state to achieve a timely response. The Docker container batch scheduling system, which uses a scheduler selection algorithm for scheduling operations, is given in [7]. The BPEL engine and Docker are combined to support a multi-tenant processing environment [8], which improves the performance of the system greatly. A resource management platform named DoCloud has been designed to improve the availability of the cloud cluster by dynamically changing the load of the containers [9]. Experiments in [10,11] resulted in an infrastructure for micro-services that has a non-negligible impact on micro-service performance, which is beneficial in terms of cost for both cloud vendors and developers. A workload prediction model for the cloud is proposed through optimized artificial neural networks [12], with a neural network and regression being used to predict the processor load [13].

Another type of work considers the size optimization of containers. Red Hat Enterprise Linux 7 Atomic Host (hereinafter referred to as Light Docker) only provides the components necessary to run Docker, removing some non-essential components to make the container lighter, and to effectively reduce the time required for system updates and start-up. A container integrated with HA (High Availability) middleware has been established, in which components in the scheduling scheme achieve high availability [14]. A distributed micro-service model based on Docker was designed, and an elastic controller for the distributed system implemented, in [15]. The feasibility of container technology as an edge computing platform is evaluated, and the experimental results prove that the strategy reduces the response time of services [16]. Docker and OpenStack are used to build a cloud service architecture that optimizes the performance of Docker [17].

In general, the achievements of the above work are to enhance the ability of the system to handle multiple tasks, or to reduce the size of the cloud necessary for deploying services. This paper works on the granularity of the message, and investigates the good elastic scalability of the container, in order to improve the overall performance in the high-concurrency message environment.

3. Elastically-Scalable Architecture for Cloud Containers

In this paper, core components of the elastically scalable model are made up of message processing components and elastic controller components. Compared with the original container processing architecture, a set of message queues are setup and mappings are created between message-queues and Docker containers. The granularity of elasticity is initialized by dividing messages to achieve elastic expansion of containers. The messages are transformed into elastic control components to complete

the flexible stretching of containers. This flexible architecture does not need to arrange messages and can eliminate large number of the complex HTTP (Hypertext Transfer Protocol) direct calls.

Elastic-Docker mainly includes a Queues Controller, Message_bus component, Elastic Controller and Docker monitor (V 1.13, dotCloud, Inc., San Franciso, CA, USA). Queues Controller, Message_bus, Elastic Controller are developed by our team. The Queues Controller classifies messages to form a number of message queues. Message_bus delivers the messages by publish/subscribe mechanism. Elastic Controller is to achieve the high-performance elastic management of containers. Docker monitor achieves the monitored data of Docker cluster, including the status of Docker containers, and the usage of CPU, memory, and network IO (Input Output). Figure 1 shows the architecture of the elastically-scalable model, called Elastic-Docker in the paper.

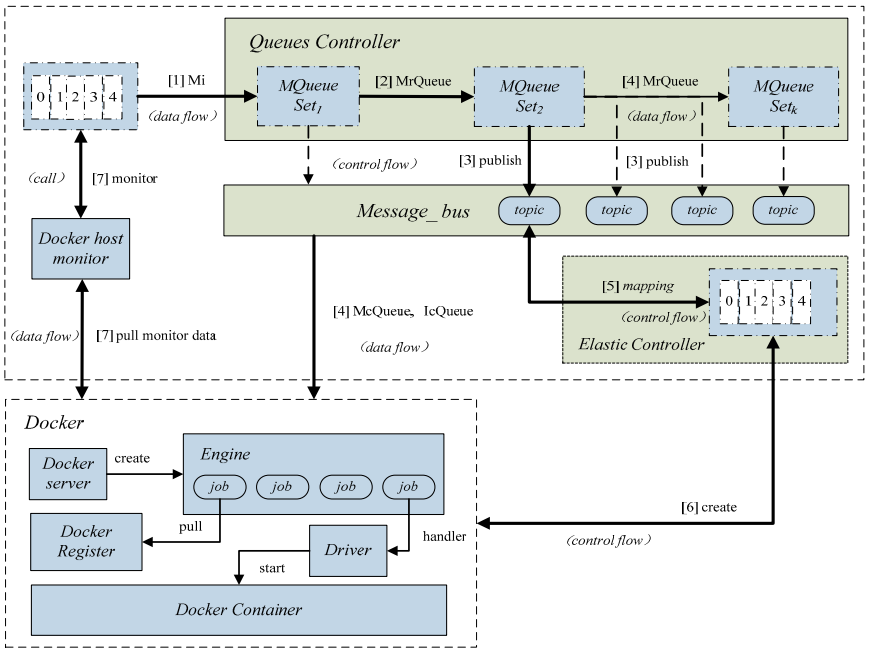

Figure 1. The architecture of Elastic-Docker.

Define Mi = <IDM, UserM, ContentM, Priority>, where IDM (ID of the Message) is the message number, UserM is the owner of message, ContentM represents the content of message, and *Priority* represents the priority of yjr message. Messages are dispatched and a number of message queues are created according to the content of messages. Then, ContentM and topic are compared by fuzzy matching to build the mappings. Messages with the same value of ContentM are mapped with containers, so the elastic management for containers is achieved based on message scheduling. MQ = {MSG, Map, Max} represents the message queue, MSG (Message Set Group) represents the queue set, *Map* represents the map value, and Max represents the queue length threshold. MQ (Message Queue) represents a set of queues defined as: $\mathrm{MQueueSet_k}\{\mathrm{MQ_1}, \mathrm{MQ_2}, \mathrm{MQ_3}, \ldots, \mathrm{MQ_i}\}$, where the range of k is in [2, k − 1].

In the case of high concurrency, users have higher requirements on the system response time and resource utilization. The message priority value is determined by resource utilization, user priority, and response time. The priority of message is calculated by multi-objective algorithm, and the Pareto optimal surface is searched according to the weight vector. $\mathbf{x} = [\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_D}]^{\mathbf{T}}$, D is the number of decision variables, the objective function expressed as $\mathrm{f_1}$, $\mathrm{f_2}$, ... , $\mathrm{f_M}$, and M is the target number. The weight is introduced by the message object and user's preference to dynamically adjust the degree of targets. $w_1, w_2, w_3 \ldots w_M$ represent the weight of the objective function, such as resource consumption, user priority, and response priority. Finally, the objective function $\mathrm{f_M(x)} = \mathrm{c_i x}$ is obtained.

The Queue Controller module classifies the messages. Then the Message_bus module uses the publish/subscribe mechanism and message queue for message delivery. Thus, message queues can be set up: the elements in the $MQueueSet_k$ queue set is combined with the message queues, and the messages in the queue are the smallest non-subdivided units. The messages are sorted by the value of priority to multiple MQ collections and, finally, a new $MQueueSet_k$ $\{MQ_1, MQ_2, MQ_3, \ldots, MQ_i\}$ is generated. The real-time messages are processed efficiently by the combination of Broker, Topic, Producer, and Consumer components.

4. Predictive Message Scheduling Strategy

In the micro-service architecture, the data is transmitted in the form of a message. Usually, the messages are executed across multiple micro-services. Therefore, message scheduling is more accurate for the micro-service level. Firstly, we establish a multi-channel model by classifying messages into k categories for k different micro-services, and then the message queues are created to classify messages. Thus, large-scale messages are dealt with by implementing message queue cache, as shown in Figure 2. And the symbols used are listed in a Table 1.

Table 1. Definition of the symbols used.

Symbols	Meaning	Symbols	Meaning
ContentM	Content of message	$X^{(0)}$	Original mount container sequence
topic	Topic of the queue	Y	Data row of the containers
k'	Represents the ID of container	a	Grey scale of the containers
k	Represents the ID of queue	B	Data matrix of the containers
m	Represents the matching degree	b	Endogenous control grey of the containers

4.1. Queued Messages Scheduling Based on Prediction

In the process of classification, a fuzzy matching strategy is used to filter the queue topics, to quickly determine the mapping relations of messages and micro-services. As fuzzy matching has greater flexibility and selectivity for cluster partitioning, the size of the container cluster can be adjusted by fuzzy matching, according to reasonable division of message topic. By the fuzzy matching principle, the content of message ContentM and the *topic* are compared to determine whether they are compliance.

Denote $ContentM \in F(topic)$, and $H(topic \cdot ContentM) = \frac{1}{n}\sum_{i=1}^{n} |\mu_{topic}(u_i) - \mu_{ContentM}(u_i)|$ is the characteristic function of topic. The function H (topic·ContentM) is used to calculate the matching degree of fuzzy sets, and $m = 1 - H$ (topic·ContentM) represents matching degree.

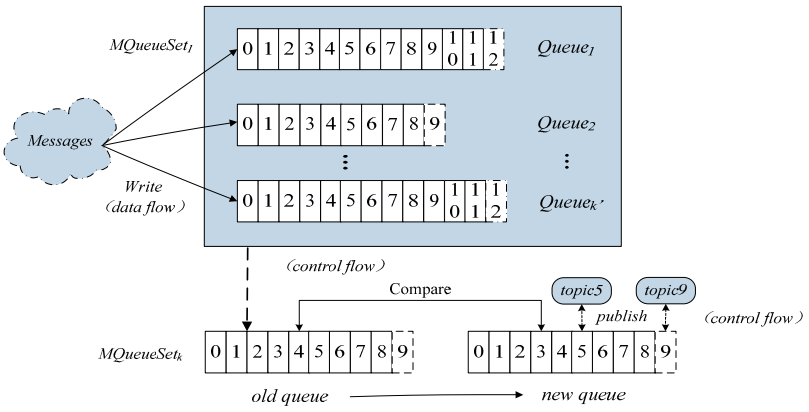

Figure 2. Model of predictive message scheduling.

In order to minimize resource consumption, this paper uses a self-response mode to determine the start time and duration. Since the Grey prediction model can predict future demand based on incomplete historical data, it is suitable to forecast the container demand. Firstly the original sequence $\mathbf{X^{(0)}}$ is formed by the original monitored data, and $\mathbf{X^{(1)}}$ is generated by accumulative generation method. It weakens the randomness of the original monitored data. The Grey model is established for transforming $\mathbf{X^{(1)}}$. The GM (1,1) model represents first-order, one-variable differential equation model to predict elastic expansion of container. According to the prediction results, the value of corresponding queue of micro-services can be dynamically adjusted to solve the problem of forecasting demand quantity of container. In this way, a multi-channel model is established to prepare for subsequent scaling prediction.

Suppose the containers mapping to n micro-services on host i, the usage of container is defined as $\{C_{i,1}^{(0)}, C_{i,2}^{(0)}, \ldots, C_{i,n}^{(0)}\}$. Then we build a GM (1,1) model by sequence $\mathbf{X^{(0)}}$. The acquisition of the real-time data is performed by the monitoring component, and the usage data of the containers in the set is then summed to obtain a new generated sequence $X^{(1)} = \{C_{i,1}^{(1)}, C_{i,2}^{(1)}, \ldots, C_{i,n}^{(1)}\}$, $C_{i,k'}^{(m)} = \sum_{j=1}^{k'} C_{i,j}^{(m-1)}, (k' = 1, 2, \ldots, n)$. In order to solve the development grey value and endogenous control grey value of container k′, the original data of container is used to establish the approximate differential equation and brought into the mean generation operator. Consequently, we can obtain an approximate differential equation for the number of containers:

$$\frac{dX^{(1)}}{d_t} + aX^{(1)} = b \tag{1}$$

where a is the Grey scale of the containers, b is the endogenous control grey of the containers.

Let $\hat{a} = \begin{pmatrix} a \\ b \end{pmatrix}$, $\mathbf{B}$ is the data matrix of the containers, $\mathbf{Y}$ is the data row of the containers, then we have $\mathbf{Y} = \mathbf{B}\hat{\mathbf{a}}$:

$$\hat{a} = \begin{bmatrix} ab \end{bmatrix}^T, Y = [C_{i,2}^{(0)}, C_{i,3}^{(0)}, C_{i,4}^{(0)} \ldots C_{i,n}^{(0)}]^T, \mathbf{B} = \begin{bmatrix} -\mathbf{0.5}[\mathbf{C_{i,1}^{(1)}} + \mathbf{C_{i,2}^{(1)}}] & \mathbf{1} \\ -\mathbf{0.5}[\mathbf{C_{i,1}^{(1)}} + \mathbf{C_{i,2}^{(1)}}] & \mathbf{1} \\ \vdots & \vdots \\ -\mathbf{0.5}[\mathbf{C_{i,n-1}^{(1)}} + \mathbf{C_{i,n-2}^{(1)}}] & \mathbf{1} \end{bmatrix} \tag{2}$$

The actual container usage for n micro-services are summed up according to Equation (1), and the differential equation of container assignment for the micro-service cluster is obtained. $C_{i,n}^{(0)} = a[-\frac{1}{2}(C_{i,n-1}^{(1)} + C_{i,n}^{(1)})] + b$ is solved to obtain the predictive value of the next container allocated for micro-services. A linear dynamic model is used to approximately generate the number of containers based on the new generated data, and the values of a and b are solved. The minimum multiplication value is obtained: $\hat{a} = [ab]^T = (B^T B)^{-1} B^T Y$. Then the variables are to be separated as follows:

$$\hat{C}_{i,k'+1}^{(1)} = (C_{i,1}^{(0)} - \frac{b}{a})e^{-ak'} + \frac{b}{a} \tag{3}$$

The sequence $\overset{\wedge(1)}{C}_{i,k'+1}$ is subtracted cumulatively, then the final distribution prediction model for containers is obtained, and the predicted value $\overset{\wedge*(0)}{C}_{i,k'+1}$ is obtained.

$$\overset{\wedge*(0)}{C}_{i,k'+1} = \overset{\wedge(1)}{C}_{i,k'+1} - \overset{\wedge(1)}{C}_{i,k'} = (C_{i,1}^{(0)} - \frac{b}{a})(1 - e^{a})e^{-ak'}, k' = 1, 2, \ldots, n \quad (3)$$

According to the prediction model, the length of the message queue and the number of queues that the micro-service belongs to are adjusted dynamically. The history of the corresponding queues are obtained and combined with the data obtained from message prediction. Based on the quantity value obtained by the above-mentioned prediction model, the new containers are started quickly and the temporarily unnecessary containers may be closed.

4.2. Elastic Management for Cloud Containers

The container demand forecasting module provides a direct basis for elastic control. Elastic Controller components send the predicted value $\overset{\wedge*(0)}{C}_{i,k'+1}$ to the Docker daemon, the driver component controls the generation and closure of particular containers, then synchronizes the length and the number of $MQueueSet_k$. Then according to the results of previous classification of message queues, $MQueueSet_k$ publishes messages to the micro-service and builds a mapping from the topic to the container. The elastic management model for cloud containers based on message scheduling is shown in Figure 3.

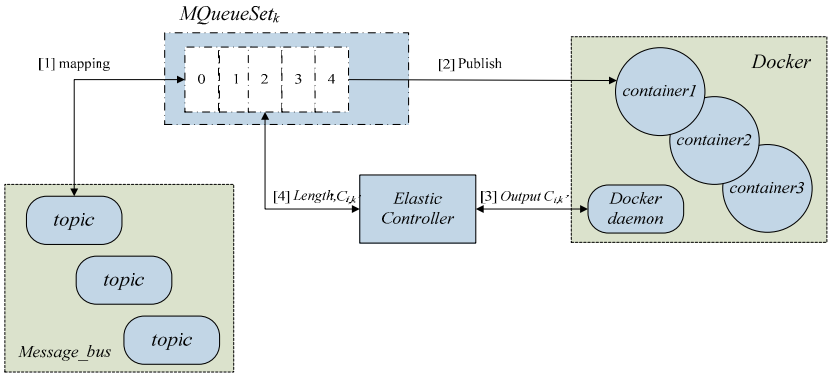

Figure 3. Elastic management model.

According to the Grey prediction model to determine the current demanded number of containers, the Docker daemon copies the services by the number of containers. The deviation rate is calculated by dynamically acquiring online monitoring values and predicted values of the container number: $\sigma = \frac{count_{actual} - count_{predict}}{count_{predict}} \times 100\%$. The threshold value is corrected by the calculation of the deviation rate to obtain the final threshold value. When the number of containers corresponding to the service is larger than the threshold, it means that the container is at full capacity and it is to select the predicted value as the threshold size when the predicted value is smaller. Then, a new container, ContainerM+1, is produced to respond the new created service. The corresponding message queue set for each micro-service is looped to maintain equalization of queue lengths. Figure 4 shows the process of elastic management for containers.

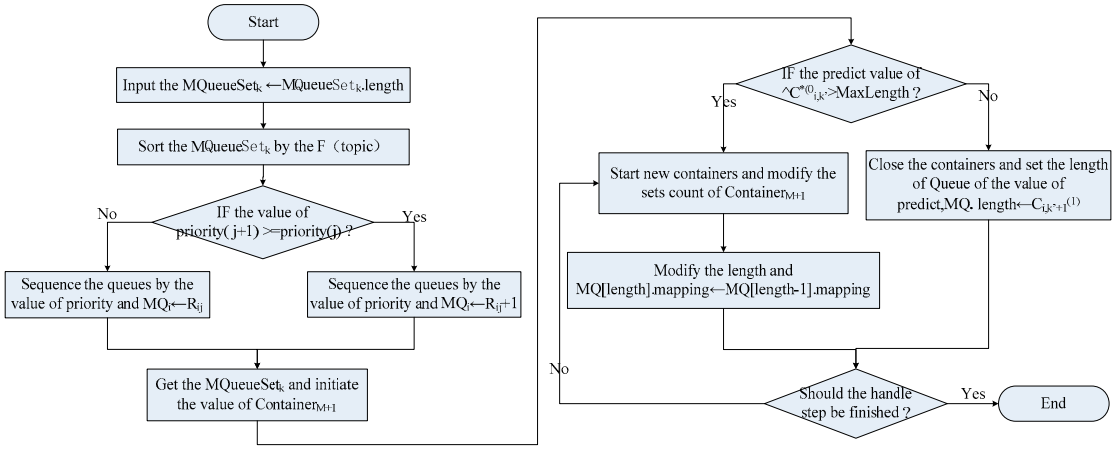

Figure 4. Flowchart of elastic management for containers.

Compared with the existed Docker systems lacking of message scheduling, the scalability of Elastic-Docker is more accurate for cloud containers. For example, Kubernetes and Swarm only choose appropriate physical nodes to run a new container, which cannot meet system expansion requirements under the micro-service framework. The main features of this work are as follow: predictive message scheduling is used as an effective solution to solve the problems of service optimization, and dynamic expansion of cluster systems is realized by adopting the flexible scalable scheme in the aspect of the flexibility of containers.

5. Prototype Design

The designed prototype system named Elastic-Docker is shown in Figure 5. Core software packages, such as docker-java.jar (V 1.0, dotCloud, Inc., San Franciso, CA, USA), docker-py.jar (V 1.0, dotCloud, Inc., San Franciso, CA, USA), Kafka_2.10.jar (V 2.10, Apache Software Foundation, Forest Hill, MA, USA), Zookeeper-3.3.4.jar (V 3.3.4, Google Inc., Mountain View, CA, USA), Zkclient-0.3.jar (V 0.3, Google Inc., Mountain View, CA, USA) and ganymed-ssh2-build210.jar, are used in the system. The third-party products include: Kafka middleware (V 0.10.2.1, Apache Software Foundation, Forest Hill, MA, USA), Mysql database (V 4.1, Mysql AB Inc., Stockholm, Sweden), Zookeeper (V 3.3.4, Google Inc., Mountain View, CA, USA), and the Docker container system (V 1.13, dotCloud, Inc., San Franciso, CA, USA). The Client module includes SSH2Client (V 4.0, NetSarang Computer Inc., Santa Clara, CA, USA), Zkclient (V 3.3.4, Google Inc., Mountain View, CA, USA), and Docker Client. Kafka (V 0.10.2.1, Apache Software Foundation, Forest Hill, MA, USA) is selected as the message bus, combined with the Elastic module and Queue module to finish the elastic management of Docker containers. SSH2 Monitor is developed in Python to achieve remote calls; the host CPU, memory, I/O, and other status information can be displayed on the interface. We developed the Queue module, Elastic module, andy SSH2 Monitor module in the prototype system.

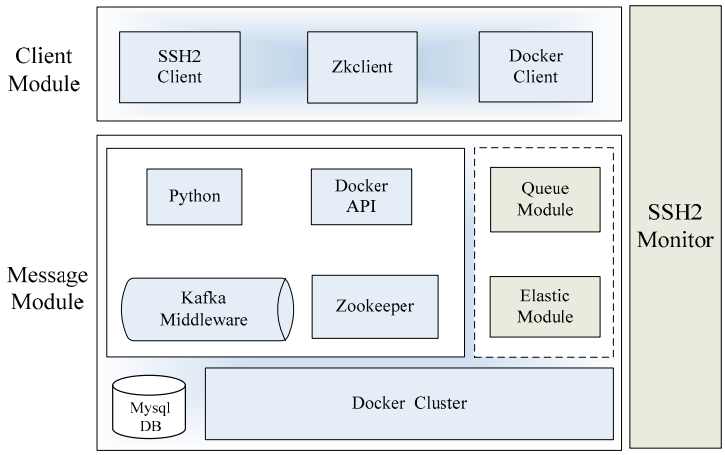

Figure 5. Prototype design of Elastic-Docker. SSH2: Secure Shell; API: Application Program Interface; DB: DataBase.

Various states and the information of containers are stored in system database monitored by the Docker API (Application Program Interface). The containers are used as the message processing part of the prototype system, and the Docker daemon is used to operate the entire Docker cluster. Queue Module is controlled by the implementation of the algorithm that completes the message processing, classification, and mapping operations. The Elastic module is used to achieve elastic expansion of container cluster by creating the mappings.

6. Experiments

The effect of elastic scalability for cloud containers has been evaluated from multi-aspect experiments. Twenty PC servers are used as the message generator, database servers, and Docker manager servers to build an experimental environment. An E-shop system was chosen as the application test case. The E-shop system is based on the benchmark program of TPC-W (Transaction Processing Performance Council for Web), and Pylot software (V 1.26, Apache Software Foundation, Forest Hill, MA, USA) as the message generator to generate high concurrent load. The E-shop system consists of several modules according to the micro-service framework, such as a shopping service, a logistics service, a transportation business service, a car service, and a business pay service. The configuration of the testing environment of the experiments is shown in Figure 6.

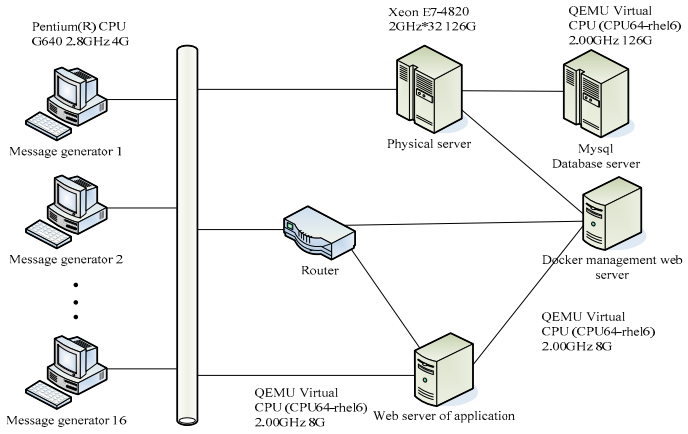

Figure 6. Configuration of the experimental environment.

As the elastic performance improvement is a multi-dimensional characterization, four sets of experiments are set up to verify the effectiveness of our work. Experiment one is to verify the

throughput evaluation of cloud containers. Experiment two is to verify the prediction accuracy and response delay of the services. Experiment three is to verify the system resource consumption. Experiment four is to verify the elastic scalability of containers.

(1) Throughput Evaluation of Cloud Containers

Experiment one evaluates the throughput of messages in four different ways. The reasons for their changes are tracked and analyzed by observing changes in the amount of messages sent in different ways. There are concurrent messages (the size of batch message is from 20 to 2000) to verify the effectiveness of message queue processing strategy. A single message is sent (every 200 bytes) as the experimental base, experimental data of the message number produced per second are recorded. According to Figure 7, we can see the comparison data of LightDocker (Red Hat Enterprise Linux 7 Atomic Host, Red Hat Inc., Raleigh, NC, USA), Docker (V 1.13, dotCloud, Inc., San Franciso, CA, USA), Docker with Workflow(An prototype system developed in [5]), and Elastic-Docker systems. Figure 8 shows the average experimental results of throughput in the environment.

After collecting running data for a period of time, it can be seen from the results that Elastic-Docker system greatly improves the concurrency processing capability compared with original Docker system. As the amount of message generation continues to increase, the message batch number of the Docker with Workflow and Docker grows slowly. As seen from the LightDocker's curve, there is a significant downward trend in the rate of message sending as the messages increase. The main reason is that the simplification of the container mirroring component reduces its processing ability.

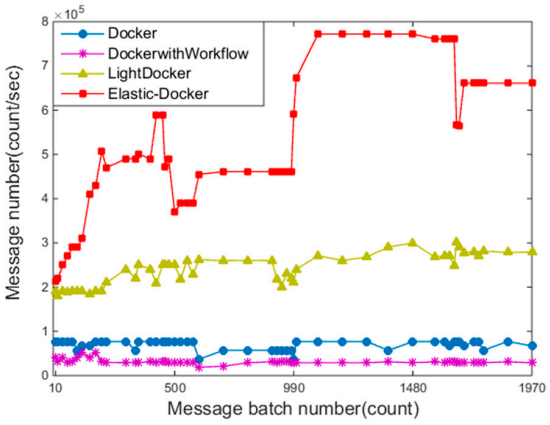

Figure 7. Comparison of the message throughput.

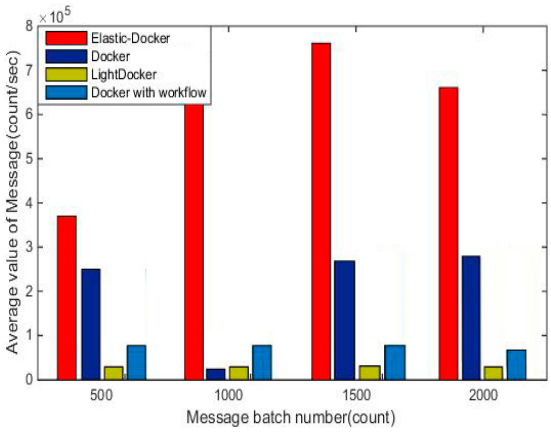

Figure 8. Comparison of the average throughput.

Elastic-Docker still maintains a low delay when the number of batch messages is 2000, and the throughput improves by 90%. Docker with Workflow curve in Figure 8 shows that the adoption of workflow mechanisms in Docker does significantly improve service efficiency, but when a large number of messages come into the system, the improvement of performance becomes inconspicuous and short. As mentioned above, Elastic-Docker effectively improved the message throughput.

(2) Prediction Accuracy and Response Delay of Services

In Experiment 2, the changes of the message processing delay and the accuracy of the predictive value are compared in four ways to verify the validity of the elastic expansion ability. In order to test the accuracy of the container demand prediction method, different numbers of messages are selected (the size of batch message is from 100 to 1500).

The Elastic-Docker system runs with different algorithm for prediction, including Grey prediction, neural network algorithm, and regression analysis algorithm. The predicted values from different methods are compared with an actual value. The experimental results are shown in Figure 9. By using four different methods to deal with elastic management, we can see the changes of the service response speed. It can be seen that the error value of Grey prediction is in the range of [0.06, 0.15], which can be used to predict the demanded quantity of container with better prediction accuracy and lower error value. The experiment shows that the presented forecasting method in the paper can effectively predict the value of the container according to less historical data, so as to continuously improve the quality and efficiency of services. Figure 10 shows the average response delay of message processing. According to the experimental data, there is no significant difference in the number of message delays when the number is in the [20, 1500]. However, as the message number continues to increase, the latency of Docker, LightDocker, Docker with Workflow are growing nearly linearly.

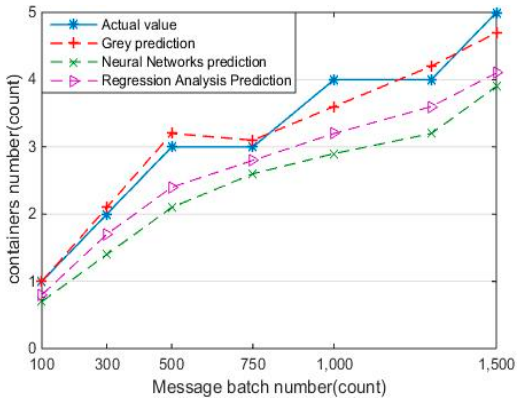

Figure 9. Actual value and predicted value.

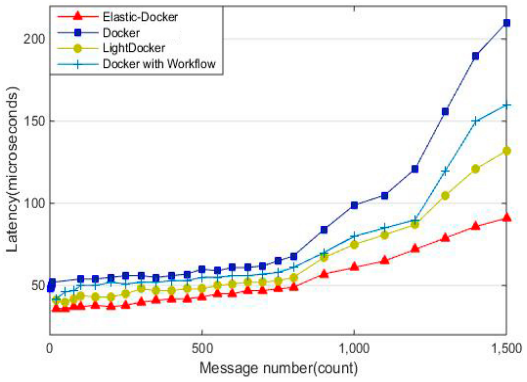

Figure 10. Comparison of service latency.

(3) System Resource Consumption

Experiment 3 compares the system resource consumption in four different ways. In order to verify message processing in the context of system resource overhead, four different situations are tested to compare CPU and memory consumption in containers under continuous monitoring.

Figures 11 and 12 show that CPU consumption and memory consumption of the four systems increase with the increase of the number of messages. With the increase of workload, the consumption of CPU and memory under the four kinds of system increase gradually. The Elastic-Docker is less expensive than the other three methods in terms of CPU and memory consumption when the number of message reaches 1500/s. Due to the accurate message scheduling for container clusters, Elastic-Docker is significantly lower than the other three systems in terms of resource consumption when faced with a message volume of 1500/s. This experiment shows the advantages of Elastic-Docker resource consumption, obviously.

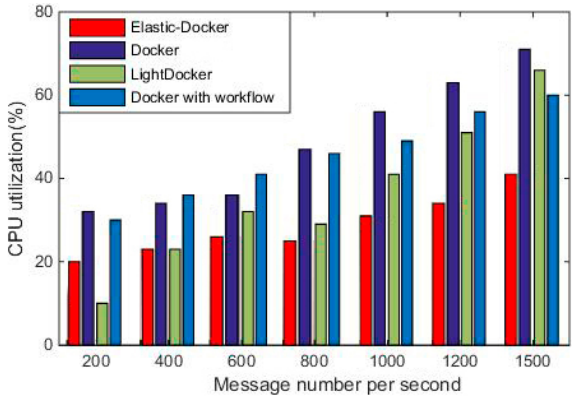

Figure 11. Comparison of CPU consumption.

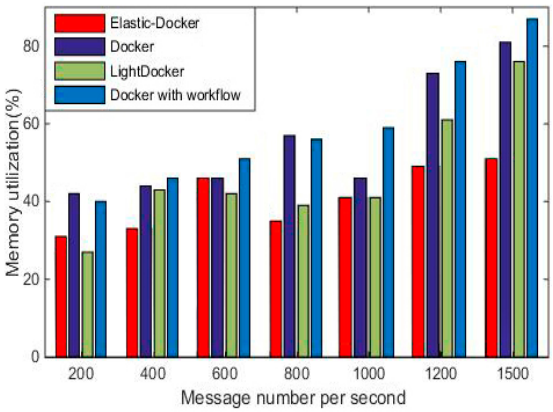

Figure 12. Comparison of memory consumption.

(4) Elastic Scalability of Cloud Containers

Experiment 4 compares the elastic scalability of Elastic-Docker with original Docker. The times of containers that can be stretched in a high concurrent environment is an important indicator to measure the flexibility under the same cluster size of hosts. In this paper, the scenarios of workload changes are frequently selected as the experimental environment. The average time of stretching and shrinking of the two systems is counted in several experiments, as shown in Figure 13.

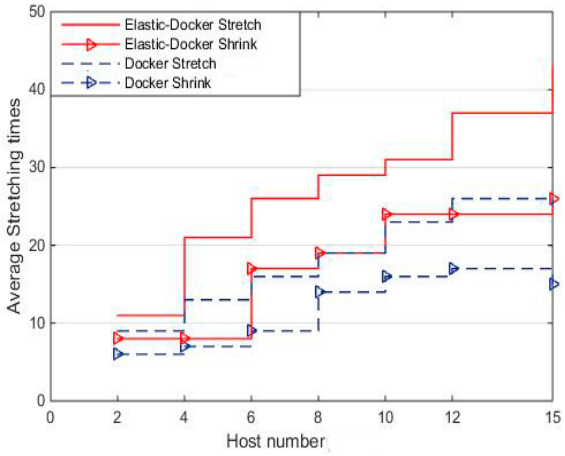

Figure 13. Comparison of scalability.

In the case of high concurrent applications, frequent changes in workload led to the pressure of the container, and also frequent changes, so it can effectively test the elasticity in this environment. With the increase of hosts in the container cluster, the elasticity is 60% higher than the original Docker system. The destruction will be carried out only when a container has no message to deal with anymore, thus, the amount of stretch is higher than the amount of shrink. The experimental results show that Elastic-Docker has higher flexibility than the original Docker.

7. Conclusions

This paper proposes an elastically-scalable policy for cloud containers, which meets the requirements of cloud environment and reduces the delay of service provision. Compared with the existed strategies, real-time prediction of containers number and message scheduling are taken into account in the presented policy, so as to protect the SLA (Service Level Agreement) of tenants and improve the provider's resource utilization. The method, based on predictive message scheduling to obtain the classifications and mappings of messages, makes message processing more targeted, which greatly improves the scalability of containers effectively. The experiments on the prototypal system show that the proposed policy can meet the demand of dynamic changes in workload and the elasticity of the system. We will address the larger message processing problem and further reduce the service response delay, and study optimization methods to improve the message processing performance in the future.

Acknowledgments: This work is supported by the Natural Science Foundation of China (No. 61363003, 61063012), the National Key Technology R and D Program of China (No. 2015BAH55F02).

Author Contributions: Yan Chengxin designs the architecture of Elastic-Docker and implements the system prototype. Analysis and interpretation of experimental data. Chen Ninjiang contributes to conception and the design of Elastic-Docker. Drafting the article and guiding it critically for important intellectual content. Zhang Shuo mainly finished the language collation and the proofreading. Collecting and arranging the correlative materials.

Conflicts of Interest: The authors declared that they have no conflicts of interest to this work.

References

1. Amaral, M.; Polo, J.; Carrera, D. Performance Evaluation of Microservices Architectures using Containers. In Proceedings of the 14th IEEE International Symposium on Network Computing and Applications, Cambridge, MA, USA, 28–30 September 2015; pp. 27–34.
2. Jaramillo, D.; Nguyen, D.V.; Smart, R. Leveraging microservices architecture by using Docker technology. In Proceedings of the 2016 IEEE South East Conference, Norfolk, VA, USA, 30 March–3 April 2016; pp. 1–5.

3. Bernstein, D. Containers and cloud: From Lxc to docker to kubernetes. *IEEE Cloud Comput.* **2014**, *1*, 81–84. [CrossRef]
4. Shojafar, M.; Canali, C.; Lancellotti, R.; Abawajy, J. Adaptive Computing-plus-Communication Optimization Framework for Multimedia Processing in Cloud Systems. *IEEE Trans. Cloud Comput.* **2016**, 1. [CrossRef]
5. Zheng, C.; Thain, D. Integrating Containers into Workflows: A Case Study Using Makeflow, Work Queue, and Docker. In Proceedings of the 8th International Workshop on Virtualization Technologies in Distributed Computing, Portland, OR, USA, 15 June 2015; pp. 31–38.
6. Hofmeijer, T.J.; Dulman, S.O.; Jansen, P.G. Dcos, a real-time light-weight data centric operating system. In Proceedings of the IASTED International Conference on Advances in Computer Science and Technology, Innsbruck, Austria, 22–24 November 2014; pp. 259–264.
7. Sallou, O.; Monjeaud, C. GO-Docker: A Batch Scheduling System with Docker Containers. In Proceedings of the IEEE Cluster 2015, Chicago, IL, USA, 8–11 September 2015; pp. 514–515.
8. Nikol, G.; Tr, M.; Harrer, S. Service-Oriented Multi-tenancy (SO-MT): Enabling Multi-tenancy for Existing Service Composition Engines with Docker. In Proceedings of the IEEE Symposium on Service-Oriented System Engineering, Oxford, UK, 29 March–2 April 2016; pp. 238–243.
9. Kan, C. DoCloud: An elastic cloud platform for Web applications based on Docker. In Proceedings of the 18th International Conference on Advanced Communication Technology, Pyeongchang, South Korea, 31 January–3 February 2016; pp. 478–483.
10. Kang, H.; Le, M.; Tao, S. Container and Microservice Driven Design for Cloud Infrastructure DevOps. In Proceedings of the 2016 IEEE International Conference on Cloud Engineering, Berlin, Germany, 4–8 April 2016; pp. 202–211.
11. Ueda, T.; Nakaike, T.; Ohara, M. Workload characterization for microservices. In Proceedings of the 2016 IEEE International Symposium on Workload Characterization, Providence, RI, USA, 25–27 September 2016; pp. 1–10.
12. Kousiouris, G.; Cucinotta, T.; Varvarigou, T. The effects of scheduling, workload type and consolidation scenarios on virtual machine performance and their prediction through optimized artificial neural net-works. *J. Syst. Softw.* **2011**, *84*, 1270–1291. [CrossRef]
13. Imam, M.T.; Miskhat, S.F.; Rahman, R.M. Neural network and regression based processor load prediction for efficient scaling of Grid and Cloud resources. In Proceedings of the 14th International Conference on Computer and Information Technology, Dhaka, Bangladcsh, 22–24 December 2011; pp. 333–338.
14. Li, W.; Kanso, A.; Gherbi, A. Leveraging Linux Containers to Achieve High Availability for Cloud Services. In Proceedings of the IEEE International Conference on Cloud Engineering, Tempe, AZ, USA, 9–13 March 2015; pp. 76–83.
15. Stubbs, J.; Moreira, W.; Dooley, R. Distributed systems of microservices using Docker and Serfnode. In Proceedings of the 7th International Workshop on Science Gateways, Budapest, Hungary, 3–5 June 2015; pp. 34–39.
16. Ismail, B.I.; Goortani, E.M.; Ab Karim, M.B.; Tat, W.M.; Setapa, S.; Luke, J.Y.; Hoe, O.H. Evaluation of Docker as Edge computing platform. In Proceedings of the IEEE Conference on Open Systems, Malacca, Malaysia, 24–26 August 2015; pp. 130–135.
17. Affetti, L.; Bresciani, G.; Guinea, S. aDock: A Cloud Infrastructure Experimentation Environment Based on Open Stack and Docker. In Proceedings of the 8th International Conference on Cloud Computing, New York, NY, USA, 27 June–2 July 2015; pp. 203–210.