Efficient Bare Metal Auto-scaling for NFV in Edge Computing

Xudong Pang, Jing Wang, Jingyu Wang, Qi Qi(✉), Jie Xu, and Zhenguang Yu

State Key Laboratory of Networking and Switching Technology,
Beijing University of Posts and Telecommunications, Beijing, China
qiqi8266@bupt.edu.cn

Abstract. Elasticity is an essential attribute of cloud data center, which is critical for operating resources in face of peaks and valleys of business. At present, the automatic scaling technique of virtual machines is widely studied, but barely for physical machines. Despite lack of flexibility, we all know that physical server can perform faster and more efficiently than virtualized instances, especially in Network Function Virtualization (NFV) systems. Some virtual network functions (VNFs) actually require high performance computing, which is a hard task for virtual machines. Besides, good management of bare metal resources can be significant for the data center power cost and human maintenance cost. Accordingly, we think that auto-scaling of physical machine is worth studying. This paper proposes a bare metal automatic scaling scheme based on workload prediction, and finally make tests on an open source NFV platform. The new scheme obtains good result on computation intensive VNFs scenario, including complete the scale in minutes, guarantee for the continuity of VNF processing business, and can cope with the load fluctuation better.

Keywords: Bare metal · NFV · Auto-scaling · Scheduling · Edge computing

1 Introduction

MEC offers storage and computational resources at the edge, reducing latency for mobile end users and utilizing more efficiently the mobile backhaul and core networks.

Network Function Virtualization (NFV) will leverage modern technologies such as those developed for cloud computing, which provides an analysis of the MEC orchestration considering standalone services, service mobility, joint network and service optimization as well as a comprehensive study of current orchestrator deployment options. That is to say there is always an edge data center behind the NFV system [1], and this kind of cloud data center usually consists of several servers, the workload time series for the whole cluster appear to be large fluctuation. Only auto-scaling of virtual machines cannot meet the needs of all. Some specific use-cases, for example, in the high-performance computing clusters, computing tasks that require access to hardware devices which cannot be virtualized [2], and for the database hosting (some databases run poorly in a hypervisor), single tenant dedicated hardware for performance, security, dependability and other regulatory requirements. When the cloud infrastructure rapidly

S. Liu et al. (Eds.): EDGE 2018, LNCS 10973, pp. 67–79, 2018.
https://doi.org/10.1007/978-3-319-94340-4_5

deployment, it's necessary for auto-scaling of bare metal resources. But managing that large number of physical servers is a hassle, if all by human operations, to ensure the quality of service (QoS) is an obvious challenge. Cloud computing itself has the characteristics of independent, safe, reliable, elastic, durable and others, so the automatic scaling of bare-metal resources is necessary.

Amazon Web Services (AWS) developed the first virtual machine auto-scaling tech in the industry, which is a solution for the holiday's 5 times shopping rush to normal flow. Moreover, the automatic scaling technique was applied to more areas, such as some of the virtualization resources of cloud computing. Accordingly, we consider how to make the bare-metal resources become more elastic for the ever-expanding data center. Traditional redundancy allocation can cause wasting of resources, so the main thing to do is to narrow the gap between allocating resources and actual needs.

However, there is no unified operating platform for bare metals currently, and we can only manage through some access and configuration tools such as IPMI, Cobbler or Ansible. Moreover, making all these processes automated is not easy, which requires a complete and elaborate workflow to complete, including new machines discovery, nodes deployment, as well as balance workloads to new online machines.

Besides, in advance allocation of resources can greatly improve quality and reduce the response latency of applications. Therefore, it is critical to find resource bottlenecks in time. We adopt Zabbix [3, 4] as a monitor, which is powerful and easy to expand. A recent survey classifies auto-scaling techniques into several major categories [5], including static policies, threshold-based polices, reinforcement learning, control theory and time-series analysis. For our design, we combine threshold rules and time-series analysis to make a suitable auto-scaling schedule algorithm. Based on the above techniques, we present an automatic horizontal scaling [6] algorithm for bare metal resources and made a series of experiments on this.

The second section introduces our related work. The third section describes details of our design framework, including core algorithm and related platform configurations. The fourth section describes detailed experiment steps and data comparisons. Finally, we conclude at the fifth section.

2 Related Work

NIST explained "rapid elasticity" as "capabilities can be elastically provisioned and released, in some cases automatically, to scale rapidly outward and inward commensurate with demand" [7]. The purpose of elasticity as defined in [8] is to provide performance, cost, increase infrastructure capacity, energy. All of these can be related to user load.

Due to the elasticity demand of physical server resources on NFV system, this paper probes into the automatic scaling of the bare metal resources. Here we use OpenStack [9] powered cloud as NFV infrastructure. It is well-known that OpenStack is an open source cloud platform, which has an excellent distributed architecture. A normal OpenStack cluster can be divided into the control nodes, network nodes, computing nodes, storage nodes. And there is also an Ironic project, which can help us manage bare metal resources. Better is, there is also a Tacker [10] project, which provide a NFV MANO conception implementations.

In addition, among many monitors we choose Zabbix, not only because it is one of the main monitoring tools of today's operation and maintenance staffs, but also its easy extendibility. Based on monitoring health status of cluster hardware, Zabbix can also make the upper application quality check, which feedback VNF response latency in time. Besides, Zabbix can also automatically discover and register new nodes. We use custom monitoring strategies, which are more suitable for both bare metal and NFV scenarios. In order to ensure the stability and smoothness of the upper services, Zabbix collects monitoring data periodically, which can be used to identify obvious features and predict loads of next moment.

We also present an auto scaling algorithm, which fully consider characteristics of bare metal resources, and also make full use of historical workload data for prediction. All these enable clusters to save as much resources as possible. Due to the process of bare metal provision is very time consuming, we need to do a good job in multi-stage pretreatment in advance, for example, packaging VNF image in advance, making some buffered servers, all these can make the new machine launch for work faster.

3 Architecture Design

3.1 Workflow of the Framework

Figure 1 shows our workflow, which based on a NFV Management and Orchestration (MANO) framework. The right part of this figure is a user request process, user requests which hit the same ASG will be dispatched to different servers by a load balancer.

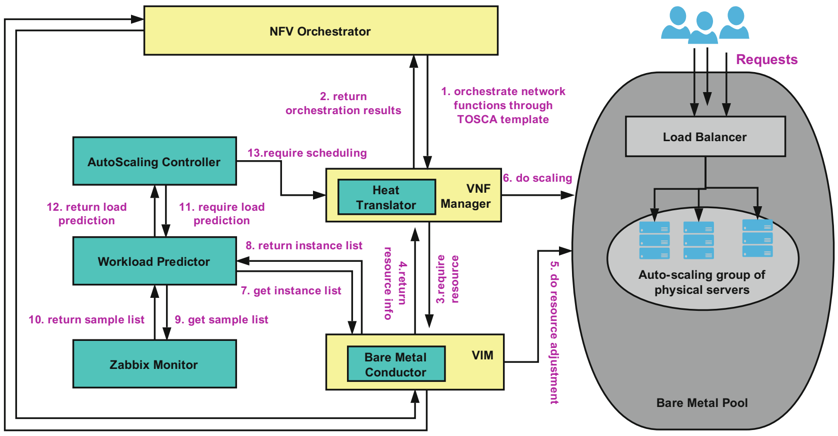

Fig. 1. Bare metal auto-scaling workflow on NFV MANO platform

Workload predictor uses historical data collected by Zabbix monitor as input sample, to make a workload prediction with our model, and then outputs a predict workload for next moment. This prediction will be send to auto-scaling controller, which will make a schedule decision with our design policy and send to VNF Manager (VNFM). VNFM is responsible for the lifecycle management of VNF. A VNF Manager can manage one or more VNFs, this is the ability to automate VNF including deployment, expansion, scaling, offline. VNFM can receive TOSCA templates from Orchestrator and transform them to VIM-understood profiles, and then handle bare metal instances though those profiles.

Topology and Orchestration Specification for Cloud Applications (TOSCA) is a service modeling language which can be used to model and deploy service function chains (SFCs). By defining TOSCA templates, which can be instantiated using concrete types, here we use this template to define VNF profiles. Orchestrator is responsible for NFV resources and is the basis of the upper software resource arrangement and management, this arrangement ability can be flexible adjustment of the VNF instances according to the requirements of the business. It's the core of the automatic ability.

Virtualized Infrastructure Manager (VIM) is a cloud platform, which is responsible for physical and virtual resource management. It can manage infrastructure resources, and schedule servers to different regions. Besides, our bare metal conductor will handle scaling actions in time.

3.2 Bare Metal Provision

According to our design, there are mainly three stages on bare metal provision, basically are as follows:

Machine Discovery: SNMP or INTEL RSA can be used to discover new electrically charged bare metals and report their information to conductor. Also, zero-conf provides a really good alternative for general bare metal discovery scenario, which does not rely on dedicated switches.

OS Installation: Nova uses Ironic driver to install operating system for the bare metals, and the images are provided by OpenStack glance service. This process is also the most time-consuming phase, average requiring approximately 20 min.

Software Configuration: Here we use tools like Ansible, Chef and puppet. Due to the new launched node will report its metadata to the controller voluntarily, and then automatically register to the corresponding cluster. We configure Zabbix agent and some service synchronize operations.

3.3 Monitor Configuration

Zabbix makes a good monitor for cloud data center, it is competent for large-scale cluster, can support IPMI and network equipment monitoring, besides, it is easy to extend, which is very suitable for NFV platform. According to [11], CPU accounts for 60% of the host's power consumption, and can basically represent machine energy consumption. We also use CPU load as default workload. Workloads of each processor can be detected by Zabbix agent periodically. Here we configure our Zabbix monitoring items: average CPU load of each physical server, average response latency of each VNF, and Power status of each physical server.

In Eq. (1), we define the workload of an ASG is the sum of workload of every processor in it.

$$Load(t) = \sum_{i=1}^{m} Load(t, i) * CoreNum(t, i) \tag{1}$$

Another monitoring item is used to reflect the real-time service quality of VNFs. In Eq. (2), we define the average VNF responding latency of an ASG is the average latency of VNFs on every active server in the same group.

$$Latency(t) = \sum_{i=1}^{m} Latency(t, i) \Big/ m \tag{2}$$

Auto-Registration: when a new machine is in the provision stage, it will be configured with a Zabbix agent, this agent can then be found by Zabbix server in local area, and then the machine will be registered to the corresponding auto-scaling group (ASG).

3.4 Workload Prediction Model

There are many forecasting models currently, including time series model, differential equation model, gray prediction model and so on. Differential equation model is based on the assumption of local rule independence, and the solution of it is difficult to obtain. And gray prediction model can be highlighted only when its prediction sequence has an exponential growth characteristic.

We choose autoregressive and moving average (ARMA) model [12], a time series prediction model, to predict the workloads of ASGs, and then convert to the schedule of bare-metal resources by schedule program.

In Eq. (3), we show an AR (3) as special shape, which is using a linear combination of the workloads of the previous three moments, to describe the workload predictions for the next moment. This equation is a result of what we screened through experiments (Fig. 2).

$$Load_{pre}(t+1) = 0.8Load(t) + 0.15Load(t-1) + 0.05Load(t-2) \tag{3}$$

$$Load_{pre}(t+1) = 0.5Load(t) + \cdots + 0.5^{t}Load(1) \tag{4}$$

Exponential smoothing prediction model is a special weighted moving average method, [13] which predicts the future workload by assigning a larger weight to the workload near the forecast period and making the weight from the near-far-decreasing exponential law. In Eq. (4), the mathematical expression of is shown in Table 1.

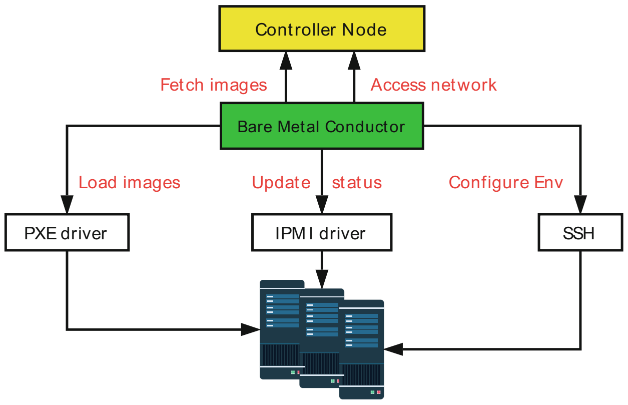

Fig. 2. Bare metal provision

Table 1. Related symbols and descriptions

Parameters	Meanings
m	Running server number of an ASG
n	Shutdown server number of an ASG
k	Total prediction times of an ASG
$Load(t,i)$	Average load of server i at time t
$Load(t)$	Predict average load of an ASG at time t
$Load_{avg}(t)$	Predict total load of an ASG at time t
$Load_{pre}(t)$	Predict total load of an ASG at time t
$CoreNum(t,i)$	Core number of running server i at time t
$CoreNum_{off}(t,j)$	Core number of an off-server j at time t
$CoreNum(t)$	Working processor number of an ASG at time t
$CoreNum_{min}(t)$	Min predict core number of an ASG at time t
$CoreNum_{max}(t)$	Max predict core number of an ASG at time t
$Latency(t,i)$	Average VNF response latency of server i
$Latency(t)$	Average VNF response latency of an ASG
$MinLoad$	Low average load threshold of an ASG
$MaxLoad$	High average load threshold of an ASG

Mean Absolute Error (MAE), Mean Square Error (MSE) and Mean Absolute Percent Error (MAPE) are three models usually used to evaluate prediction accuracy. The smaller obtained result is, the more accurate the prediction is. In this paper, we use MSE method to achieve better prediction accuracy, and Eq. (5) shows its mathematical expression.

$$MSE = \sqrt{\sum_{t=1}^{k} [Load_{pre}(t) - Load(t)]^2 \Big/ k} \tag{5}$$

Table 2 shows the workload prediction MSE of Eqs. (3) and (4). We have a one-week statistical analysis of the load data and forecast data from our testbed. Comparing their mean squared errors of the two models, we can see that both models got good experiment results, and especially, the AR model obtained better prediction accuracy and is more stable.

Table 2. Mean squared error of prediction models

	Sun	Mon	Tue	Wen	Thu	Fri	Sat
AR	0.081	0.079	0.120	0.083	0.181	0.168	0.125
MA	0.102	0.092	0.107	0.101	0.239	0.156	0.164

3.5 Auto-Scaling Control Logic

First, we consider scale in and scale out as the following ways, that's a little bit different from usually on virtual machines.

Scale-out: when VNFM triggers scale out for an ASG, according to the need to increase the number of servers, first to find are the configured but shutdown servers in the same group, power them up to fit the need. But if no enough off machines left, it will then request for filling the gaps in demand by triggering provision process of bare metal in the resource pool. **Scale-in:** when VNFM triggers scale in action, it just chooses the right nodes and turn them off.

Notice that here requires all VNFs should be either a stateless type or can be automatically migrated. Stateless VNFs use a shared data layer, so they don't need to manage their own data. This is much easier and can be easily switched to meet the changing requirements [14].

In fact, the bare metal pool has a garbage collection mechanism. When there is not enough resource in it, it will find some shutdown servers in ASGs by Least Recently Used (LRU) policy [15] and then reset PXE as the first boot device.

In the Fig. 3, there are 3 statuses for a bare metal, those are UP, DOWN and EMPTY, which represent the basic status of active, shutdown, and released. Besides, there are generally 4 actions for a bare metal, those are ON, OFF, DEPLOY and RELEASE. Due to the DEPLOY action is very time-consuming, scale out always try to trigger ON action at first, but if there is no resource in ASG, it will request from bare metal pool, and DEPLOY new instances. Based on the above rules, a practical workflow carried out in stages. That is about how to deal with specific overload and low load.

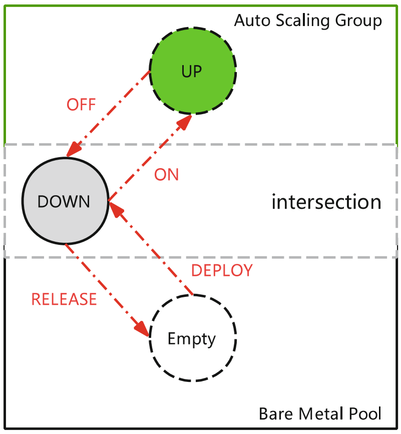

Fig. 3. State transition diagram of a bare metal

Stage One: real-time acquisition of prediction samples from Zabbix's monitoring data. Workload predictor will retrieve sample data every 30 min (according to physical machine acceptable interval, also avoid fluctuations). We use data of latest previous p moments as input, and output predict results of next moment [16]. Equation (3) describes a predict workload, it's the sum of all processor workloads in an ASG.

Stage Two: in this threshold judgment phase, according to the real-time load of each server, when the average load of an ASG occurs overload or low load situation, that ASG needs to be expanded or constrained according to its current state. We use the predict workload to adjust the number of working processors, according to Eq. (6) we can get the average value of predict load at time $t + 1$. Equations (7) and (8) shows the range of predict processor num. In Eq. (9) shows that conditions whether we need to adjust core number or not. We take the ideal situation, where MinLoad equals 0.75, MaxLoad equals 1.75.

$$Load_{avg}(t+1) = Load_{pre}(t+1) \big/ CoreNum(t) \tag{6}$$

$$CoreNum_{\mathrm{min}}(t+1) = Load_{pre}(t+1) \big/ MaxLoad \tag{7}$$

$$CoreNum_{\mathrm{max}}(t+1) = Load_{pre}(t+1) \big/ MinLoad \tag{8}$$

$$CoreNum_{\mathrm{min}}(t+1) \leq CoreNum(t) \leq CoreNum_{\mathrm{max}}(t+1) \tag{9}$$

Stage Three: Scaling decision phase, when the condition described by inequality (9) is not satisfied, there should be an adjustment of that ASG. if the right condition of the inequality is not satisfied, that is, the ASG will be in a low load state and current resources will exceed real needs. The target of the processor resource to be adjusted is the result shown in the formula (8), the surplus processor quantity is the result shown in formula (10). Similarly, if the condition of the left side of the inequality is not satisfied, that is, the ASG will be in an over load state and current resources will do not meet real needs.

The target of the processor resource to be adjusted is the result shown in formula (7), insufficient core number is the result shown in formula (11).

$$\Delta_{lowload} = CoreNum_{\max}(t+1) - CoreNum(t) \tag{10}$$

$$\Delta_{overload} = CoreNum_{\min}(t+1) - CoreNum(t) \tag{11}$$

$$SORT([CoreNum(t,1),\ldots,CoreNum(t,i),\ldots CoreNum(t,m)],ASC) \tag{12}$$

$$SORT([CoreNum_{off}(t,1),\ldots,CoreNum_{off}(t,j),\ldots,CoreNum_{off}(t,n)],DESC) \tag{13}$$

Stage Four: in scheduling phase, the method of sorting allocation is adopted. If it will be in low load state, running servers in that ASG will be sorted by core number in ascending order according to formula (12), and the servers are selected sequentially until selected core number satisfies the result shown in formula (10). However, if it will be in overload state, off servers in that ASG will be sorted by core number in descending order according to formula (13), and the servers are selected sequentially until selected core number meets the result shown in formula (11). When the result cannot be satisfied, requests of remaining amount should be sending to the bare metal pool. Similarly, do sorting for bare metals according to formula (13), and select bare metals sequentially until it meets the demand, if resources in bare metal pool still cannot meet needs, then alert for datacenter resource shortage and wait for free resources in queue.

4 Implementation and Evaluation

Experiment should be in real production environment, the biggest advantage of this is we can validate the stability. But in order to avoid the experiment cost and in a more controlled environment, we decided to setup a customized test bed, this will cost some effort on system configuration.

In this platform, we used ten physical servers to test and optimize our auto-scaling algorithm. Each physical machine has 12 Intel® Core CPUs, 24 GB RAM and 10G Ethernet, and powered with CentOS7 system. There are two active nodes at initialization phase, one controller and network node, one compute.

In this paper we use Deep Packet Inspection (DPI) as the experimental VNF, for it's a computation intensive network function and this is a typical bare metal scenario. Figure 4 shows our measuring method, we use a client program to send requests to a flow receiver and measure the response latency difference between closing and opening the DPI firewall [17]. We wrote about 2 thousand lines of code (RScript and Python) for this test bed, including prediction algorithms, auto-scaling control and automation combination logic.

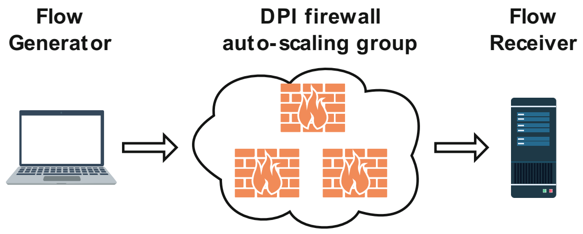

Fig. 4. DPI performance test

We compare the actual performance of multiple scheduling algorithms, assessing the VNF response rate and the number of physical servers used. Three schedule algorithms are chosen as the evaluation benchmarks. The four schedulers prediction-based and with transition state, no-prediction but with transition state, prediction-based but with no transition state, and no-prediction and with no transition state, represent four different scaling strategies are denoted by s1, s2, s3 and s4, respectively.

The performance of these schedulers is evaluated mainly from two perspectives, eager mode and idle mode, which represent two extremes that can highlight our auto-scaling policies. For the sake of statement, here we define the request density from 0 to 1, meaning density increasing.

First, we show a more comprehensive perspective (from idle mode to eager mode) in Fig. 5. Each point represents the average workload per processor. We can see that the CPU loads trend of our proposed auto-scaling scheme is more stable and is the case of the smallest fluctuations.

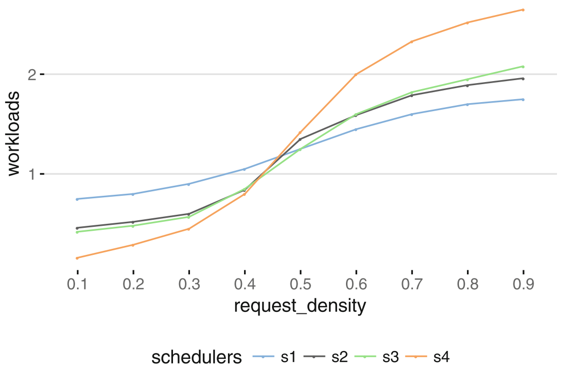

Fig. 5. Average workloads per processor

Figure 6 shows that when bare metal resources are insufficient, requests can be delayed due to the high CPU loads. We can see that s2 and s3 show roughly the same performance. And our strategy is about 18% less than their average delay. As for the no-prediction and no transition state scheduler, our strategy is about 29% less than the average delay. The conclusion is our prediction-based auto-scaling scheme can improve the quality of service VNF under eager mode.

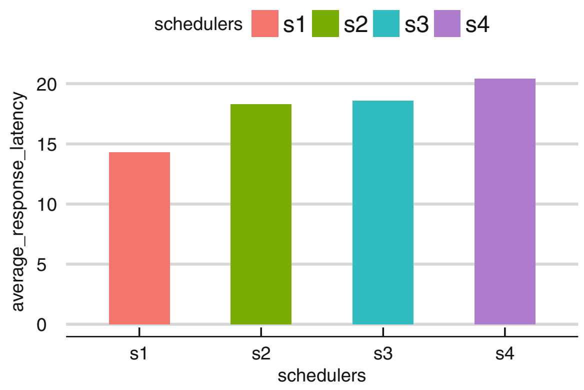

Fig. 6. Average VNF latency for eager mode (ms)

Figure 7 shows that when the bare metal resources are sufficient, power resources may be wasted, due to the adjustment is not timely. We can see that s2 and s3 generally use the same number of servers, about wasted 16% processor resources than our scheduler. Moreover, the no prediction and no transition state scheduler wasted about 21% of the power resources than ours. The conclusion is our prediction-based auto-scaling scheme can reduce power waste under idle mode.

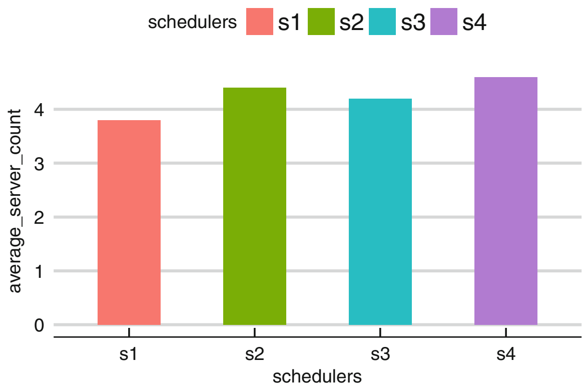

Fig. 7. Average server counts for idle mode

5 Conclusion

In this paper, we presented a bare metal auto scaling mechanism based on NFV system. This mechanism uses a method of forecasting and pre-allocating physical resources, to make VNFs scale in and scale out more smoothly. Compared with traditional way of relying more on human operations, it provides better QoS. Our work on bare metal auto-scaling may provide an idea for the NFV platform maintenance, with more intelligent, green, tight performance. Using this method, we believe that we opened a door for bare metal auto-scaling to obtain significant improvements in both cost and execution time. As the future work we intend to extend the prediction algorithm optimization, there is still great room for improvement. And we also want to study on the auto-scaling supporting both virtual machine and physical hosts, in a hybrid mode. This may be a more normal scenario in practice. In the future, the NFV resource scaling may can be optimized by the popular machine learning mechanisms [18–21].

Acknowledgment. This work was jointly supported by: (1) National Natural Science Foundation of China (No. 61771068, 61671079, 61471063, 61372120, 61421061); (2) Beijing Municipal Natural Science Foundation (No. 4182041, 4152039); (3) the National Basic Research Program of China (No. 2013CB329102).

References

1. ETSI: Network Functions Virtualisation (2012). https://portal.etsi.org/nfv/nfvwhitepaper.pdf
2. Gupta, A., Kale, L.V., Milojicic, D., Faraboschi, P., Balle, S.M.: HPC-aware VM placement in infrastructure clouds. In: Cloud Engineering (IC2E) (2013)
3. Naik, P., Shaw, D.K., Vutukuru, M.: NFVPerf: online performance monitoring and bottleneck detection for NFV. In: Network Function Virtualization and Software Defined Networks (NFV-SDN), 7–10 November 2016
4. Managing and monitoring performance in SDN/NFV (2015). https://www.virtualizationpractice.com/managing-monitoring-performance-sdn-nfv-32088/. Accessed 15 Dec 2015
5. Gajjar, P., Shah, B.: An efficient scalable framework for auto scaling services in cloud computing environment. IJIRT **2**, 113–120 (2016)
6. Chaloemwat, W., Kitisin, S.: Horizontal auto-scaling and process migration mechanism for cloud services with skewness algorithm. In: International Joint Conference on Computer Science and Software Engineering (2016)
7. Mell, P., Grance, T.: The NIST definition of cloud computing (2011)
8. Sakellariou, R., Zhao, H.: A hybrid heuristic for dag scheduling on heterogeneous systems. In: 18th IEEE International Parallel and Distributed Processing Symposium (2004)
9. OpenStack (2010). https://www.openstack.org/
10. Tacker (2015). https://wiki.openstack.org/wiki/Tacker
11. Fan, X., Weber, W.D., Barroso, L.A.: Power provisioning for a warehouse-sized computer. ACM SIGARCH Comput. Architect. News **35**(2), 13–23 (2007)
12. Wu, Y., Hwang, K., Yuan, Y., Zheng, W.: Adaptive workload prediction of grid performance in confidence windows. IEEE Trans. Parallel Distrib. Syst. **21**(7), 925–938 (2010)
13. Liang, W., Huang, T., Chen, J., Liu, Y.: Workload prediction-based algorithm for consolidation of virtual machines. J. Electron. Inf. Technol. **35**(6), 1271–1276 (2013)
14. Kablan, M., Caldwell, B., Han, R., Jamjoom, H., Keller, E.: Stateless network functions. In: SIGCOMM (2015)
15. Liu, C., Liu, Z., Zhang, D.: A cloud computing physical machine recovery method and its device. CN102831016 A (2012)
16. Chuprikov, P., Nikolenko, S., Kogan, K.: On demand elastic capacity planning for service auto-scaling. In: IEEE International Conference on Computer Communications (INFOCOM) (2016)
17. Garcia, J.: A clustering-based analysis of DPI-labeled videoflow characteristics in cellular networks. In: Integrated Network and Service Management (IM), 8–12 May 2017

18. Xu, P., Yin, Q., Huang, Y., Song, Y.-Z., Ma, Z., Wang, L., Xiang, T., Kleijn, W.B., Guo, J.: Cross-modal subspace learning for fine-grained sketch-based image retrieval. Neurocomputing **278**, 75–86 (2018)
19. Ma, Z., Xue, J.-H., Leijon, A., Tan, Z.-H., Yang, Z., Guo, J.: Decorrelation of neutral vector variables: theory and applications. IEEE Trans. Neural Netw. Learn. Syst. **29**(1), 129–143 (2018)
20. Liu, W., Cao, J., Yang, L., Xu, L., Qiu, X., Li, J.: AppBooster: boosting the performance of interactive mobile applications with computation offloading and parameter tuning. IEEE Trans. Parallel Distrib. Syst. **28**(6), 1593–1606 (2017)
21. Ma, Z., Rana, P.K., Taghia, J., Flierl, M., Leijon, A.: Bayesian estimation of dirichlet mixture model with variational inference. Pattern Recogn. **47**(9), 3143–3157 (2014)