Tetris: I/O Aware container placement strategy to avoid SLO violation in Kubernetes

Kyle Martin
kdmarti2@ncsu.edu

Vasudev Bongale
vabongal@ncsu.edu

Sohail Shaikh
sashaikh@ncsu.edu

ABSTRACT

Kubernetes (K8s) when provisioning resources for containerized applications does not measure the current I/O workload of the target provisioning node. This could lead to multiple I/O intensive containers creating an I/O bottleneck, leading to both CPU and memory being under utilized. This could potentially create a horrendous feedback loop as the scheduler attempts to handle additional resource allocation requests by unwittingly provisioning additional I/O intensive containers on an I/O bottle-necked node. Also, resource schedulers do not determine the potential ramifications of provisioning resources from an anomalous request that could put a Kubernetes node into an unstable state. We hope to improve K8s ability to provide better resource management by introducing I/O heuristics and reliability. We aim to achieve this by proactively avoiding unstable resource allocations and load-balancing through predictive forecasting algorithms into the master scheduler through the implementation of a scheduling extension.

KEYWORDS

Kubernetes, I/O workload, Container resource allocation and Load-balancing

1 INTRODUCTION

Containers are replacing virtual machines in most large-scale and dynamic application deployments. Orchestrating containers to handle a diverse ecosystem of applications provides few notable benefits: they are extremely lightweight, can be spawned instantaneously, and provide flexibility through dynamic configurations depending on application requirements.

An efficient resource allocation for managing multiple containers in parallel is extremely crucial. Over-provisioning of resources leads to under-utilization whereas under-provisioning may lead to starvation and Service Level Objective (SLO) violations. Monitoring current resource utilization is essential to optimize the deployment of additional containers and predict future consequences.

The provisioning of containers is further complicated by heterogeneous resource demands of containers co-existing on the same node. Most of the workloads can be classified as being CPU, memory or I/O intensive. A container having high I/O demand can impact the I/O of other containers. Hence, it becomes necessary to schedule containers based on their workload types.

Docker [1] is one of the most adopted container-based virtualization solutions along with Kubernetes [2]. Kubernetes (K8s) is a multi-host container orchestration and management system that operates in a Master-Slave model. Each slave in K8s cluster has an agent called *kubelet*. A K8s slave also contains an open-source container monitoring tool, *cAdvisor*, which gathers information on resource utilization of all containers in that node. The smallest deployment unit of K8s is called a *pod*, which comprises a single container or collection of related containers. *Kubernetes scheduler* is a master node component responsible for scheduling pods in K8s. System metrics are collected by *metrics server* and are exposed as Kubernetes API Resource to the scheduler which aids in provisioning decisions. However, K8s considers only CPU and memory utilization in its provisioning strategy. One of the K8s controllers, *Horizontal Pod AutoScaling* (HPA) provides horizontal elasticity by reactively scaling the number of pods based on predefined thresholds.

Existing CPU and memory based provisioning strategies may not be efficient in scheduling container applications that are I/O intensive. We propose to incorporate I/O based heuristics into scheduling decisions of *Kubernetes Scheduler* to provide a container-to-node allocation solution which is not limited only to CPU and memory intensive applications. I/O metrics can be characterized in terms of disk I/O or network I/O. We wish to evaluate the impact of these metrics and make an informed decision on incorporating one of these.

Currently, K8s does not take into account the post-provisioning impact of executing a particular container allocation request. There are no well-defined mechanisms to safe-guard the system from possible ill-effects that may occur due to an anomalous provisioning. An anomalous provisioning can be described as a deployment which leads to degradation in performance of containerized applications. We aim to implement a protection mechanism that tries to provide some guarantee about system stability by adding intelligence to container placement. We propose a proactive strategy by predicting system SLO compliance state using historical data of container resource metrics.

Containers allocated to a particular K8s node do not migrate without user intervention and will continue to run until termination. Redistribution of containers among nodes after container terminates could improve stability of an entire cluster. Also, identifying idle containers for termination could improve data collection as idle containers do not provide continuous historical data but sparse intermittent data. This data does not provide an accurate forecasting model. We wish to avoid such models by detecting and pruning idle containers. This process of redistribution and provisioning can be formulated as a bin-packing problem, where a node represents a bin. Given the hardness of arriving at an optimal solution in bin-packing, we propose to employ the algorithm on smaller sub-problems [15] or use a Greedy algorithm [8] to attain overall optimal system resource utilization.

To summarize, following are the proposed contributions of our project:

- Incorporation of I/O metrics in Kubernetes.
- Implementing learning algorithms to predict system SLO compliance when provisioning a new container.
- Implementing load-balancing strategies to minimize idle containers to provide system stability.

2 RELATED WORK

Reliability is one of the key performance indicators of distributed systems. Applications in the cloud have the flexibility to achieve performance and scalablity with a consumption based cost model. To minimize the cost and meet SLA requirements, resources provisioned for an application should be elastic. Several works have been carried out towards elastic solutions for auto-scaling resource provisioning.

Predicting short term future demands in a cloud architecture is essential to adapt to dynamic requests and thus reduce costs [4]. Compared to reactive approaches, proactive mechanisms are best at adapting to changing demands to meet elasticity goals. Considering the nature of various application workloads, it is inefficient to rely on a single predictor to forecast demands. Proposed algorithms for workload prediction can be classified into four types, viz. naive, regression, temporal and non-temporal [11]. An optimal predictor can be chosen depending on workload [4][11]. It is desirable to choose a light-weight algorithm without compromising accuracy and such an algorithm based on probabilistic modeling has been proposed by Zhenhuan et. al [18].

Predictive and statistical approaches have been developed to influence resource scaling by leveraging patterns in system metrics. Wajahat et al. [16] present neural-network based solution called MLScale for predicting number of instances that must be added to a cluster to meet SLA. Emiliano et al. [10] used historical data to develop models for estimating future resource usage of applications. They focus on representing resources consumed by a computational task rather than global system metrics. Nicolo M. et al. [5] develop a self-organizing probabilistic model, distributed over all instantiated cloud resources with the ability to make autonomous decisions for allocating or deallocating resources. Although their contribution deals with distributing resources in a *federated* cloud environment, the principles can be applied to distribute pods in a K8s cluster.

The default monitoring framework of Kubernetes relies on *cAdvisor* to gather system metrics. The appropriate choice of metrics for scaling and load-balancing depends on the application behavior. The impact of absolute metrics that are derived from `/proc` and the relative metrics gathered from docker tools or *cAdvisor* on container scaling has been studied [6]. The built-in scaler mechanism in Kubernetes supports CPU and memory statistics. Chang et al. [7] develop a generic platform with a comprehensive monitoring mechanism to facilitate dynamic resource provisioning using custom metrics.

Research in dynamic load balancing schemes include, optimizing CPU utilization through process profiling and preemptive migration [9], memory resource allocation through aggregating unused memory pages from a cluster of workstations [3], and implementing load balancing algorithms to minimize both idle CPU cycles and page faults to maximize resource utilization in a heterogeneous cluster [17]. These approaches are unable to balance I/O intensive applications due to improper load balancing of I/O workloads [14]. This problem has been addressed by preemptive job migration to continuously maintain high performance for homogeneous clusters [13]. Also for heterogeneous clusters, the development of two competing I/O heuristics was used in parallel to influence load balancing decisions to increase resource utilization [14]. Finally, I/O aware load balancing models for both homogeneous and heterogeneous distributed systems provided promising results when attempting to balance I/O intensive applications [12].

REFERENCES

[1] [n. d.]. Docker [Online]. Retrieved February 4, 2019 from https://docker.com
[2] [n. d.]. Kubernetes[Online]. https://kubernetes.io
[3] Anurag Acharya and Sanjeev Setia. 1999. Availability and utility of idle memory in workstation clusters. In *ACM SIGMETRICS Performance Evaluation Review*, Vol. 27. ACM, 35–46.
[4] Fabio Jorge Almeida Morais, Francisco Vilar Brasileiro, Raquel Vigolvino Lopes, Ricardo Araujo Santos, W. Satterfield, and L. Rosa. 2013. Autoflex: Service Agnostic Auto-scaling Framework for IaaS Deployment Models. In *2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*. IEEE, Delft, 42–49. https://doi.org/10.1109/CCGrid.2013.74
[5] Nicolo M. Calcavecchia, Bogdan A. Caprarescu, Elisabetta Di Nitto, Daniel J. Dubois, and Dana Petcu. 2012. DEPAS: a decentralized probabilistic algorithm for auto-scaling. *Computing* 94, 8-10 (Sept. 2012), 701–730. https://doi.org/10.1007/s00607-012-0198-8
[6] Emiliano Casalicchio and Vanessa Perciballi. 2017. Auto-Scaling of Containers: The Impact of Relative and Absolute Metrics. In *2017 IEEE 2nd International Workshops on Foundations and Applications of Self* Systems (FAS*W)*. IEEE, Tucson, AZ, USA, 207–214. https://doi.org/10.1109/FAS-W.2017.149
[7] Chia-Chen Chang, Shun-Ren Yang, En-Hau Yeh, Phone Lin, and Jeu-Yih Jeng. 2017. A Kubernetes-Based Monitoring Platform for Dynamic Cloud Resource Provisioning. In *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*. IEEE, Singapore, 1–6. https://doi.org/10.1109/GLOCOM.2017.8254046
[8] Zhenhuan Gong and Xiaohui Gu. 2010. Pac: Pattern-driven application consolidation for efficient cloud computing. In *18th IEEE/ACM International Symposium on Modelling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS 2010)*. IEEE, 24–33.
[9] Mor Harchol-Balter and Allen B Downey. 1997. Exploiting process lifetime distributions for dynamic load balancing. *ACM Transactions on Computer Systems (TOCS)* 15, 3 (1997), 253–285.
[10] Gideon Juve, Benjamin Tovar, Rafael Ferreira da Silva, Dariusz Krol, Douglas Thain, Ewa Deelman, William Allcock, and Miron Livny. 2015. Practical Resource Monitoring for Robust High Throughput Computing. In *2015 IEEE International Conference on Cluster Computing*. IEEE, Chicago, IL, USA, 650–657. https://doi.org/10.1109/CLUSTER.2015.115
[11] In Kee Kim, Wei Wang, Yanjun Qi, and Marty Humphrey. 2016. Empirical Evaluation of Workload Forecasting Techniques for Predictive Cloud Resource Scaling. In *2016 IEEE 9th International Conference on Cloud Computing (CLOUD)*. IEEE, San Francisco, CA, USA, 1–10. https://doi.org/10.1109/CLOUD.2016.0011
[12] Xiao Qin, Hong Jiang, Adam Manzanares, Xiaojun Ruan, and Shu Yin. 2009. Dynamic load balancing for I/O-intensive applications on clusters. *ACM Transactions on Storage (TOS)* 5, 3 (2009), 9.
[13] Xiao Qin, Hong Jiang, Yifeng Zhu, and David R Swanson. 2003. Boosting performance of I/O-intensive workload by preemptive job migrations in a cluster system. In *Computer Architecture and High Performance Computing, 2003. Proceedings. 15th Symposium on*. IEEE, 235–243.
[14] Xiao Qin, Hong Jiang, Yifeng Zhu, and David R Swanson. 2003. Dynamic load balancing for I/O-intensive tasks on heterogeneous clusters. In *International Conference on High-Performance Computing*. Springer, 300–309.
[15] Maria A Rodriguez and Rajkumar Buyya. 2015. A responsive knapsack-based algorithm for resource provisioning and scheduling of scientific workflows in clouds. In *Parallel Processing (ICPP), 2015 44th International Conference on*. IEEE, 839–848.

[16] Muhammad Wajahat, Anshul Gandhi, Alexei Karve, and Andrzej Kochut. 2016. Using machine learning for black-box autoscaling. In *2016 Seventh International Green and Sustainable Computing Conference (IGSC)*. IEEE, Hangzhou, China, 1–8. https://doi.org/10.1109/IGCC.2016.7892598

[17] Li Xiao, Xiadong Zhang, and Yanxia Qu. 2000. Effective load sharing on heterogeneous networks of workstations. In *Parallel and Distributed Processing Symposium, 2000. IPDPS 2000. Proceedings. 14th International*. IEEE, 431–438.

[18] Zhenhuan Gong, Xiaohui Gu, and John Wilkes. 2010. PRESS: PRedictive Elastic ReSource Scaling for cloud systems. In *2010 International Conference on Network and Service Management*. IEEE, Niagara Falls, ON, Canada, 9–16. https://doi.org/10.1109/CNSM.2010.5691343