Container Rebalancing: Towards Proactive Linux Containers Placement Optimization in a Data Center

Pongsakorn U-chupala*, Yasuhiro Watashiba*, Kohei Ichikawa*, Susumu Date† and Hajimu Iida*
*Nara Institute of Science and Technology, Nara, Japan
Email: {pongsakorn.uchupala.pm7,watashiba,ichikawa}@is.naist.jp, iida@itc.naist.jp
†Osaka University, Osaka, Japan
Email: date@ais.cmc.osaka-u.ac.jp

***Abstract*—Similar to Virtualization, Linux Containers (LXC) provides high-performance, lightweight computing resource allocation and isolation. Each LXC container has a resource overhead smaller than that of a virtual machine, leading to significantly lower container migration time and making frequent container placement modification a viable optimization technique. Traditional container scheduling mechanisms do not leverage this property of LXC. Generally, a scheduler tries to find the most optimal placement for a new container; the allocated host then executes the scheduled container until the end of the container's life cycle. This strategy works fine for short-lived containers. With a long-lived container such as a server process becoming more and more common, and the container placement calculated at the beginning of the execution may not remain optimal during the container's lifetime, since the other containers are moving in and out of the cluster. This research proposes container rebalancing, a novel scheduling mechanism with a rebalancing process working alongside a scheduling process. The container rebalancing method increases LXC cluster utilization while maintaining minimal interference with the scheduling process. This is done by continuously modifying container placement, by using the rebalancing process, in order to load-balance utilization of each host in the LXC cluster. LXC cluster simulation driven by Google's cluster data is used to verify the feasibility of container rebalancing. Simulation results show an observable increase in container scheduled rate and cluster utilization with no drawback, suggesting that container rebalancing is a promising method.**

I. Introduction

Virtualization technology [1], [2] allows resources on a single computer to be sliced into multiple isolated units. This technology is considered mature and is commonly used in data centers to enable more granular resource allocation resulting in a more flexible resource management, as well as a higher resource utilization. With Virtualization, resource isolation is achieved by virtualizing an entire computer with an allocated amount of resources. Each virtualized unit is called a "virtual machine" (VM). This method, however, takes inherent performance overhead [3].

Linux Containers (LXC) [4] is another technology for resource isolation. Instead of virtualizing an entire machine, thus requiring a separate "guest" operating system, LXC provides each "container" with its own separate Linux environment while still sharing the same underlying Linux kernel. Compared to VM, an LXC container takes significantly lower performance overhead [5], [6]. LXC containers are also instantiate

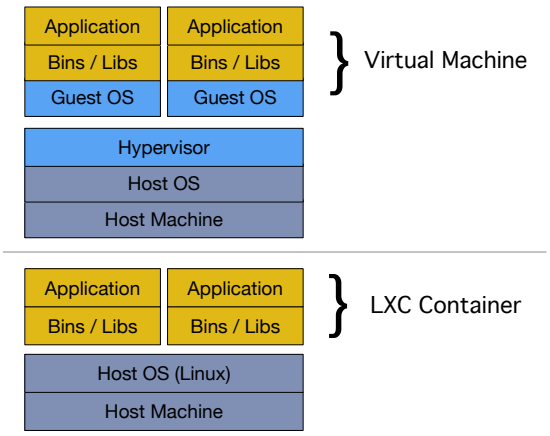


Figure 1: Abstraction comparison between Virtualization and Linux Containers

significantly faster and require less storage space compared to VM (as elaborate in Section II). These advantages make many LXC-based tools such as Docker [7] and CoreOS [8] popular alternatives to full virtualization-based solutions [9]. Figure 1 illustrates the difference between Virtualization and LXC.

Rapid container migration is a powerful technique made possible with LXC. An LXC container (referred to here as "container" from this point onward) may take *seconds* to boot up whereas a similar VM may take *minutes*. A container also takes less disk space than a VM, since a container does not have to contain the whole operating system. These two traits combined make container migration much faster than migrating the VM and offer a feasible resource management technique.

None of the existing container orchestration solutions take advantage of rapid container migration. With the rising popularity of LXC, many tools and technologies are being developed [10]–[13]. However, they concentrate on resource management and do not try to take advantage of the unique capabilities of LXC such as rapid container migration.

This paper explores the possibility of leveraging rapid container migration as a resource management technique in conjunction with existing optimization techniques so as to increase data center efficiency. In the process, we proposes

0730-3157/17 $31.00 © 2017 IEEE
DOI 10.1109/COMPSAC.2017.94

"container rebalancing", a novel mechanism with a rebalancing process constantly optimizes LXC container placement by using rapid container migration.

The rest of this paper is structured as follows. Section II provides a brief introduction to LXC technology and common LXC cluster scheduling mechanisms. Section III proposes a container rebalancing mechanism. Section IV describes the performance evaluation of container rebalancing using an LXC cluster simulation and the results. Finally, Section V provides the conclusion.

II. Background

This paper proposes container rebalancing, a novel method to increase LXC cluster efficiency by increasing the optimal overcommit ratio using rapid container migration. The approach is explained in detail in Section III. This section gives a brief introduction to Linux Containers technology and the rapid container migration technique. This section also discusses existing LXC resource management techniques including overcommitting.

A. Linux Containers

Linux Containers (LXC) is a technology that combines several Linux kernel features to create a contained process called "container". Each container has its isolated view of the operating system environment with only an allocated amount of resources, virtually achieving an isolated Linux environment without the use of Virtualization technology. LXC's containers incur significantly lower performance overhead than Virtualization since LXC's containers work on an operating-system-level [5], [6]. LXC may be viewed, in a sense, as a lightweight Virtualization, though there are several differences.

LXC technology offers several advantages over Virtualization. Since creating a new container is essentially the same as spawning a new process, a container could be instantiated almost immediately. A container also typically requires less disk space, since it does not contain the whole operating system. Docker reduces the disk space requirement further by using AuFS to enable a layered file system, allowing images to be stacked on top of each other. This results in each image containing only the data which differs from base images [14].

B. Rapid Container Migration

Migrating an LXC container could be considered as a trivial operation. Total time for migrating an isolated computing unit (t_{mi}), whether it is a container or a VM, could be broken down into 3 parts: disk copying time (t_{disk}), memory copying time (t_{mem}), and instantiation time (t_{inst}). The following equation describes the model of total migration time:

$$t_{mi} = t_{disk} + t_{mem} + t_{inst}. \quad (1)$$

Container migration time is greatly reduced compared to migrating a VM since speedy instantiation (small t_{inst}) and a relatively smaller disk space requirement (leading to small t_{disk}) are traits of LXC. This reduction of migration time is an important feature unique to LXC.

High VM migration time is a problem for migration-based scheduling strategy [15]. With significant reduction in migration time of LXC, rapid container migration becomes a viable optimization strategy.

C. LXC Scheduling and Overcommitting

Existing scheduling solutions for LXC clusters are typically designed as a general-purpose scheduling platform [10], [11]. While some solutions are designed specifically for managing LXC clusters by providing convenient workflow for containerized application deployment [12], [13], they are not taking advantage of LXC's unique capability. Even with a fairly efficient scheduling algorithm, actual resources utilization could still be at about 50–60%, while available resources (such as CPU cores and memory) are mostly allocated [16].

One common technique for increasing scheduling efficiency is overcommitting resources [17], [18]. Overcommitting is done by allowing the scheduler to allocate more resources than the actual capacity of the system on the assumption that allocated resources are typically higher than actual utilization. This method increases scheduling efficiency because it is difficult to predict accurately the required amount of resources leading to the common occurrence of an over-allocated resources request [11], [16]. Overcommitting is commonly done statically by setting a static overcommit ratio for each type of resource.

The overcommit ratio should not be set too high or too low. Setting overcommit ratio too high increases the chance of container failure (as there are not enough actual resources for the operations), increases the risk of resource congestion, and causes instability [19]. Without overcommitting or by setting the overcommit ratio too low (for a given system), resources are underutilized, which means inefficient scheduling and a waste of resources.

III. Container Rebalancing

Container rebalancing takes advantage of rapid container migration to increase the efficiency of LXC cluster scheduling. This section explains the design goals and mechanism of container rebalancing.

A. Design Goals

Three goals are considered in designing a container rebalancing method: proactive-optimization, compatibility, and scalability.

1) Proactive-Optimization: Container rebalancing anticipates future workloads and *proactively* optimizes container placement accordingly. Common scheduling methods usually optimize container placement *reactively* in response to changes in available resources. Proactive optimization enables the system to prepare quickly for future workloads and it works especially well in a busy cluster. This approach requires rapid migration which is a costly operation with Virtualization technology. With LXC, rapid container migration is a relatively cheap operation, thus making proactive optimization viable.

2) Compatibility: Container rebalancing should work alongside the existing scheduling process. Task scheduling is a rigorously researched area, and existing algorithms are fairly efficient. Instead of designing a better scheduling algorithm to replace an existing one, container rebalancing should be another process that works in conjunction with the scheduling process while minimizing interference to the scheduler.

3) Scalability: Container rebalancing should be able to handle a large number of containers efficiently. Since a container usually requires fewer resources than a VM with a similar configuration, an LXC cluster is expected to be able to deal with a higher number of containers. Google cluster data is a 1-month trace from one of Google's LXC clusters [20], [21]. This single month of data already contains 24,281,242 distinct tasks (containers). Given the expected size of the LXC cluster, the container rebalancing technique should be able to scale up gracefully.

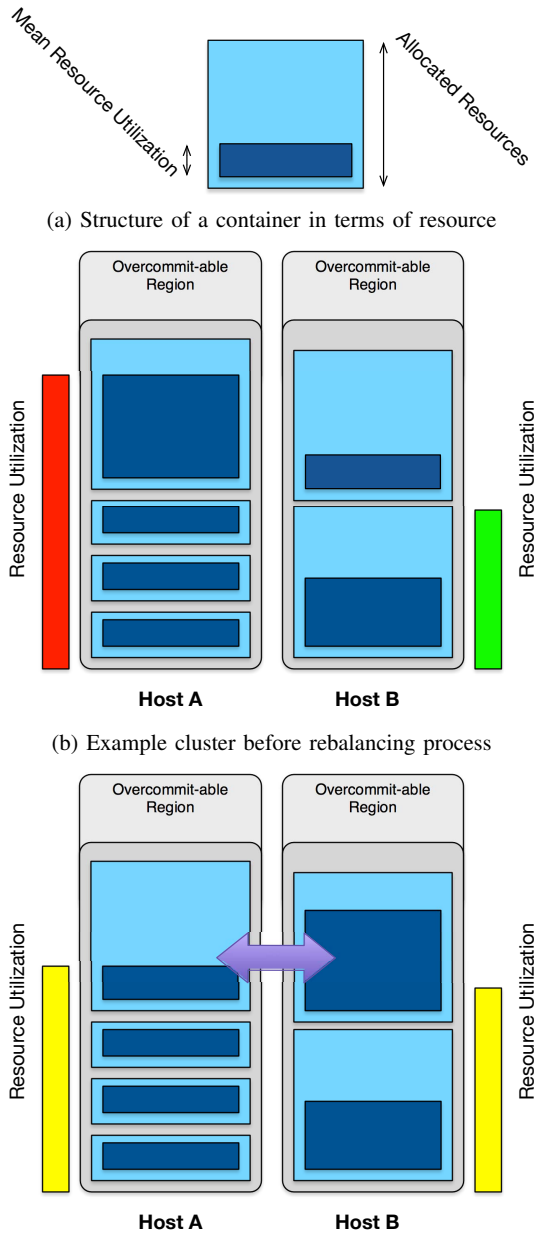
B. Implementation

The container rebalancing method consists of the rebalancing process that works alongside the scheduling process to load-balance resources between hosts of the cluster in real-time. This dynamic load-balancing process increases the probability of overcommit success as it proactively prepares rooms for overcommitting resulting in a higher optimal overcommit ratio. Figure 2 provides an example case illustrating the benefit of the container rebalancing method. Before the rebalancing process (Figure 2b), only host B has enough resources to accommodate overcommitted containers. However, both hosts have an equal chance of being overcommitted as both of them are equally allocated, making host A prone to overcommit failure. After the rebalancing process (Figure 2c), both hosts can accommodate overcommitted containers, thus increasing the chance of overcommit success.

To minimize the effect on the scheduling process, only actual resource utilization is load-balanced while resources allocation is maintained. Actual resources utilization of a container is often significantly less than the allocated amount of the container, as it is difficult to predict accurately and subject to human errors [16]. This trait of container request makes it possible to load-balance actual resources utilization while still maintaining the same allocation amount on each host dictated by the scheduler.

For scalability, the container rebalancing mechanism only load-balances long-lived containers. Container requests can be classified into two groups: long-lived (service) containers and short-lived (batch) containers. Although most containers are short-lived containers, most resources of the system are consumed by long-lived containers [11]. By considering only long-lived containers, the container rebalancing process maintains significant impact on the system while drastically reducing the number of containers the process must keep track of.

The container rebalancing process is divided into four repeating steps:



(a) Structure of a container in terms of resource

(b) Example cluster before rebalancing process

(c) Example cluster after rebalancing process

Figure 2: Example case illustrating the benefit of the container rebalancing method

1) Container Classification: Containers are classified into long-lived containers or short-lived containers as they are inserted into the system. The classification is done using container runtime duration in each container request.

2) Building Comparable Container Space: Long-lived containers are grouped together according to the amounts of their allocated resources and according to their assigned hosts, creating comparable container space.

Table I: An example job in the workload

Job ID	Container Index	Start Time (μs)	Duration (μs)	CPU Request (cores, normalized)	Mean CPU Usage (cores, normalized)
6252082279	0	6736765567	98000000	$6.76000e^{-3}$	$6.17230e^{-3}$
6252082279	1	6736765567	98000000	$6.76000e^{-3}$	$4.22395e^{-3}$

3) Searching Comparable Container Space: A pair of hosts with a significant resource utilization difference is selected. Comparable container space is then searched for the pair of containers from this host pair with the highest container actual utilization difference. This pair of containers is called a "swappable container pair".

4) Container Swapping: Each container in the swappable container pair is migrated to the host of its counterpart.

By repeating this process, overall utilization of the cluster is load-balanced while still retaining the original allocation assigned by the scheduler on each host. Better load-balanced cluster leads to a higher overcommit success rate, a higher optimal overcommit ratio, and a higher cluster utilization.

IV. Evaluation

An LXC cluster simulation is used to evaluate the performance and validate the feasibility of the container rebalancing mechanism. The simulation compares performance in terms of container scheduled rate and cluster utilization of a general scheduling mechanism (using only a scheduling process) to the container rebalancing mechanism (using both the scheduling process and the rebalancing process). The simulation is driven by a real-world workload from Google's cluster data. This cluster trace data is collected from Google's LXC cluster covering the period of one month and is publicly available [20], [21].

A. Workload Data Preprocessing

The workload for the simulation is extracted from Google's cluster data and organized into "jobs" and "containers". A "job" contains one "container", or multiple identical "containers" that have to be considered together while scheduling. Table I shows the organization of the workload and an example job. Google's cluster data already organizes the information into "jobs" and "tasks". A "task" is an equivalent to a "container" in the workload organization. Google's cluster data contains 672,074 jobs and 24,281,242 tasks from a 1-month period. Containers that are started before the trace period or still running after the trace period are excluded, since it is impossible to know the exact duration of these containers. 869,283 containers (3.58% of the total containers) are excluded in this process.

CPU cores are the only resources taken into account in the simulation. This is to simplify the simulation process and is required, due to time constrains, to speed up the simulation time. More investigation will be done to quantify the effect of this simplification on the accuracy of the result. The value provided by the trace data is normalized with the number of cores in the machine with the most cores in Google's cluster for obfuscation [20], [21].

B. Simulation

The simulation is an event-driven simulation with four processes running simultaneously: producer, scheduler, rebalancer, and monitor. The general scheduling mechanism is simulated using producer, scheduler, and monitor processes. The rebalancing mechanism is simulated using all four processes. Each machine in the simulated cluster has the same number of CPU cores which is equal to the number of CPU cores in the machine with the most cores in Google's cluster data. 2,000 machines in the LXC cluster are simulated in this research so that the total number of cores in the cluster is roughly equal to total number cores in Google's cluster. Although Google's cluster data contains the trace of a full month, to speed up the process, only the first week of the records was used in the simulation. SimPy 3.0.8 [22] was chosen for the implementation due to the authors' familiarity with Python.

1) Producer: The producer process inserts a "job" into the "job_queue" when the simulation time reaches the starting time of each "job". Algorithm 1 describes the producer process. Using the recorded duration in the trace data, this process also categorizes each "container" in a "job" as a long-lived container or a short-lived container. Containers with a duration shorter than 2,000 seconds are categorized as a short-lived containers, while the rest are categorized as a long-lived. 2,000 seconds was chosen as the threshold given that the 80th percentile runtime of a short-lived container is 12–20 minutes and that of a long-lived container is 29 days, according to the work by Schwarzkopf et al. [11].

2) Scheduler: The scheduler process implements a common scheduling strategy with overcommitting. Algorithm 2 describes the scheduler process. The scheduler handles container requests in a "job" batch. The scheduler uses a random-first-fit algorithm to find an open slot for each "job" for speed and simplicity. If all "containers" in a "job" could not be scheduled simultaneously, the "job" is skipped and queued for retrying during the next scheduling iteration. For this simulation, the *rebalancing interval* is set to 3 seconds and *max retries* is set to 300 times allowing a job to stay in the queue for about 15 minutes. If the "job" could not be scheduled within this time, it is skipped. Scheduled "jobs" and "containers" may fail if they are allocated to an overcommitted host which lacks sufficient actual capacity to facilitate them.

3) Rebalancer: The rabalancer process searches through scheduled containers for swappable container-pairs as described in Subsection III-B, and migrates each container to its counterpart's host. Algorithm 3 describes the rebalancer process. The search is performed in the host's utilization-difference order and the container's mean-cpu-usage-difference order respectively. A "container" with a mean CPU usage

Algorithm 1 Producer Process

loop
  Get job from preprocessed trace data (ordered by $start\_time$)
  if $job : start\_time$ exceeds current simulation time **then**
    Step current simulation time to $start\_time$
  end if
  Categorize and mark $containers$ in the job as long-lived container or short-lived container
  Insert job to job_queue
end loop

Algorithm 2 Scheduler Process

loop Every $scheduling\_interval$
  Get job from job_queue
  if $job : retries > max\_retries$ **then**
    Mark job and corresponding $containers$ as $skipped$
  else
    Try scheduling all $containers$ in the job into the cluster using random first-fit algorithm
    if job could not fit into the cluster **then**
      $job : retries = job : retries + 1$
      Put job back to job_queue
    else
      if Scheduled job cause over-utilization in the cluster **then**
        Mark job and corresponding $containers$ as $failed$
      else
        Mark job and corresponding $containers$ as $scheduled$
        for all $container$ **in** $job : containers$ **do**
          if $container$ is long-lived container **then**
            Add $(container, host)$ to $scheduled\_long\_lived\_containers\_list$
          end if
        end for
      end if
    end if
  end if
end loop

Algorithm 3 Rebalancer Process

loop Every $rebalancing\_interval$
  $max\_host =$ host with maximum CPU utilization
  $min\_host =$ host with minimum CPU utilization
  if $CPU(max\_host) - CPU(min\_host) \geq cpu\_utilization\_difference\_threshold$ **then**
    $container\_pairs = LongLivedContainers(min\_host) \times LongLivedContainers(max\_hosts)$
    Create empty $candidate\_pair$
    for all $pair$ **in** $container\_pairs$ **do**
      if $pair$ can be swap without exceeding available resources on both host **then**
        if ($candidate\_pair$ is empty) $\vee$
($CPUUtilizationDifference(pair) > CPUUtilizationDifference(candidate\_pair)$) **then**
          $candidate\_pair = pair$
        end if
      end if
    end for
    if $candidate\_pair$ is not empty **then**
      Swap $container$ of the $candidate\_pair$
    end if
  end if
end loop

difference lower than 0.3 is considered comparable. For this simulation, the *rebalancing interval* is set to 3 seconds, and the *cpu utilization difference threshold* is set to 0.1 to keep the rebalancing process from being too aggressive.

4) Monitor: The monitor process keeps track of resource utilization of hosts and containers in the cluster and also generates reports. After every *monitoring interval* of 60 seconds, this process logs utilization of each host in the simulated cluster and computes evaluation metrics, including *container scheduled rate* and its variances, as described in Subsection IV-C.

C. Results

Three metrics are used as the indicators of the performance of the simulated cluster: *container scheduled rate (CSR)*, *long-lived container scheduled rate (LCSR)*, and *short-lived container scheduled rate (SCSR)*. These metrics are defined as follows:

$$CSR = \frac{Scheduled\ Containers}{Total\ Containers}, \tag{2}$$

$$LCSR = \frac{Scheduled\ Long\text{-}lived\ Containers}{Total\ Long\text{-}lived\ Container}, \tag{3}$$

$$SCSR = \frac{Scheduled\ Short\text{-}lived\ Containers}{Total\ Short\text{-}lived\ Container}. \tag{4}$$

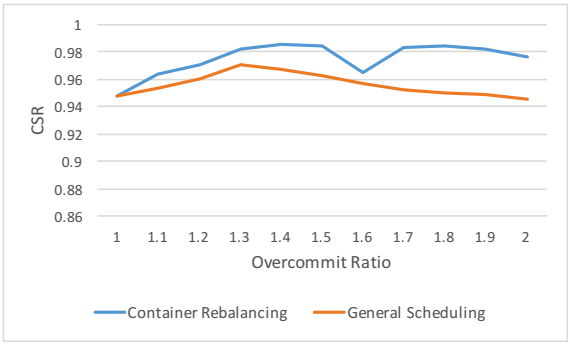
Figure 3 shows CSR, LCSR, and SCSR from the simulations with various overcommit ratios. The resulting CSR suggests that the optimal overcommit ratio of the simulated cluster with the general scheduling mechanism is about 1.3. The optimal overcommit ratio increased to about 1.4 with the container rebalancing mechanism. From the results, LCSRs also conform with CSRs, and there is no remarkable improvement with SCSRs. These two facts suggest that, among the factors contributing to the increase in utilization, rebalancing only the long-lived containers is significant.

There are notable CSR, SCSR, and LCSR drops at an overcommit ratio of 1.6. They are considered to be outliers and disappear when the random seed of the simulation is changed.

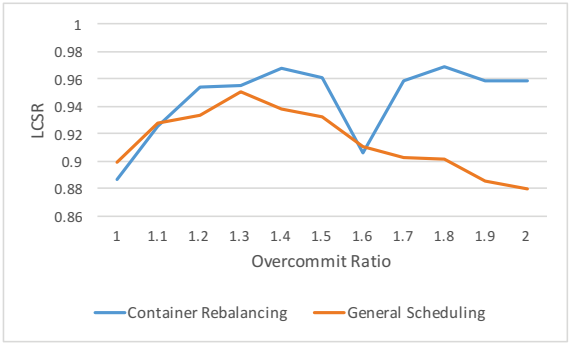
Figure 4 shows the average utilizations of the simulated clusters using the general scheduling mechanism and the container rebalancing mechanism. Figure 5 shows changes in utilizations of the simulated clusters over the course of the simulations. Only the overcommit ratios of 1.3 and 1.4 are shown in the figures since they are optimal overcommit ratios for general scheduling and container rebalancing, respectively. The results suggest that, at any given simulation time, the container rebalancing mechanism typically produces a higher cluster utilization compared with the general scheduling. Container rebalancing also outperforms general scheduling even at their respective optimal overcommit ratios (as indicated by comparing (a) the container rebalancing at overcommit ratio 1.4 to (b) the general scheduling at overcommit ratio 1.3).

D. Discussion

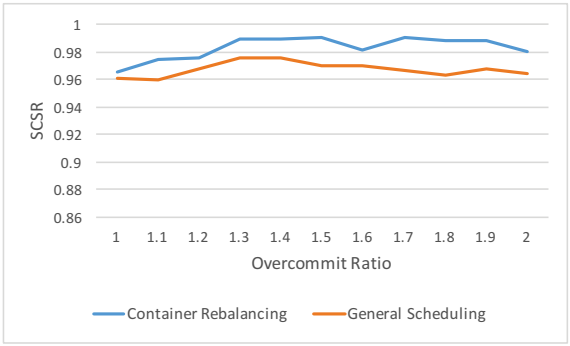
From the results (Figure 3), container rebalancing increases the optimal overcommit ratio of an LXC cluster by a small



(a) Container Scheduled Rate



(b) Long-lived Container Scheduled Rate



(c) Short-lived Container Scheduled Rate

Figure 3: CSR, LCSR and SCSR from the simulations

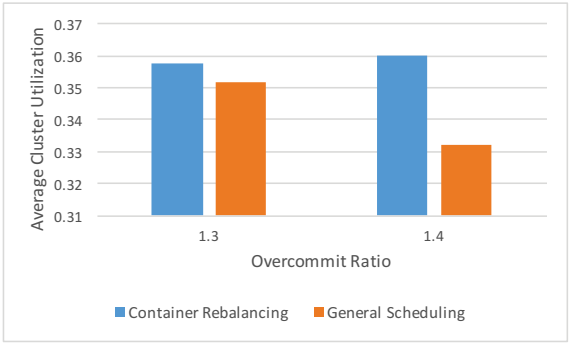


Figure 4: Average cluster utilizations from the simulations

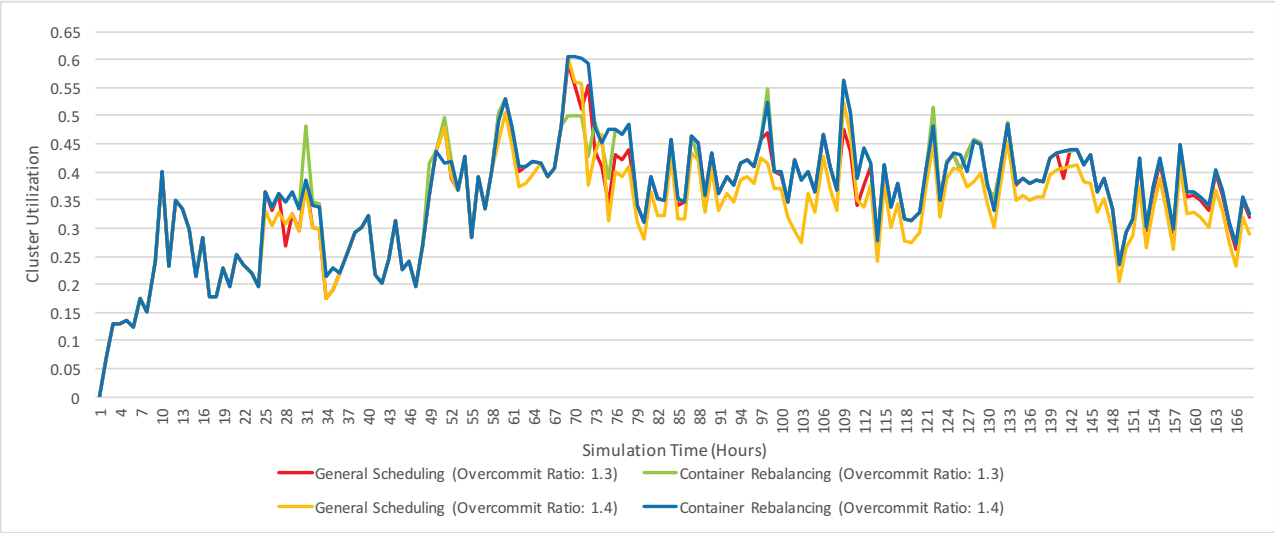


Figure 5: Cluster utilizations from the simulations

margin. This increase still creates observable utilization improvement in Figure 4 and Figure 5. The improvement is desirable as there is no drawback to using cluster rebalancing.

Container rebalancing also improves cluster performance (measured by container scheduled rate and cluster utilization in this research) regardless of the overcommit ratio. We believe that this is because a load-balanced cluster enables more containers to be successfully overcommitted, allowing more works to be done with the same amount of resources. Specifically, in this simulation, at overcommit ratio 1.4, 1.8% more containers are executed (an increase from 4,958,643 containers to 5,050,295 containers).

The overcommit ratio should be determined specifically for each cluster. Although the results suggest that the container rebalancing mechanism increases the overcommit ratio, multiple factors are influencing the optimal overcommit ratio of a particular cluster. For example, the number of hosts, the capacity of each host, and the scheduling algorithm can drastically change the value. Cluster simulation could be used to calculate the optimal overcommit ratio by using a cluster configuration similar to the targeted cluster and the real captured workload.

Rapid container migration adds little or no overhead to the execution. At overcommit ratio 1.4, 67,718 unique containers are migrated. This number is 1.32% of all containers in the simulation (5,127,542 containers) and 5.96% of all long-lived containers in the simulation (1,136,517 containers). Figure 6 shows the distribution of unique containers by their migration count at overcommit ratio 1.4. Around half of the affected containers are migrated only once. Almost all of the affected containers are migrated less than 20 times. Migrating a container is also considered a trivial operation (as is discussed in Section II). The other overcommit ratios display similar results; only the result at overcommit ratio 1.4 is shown, since it is the optimal overcommit ratio of this simulation.

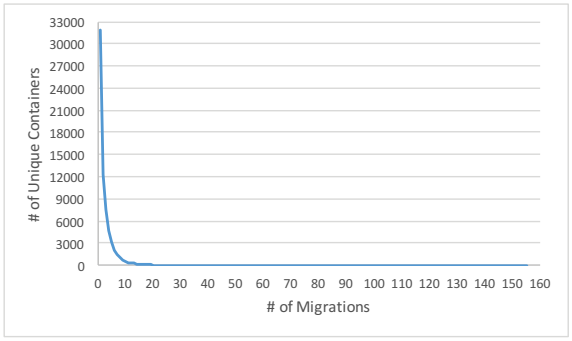


Figure 6: Distribution of unique containers by their migration count throughout the container rebalancing simulation at overcommit ratio 1.4

Due to time constraints, various compromises are taken to keep each simulation runtime to less than 24 hours. Although Google's cluster data contains one month of trace data, the simulation only simulates a single week, using the data of the first week in the trace. The results from Figure 5 are approaching stable values, which suggests that simulating only the first week is enough to get meaningful results. The results may be considered preliminary, because the simulation only takes into account a single resource, the CPU cores. More work has to be done to validate if cluster rebalancing is also feasible for multi-objective optimization. As a start, fuzzy-sets and weight-averaging fuzzy operators could be used to approach multi-objective optimization. This is similar to the approach in the work of Xu et al. [23].

V. Conclusion

This paper proposes container rebalancing, a novel scheduling mechanism with a rebalancing process working in conjunction with an existing scheduling process of LXC clusters.

Container rebalancing takes advantage of LXC's rapid container migration to increase the optimal overcommit ratio of an LXC cluster. This improvement, in turn, increases overall resources utilization of resources of the cluster.

The simulation is used to evaluate the performance and validate the feasibility of container rebalancing. Although many simplifications and compromises have been made due to time constraints, the results still suggest that container rebalancing is a promising method. More work is being done to investigate the effectiveness of this method, to improve the accuracy of the simulation, and to see the effect of this method with multi-objective optimization.

REFERENCES

[1] A. Kivity, U. Lublin, A. Liguori, Y. Kamay, and D. Laor, "kvm: the Linux virtual machine monitor," *Proceedings of the Linux Symposium*, vol. 1, pp. 225–230, 2007. [Online]. Available: https://www.kernel.org/doc/mirror/ols2007v1.pdf#page=225

[2] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," in *Proceedings of the nineteenth ACM symposium on Operating systems principles - SOSP '03*, vol. 37, no. 5. New York, New York, USA: ACM Press, oct 2003, p. 164. [Online]. Available: http://dl.acm.org/citation.cfm?id=945445.945462

[3] N. Regola and J.-C. Ducom, "Recommendations for Virtualization Technologies in High Performance Computing," *2010 IEEE Second International Conference on Cloud Computing Technology and Science*, pp. 409–416, 2010.

[4] "Linux Containers - LXC - Introduction." [Online]. Available: https://linuxcontainers.org/lxc/introduction/

[5] M. G. Xavier, M. V. Neves, F. D. Rossi, T. C. Ferreto, T. Lange, and C. a. F. De Rose, "Performance Evaluation of Container-based Virtualization for High Performance Computing Environments," *Proceedings of the 2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, pp. 233–240, 2013.

[6] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, "An Updated Performance Comparison of Virtual Machines and Linux Containers," *Technology*, vol. 25482, 2014. [Online]. Available: http://domino.research.ibm.com/library/CyberDig.nsf/papers/0929052195DD819C85257D2300681E7B/$File/rc25482.pdf

[7] D. Merkel, "Docker: lightweight Linux containers for consistent development and deployment," *Linux Journal*, vol. 2014, no. 239, p. 2, mar 2014. [Online]. Available: http://dl.acm.org/ft_gateway.cfm?id=2600241&type=html

[8] "CoreOS is Linux for Massive Server Deployments." [Online]. Available: https://coreos.com/

[9] D. S. Bernstein, "Containers and Cloud: From LXC to Docker to Kubernetes," *IEEE Cloud Computing*, vol. 1, no. 3, pp. 81–84, sep 2014. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7036275

[10] B. Hindman, A. Konwinski, M. Zaharia, A. Platform, F.-G. Resource, and M. Zaharia, "Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center." *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, pp. 295–308, 2011. [Online]. Available: http://static.usenix.org/events/nsdi11/tech/full_papers/Hindman_new.pdfhttps://www.usenix.org/conference/nsdi11/mesos-platform-fine-grained-resource-sharing-data-center

[11] M. Schwarzkopf and A. Konwinski, "Omega: flexible, scalable schedulers for large compute clusters," *EuroSys '13 Proceedings of the 8th ACM European Conference on Computer Systems*, pp. 351–364, 2013. [Online]. Available: http://dl.acm.org/citation.cfm?id=2465386

[12] "Kubernetes - What is Kubernetes?" [Online]. Available: http://kubernetes.io/docs/whatisk8s/

[13] "Swarm Overview." [Online]. Available: https://docs.docker.com/swarm/overview/

[14] "AUFS storage driver in practice." [Online]. Available: https://docs.docker.com/engine/userguide/storagedriver/aufs-driver/

[15] J. Hu, J. Gu, G. Sun, and T. Zhao, "A scheduling strategy on load balancing of virtual machine resources in cloud computing environment," *Proceedings - 3rd International Symposium on Parallel Architectures, Algorithms and Programming, PAAP 2010*, pp. 89–96, 2010.

[16] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. a. Kozuch, "Heterogeneity and dynamicity of clouds at scale," *Proceedings of the Third ACM Symposium on Cloud Computing - SoCC '12*, pp. 1–13, 2012. [Online]. Available: http://dl.acm.org/citation.cfm?id=2391229.2391236

[17] "33.4. Overcommitting Resources." [Online]. Available: https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/5/html/Virtualization/sect-Virtualization-Tips_and_tricks-Overcommitting_with_KVM.html

[18] "IBM Knowledge Center — Platform Resource Scheduler 2.2.0 ¿ Administering ¿ Resource over commit allocation ratios," jan 2013. [Online]. Available: http://www.ibm.com/support/knowledgecenter/SS8MU9_2.2.0/Admin/concepts/resourceovercommit.dita

[19] D. Breitgand, Z. Dubitzky, A. Epstein, A. Glikson, and I. Shapira, "Sla-aware resource over-commit in an iaas cloud," *Proceedings of the 8th International Conference on Network and Service Management*, pp. 73–81, oct 2012. [Online]. Available: http://dl.acm.org/citation.cfm?id=2499406.2499415

[20] J. Wilkes, "More Google cluster data," Google research blog, nov 2011.

[21] C. Reiss, J. Wilkes, and J. L. Hellerstein, "Google cluster-usage traces: format + schema," Google Inc., Mountain View, CA, USA, Technical Report, nov 2011.

[22] "Overview SimPy 3.0.8 documentation," 2015. [Online]. Available: http://simpy.readthedocs.io/en/3.0.8/

[23] J. Xu and J. A. B. Fortes, "Multi-objective virtual machine placement in virtualized data center environments," in *2010 IEEE/ACM International Conference on Green Computing and Communications, GreenCom 2010, 2010 IEEE/ACM International Conference on Cyber, Physical and Social Computing, CPSCom 2010*, 2010, pp. 179–188.