Aridac: Adaptive Resource Isolation of Non-volatile Devices Under Heterogeneous Containerized Environment

Xiang Yue
School of Computer Science
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213
Email: xiangyue@andrew.cmu.edu

Xuan Peng
Information Networking Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213
Email: xuanpeng@andrew.cmu.edu

Zeyu Wang
Information Networking Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213
Email: zeyuwang@cmu.edu

***Abstract*—Container technology makes cloud computing possible by offering resource isolation and scalability, however, resource sharing brings the problem where heavy containers consume most of the resources and break the fairness. To achieve fairness, we plan to propose an adaptive resource isolation policy, Aridac, for adjusting the disk resource quota of containers.**

1. Introduction

Cloud computing platforms, as a typical instance of infrastructure-as-a-service, nowadays have become the first choice for internet application developers to deploy their services. The main reason behind the popularity is - cloud computing offers reliability, convenience, isolation, and scalability, along with the pay-as-you-go pricing model. To achieve those features, the quality of service must be ensured with the achievement of resources isolation technologies.

Container, as the core part of isolation, creates an illusion for customers as owning the entire system. This lightweight virtualization notion has been widely leveraged by cloud service providers. Based on Linux Cgroup, which offers the ability to isolate resource (e.g. namespace, network I/O, disk I/O), container technologies and applications raised, like Linux Container (LXC) [1] and Docker [2]. Those form the basic units of cloud platforms and are still a great domain for cloud providers to dive deep into.

However, behind the great idea of containerization, there come new problems with resources isolation in practice. One of the glaring issues to resolve is resource allocation fairness, which means avoiding the resources allocated to containers interfering with each other. For instance, when there are several containers within a node and one of them is running heavy workloads with intense I/O operations, the resources of other nodes can be consumed, greatly delaying the delivery of tasks, like the completion of file reading, the transmission of RPC requests, etc. In addition, Xavier et al. [3] has also analyzed and confirmed the interference of heavy disk workload without limits on resource allocation can degrade the overall performance of the system. Therefore, to avoid sudden delay of container processes, a limitation of resource quota is a good way to control.

Although there are already some tools (cgroup, trickle) for adjusting resource quota, the problem is, those tools only offer fixed limitations of resource quota, whereas, in industry, cloud providers can never know how to adjust quota in advance or in real-time - the application in containers and containers themselves are dynamically created, killed and changed frequently. In addition, one of our group members met with the same problem with an intolerant latency of services when resources are consumed mostly by a disk-intensive container. Some works [4] manages to alleviate the issue. Since there are still few efforts working on this topic in both academy and industry, our group planned to look into a dynamic way of isolation adjustment.

We plan to propose the Aridac, an Adaptive Resource Isolation algorithm of Non-volatile Devices under heterogeneous containerized environment. The basic idea is to monitor, collect and analyze the resource usage habit of containers as time goes on, and allocate their resources to ensure fairness under certain rules. The quota of disk I/O can be allocated by, for instance, the priority of the application, or by containers' history maximum usage. And the objective is to achieve fairness. Besides the algorithm, we plan to design the testing workloads and design experiments for comparing fairness between different policies with specific metrics. Also, we will provide the corresponding benchmark data for further research. To narrow down the scope, we focus on disk isolation, which can be extended to the network I/O or other resources allocation scenario. The basic tools of system design and experiments include cgroup, bash scripts.

The rest of the proposal is organized as follows: Methodology section shows how we will define the effectiveness of Aridac; Goals includes our promising results of research; Final paper plan illustrates how we will write the final paper.

2. Methodology

2.1. Evaluation

Supposedly, to evaluate the effectiveness of Aridac, we will carry out benchmark experiments under different kinds of workload. Most experiment setttings will be invariants,

including the host machine (physical machine), container software, container version, testing programs, etc. The only variant is the isolation policy. One setting will use Aridac, while the other one simply does not use any.

2.2. System Measurement Metrics

The detailed discussion of the metrics selection is still ongoing and will be posted here soon. This is the very essential component of our project, and it is also a very difficult part, for which we need to be very careful and take many aspects into consideration. So we choose to post a placeholder here, instead of rushing with some metrics that are not good enough.

In general, we want the metrics to be correct, precise, and able to take the semantics of different I/O operations into consideration. For example, between the extremely heavy disk write operations introduced by image pulling during deployment in one container, and moderately heavy disk write operation from a critical web service in the other container, we definitely want Aridac to catch difference among those operations, whether through some OS observability methods or manually injected configurations. It's fair to say that this metrics will determine the effectiveness of Aridac.

3. Goals

The overall goal is to finish implementing Aridac, testing it under various kinds of workload scenarios, measuring how much improvement we achieve, and finally, analyzing the overhead brought by Aridac and discussing the trade-offs.

To evaluate Aridac, we could start from the following perspectives:

- The completeness of implementation. i.e., whether the isolator can be applied to the resource contention scenario that we mentioned before.
- To what extent does Aridac help. i.e., how much peak & average disk throughput improvement can we gain with Aridac.
- What's the price we need to pay? i.e., how many extra machine resource will be consumed by Aridac.

Here, we set 3 concrete goals in terms of the final progress, which represent 75%, 100%, and 125% degree of completion, respectively.

For the 75% completion goal, our aim is to cover the following aspects:

- Finish implementing Aridac, the adaptive disk resource isolator.
- Make sure that Aridac could run successfully without crashing or bugs.
- Aridac achieve a better disk I/O throughput than the non-optimized version.

For the 100% completion goal, we will strive to finely tune the isolation policy in order to achieve an optimal resource sharing among containers, which will be gauged by overall throughput of the system. We will use the following steps to check whether Aridac has achieved this goal:

- A set of tests carrying different workload will be designed to simulate typical resource sharing situations in real-world datacenter containerized environment.
- Aridac must excel in all test cases that we designed.
- Concrete benchmark data will be given, specifying by how much Aridac is leading in terms of system I/O throughput.

Beyond the 100% goal, if everything goes well, we will further develop a fuzzy disk I/O workload generator, so that all possible real-life cases will be covered, incluing assorted corner cases and rare cases. Our anticipation for this 125% goal is that Aridac could survive through this fuzzy test. We will also carry out benchmarking, and do some statistical work to show how well Aridac could be in a totally random environment.

4. Final Paper Plan

In our final paper, we will first introduce the resource isolation problem of the container environment and discuss the background and related work. Next, we will briefly outline the system design and implementation of Aridac. Last, we will describe our experimental setup, including what machines and environments were used, what workload we developed to test our implementation. The most crucial part of the paper will be concerned with describing and evaluating our methods for container resource isolation. We will provide performance analyses, and conclude with a discussion about potential optimization schemes, and if possible, a set of benchmarks for more detailed performance evaluation.

The early stage of our project will be analyzing existing issues caused by weak isolation of the container environment, researching existing relative works on different levels of virtualization technologies and container resource isolation, and finding out a doable logic for performing a dynamic resource isolation scheme in the container environment. Our project will then focus on developing a dynamic resource isolation policy for the container environment, to achieve the goal of balancing the resources among multiple containers running on the same physical machine. We will implement a program working on dynamic resource allocation and isolation for containers sharing the same underly resources, and may try out different policies for resource allocation of network bandwidth and disk I/O of the physical machine to learn their performance characteristics.

We will design and develop a set of workloads to simulate some most common applications running in the industry. Then we will let a few containers run that workload and monitor their resource and performance, to see if any rapid resource obtaining will happen and broke the resource allocation balancing. If we can reproduce the scenario of container resource exhausting caused by weak isolation, then we may follow up by testing whether it strengthens the resource isolation among containers and result in better overall performance.

To measure the success of our project, we will focus on completeness rather than performance. But we will examine if there are any possible optimization for the implementation and leave it as potential future works. Other possible topics include comparisons between static isolation approaches and our dynamic isolation approach, discussing the pros and cons of using different levels of encapsulations and virtualization technologies, and even recursive virtualization for more stable overall performance. We will also try to figure out the weight of different resources in the container environment, discussing trade-offs and complexities. Hopefully, the results and discussions will help us know the effectiveness and generality of the dynamic isolation approach for various working scenarios.

Here are some questions that our experiments will answer:

What kinds of scenarios will require strong resource isolation for applications running in the container environment?

Is dynamic resource allocation and isolation beneficial for the overall performance of applications running in the container environment?

What are the appropriate resource allocation approaches for applications with different characteristics?

How strong is the isolation provided by the dynamic isolation and allocation approach?

How stabilized has the performance of the container environment been achieved after adopting stronger isolation?

What is the overall overhead of our approach?

References

[1] "Linux containers." [Online]. Available: http://lxc.sourceforge.net

[2] "Docker." [Online]. Available: https://www.docker.com/

[3] M. G. Xavier, I. C. D. Oliveira, F. D. Rossi, R. D. D. Passos, K. J. Matteussi, and C. A. F. D. Rose, "A performance isolation analysis of disk-intensive workloads on container-based clouds," *2015 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, pp. 253–260, 2015.

[4] S. Ahn, K. La, and J. Kim, "Improving I/O resource sharing of linux cgroup for NVMe SSDs on multi-core systems," in *8th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 16)*, 2016.