An Image Caching Strategy in Docker Registry Memory for Optimizing Container Startup

Chen Zhang

School of Information Science and Technology, JiNan University, Guangzhou 510632, China.

316598424@qq.com

Abstract

With the rapid development of container technology, Docker has played an important role in cloud computing. When an enterprise deploys a large amount of business, it is necessary to retrieve the image from the Docker registry and allocate resources for it, then the cold start time overhead of the entire container increases. In order to reduce the startup time of the container, this article researches the process of image pulling to design an R-cache strategy in the Docker registry for optimizing container startup. This strategy caches some small hotspot images from the object storage server to the registry memory, and the memory of object store server is de-redundant to store more effective images, thereby reducing the image pulling delay and improving the container startup delay. The experimental results show that the R-cache strategy has a 10.2% to 36.8% improvement in image pulling delay over native Docker.

Keywords

Container technology, Docker registry, Cache, Memory de-redundancy.

1. Introduction

Cloud computing[1] has developed rapidly in recent years, providing good support for artificial intelligence[2], big data[3], and the Internet of Things[4]. In terms of managing cloud platform resources, container technology is emerging, and its light weight, flexibility, and efficiency have made more and more enterprises migrate application systems from traditional virtual machines[5] to containers[6]. Although the Docker container starts at a fast speed, in some large-scale task deployment scenarios, the delay overhead of all containers cold-started is very large. If one of the images is pulled too slowly, the overall process will be prolonged[7], and software development will be delayed, causing losses to software development[8].

Through deep analysis of the image pulling process, this paper proposes an R-Cache strategy in the Docker registry and reduce the image pull time. R-Cache caches smaller hotspot images in the registry memory to get closer to the user, while checking the memory of the object storage server, removing redundant images from the registry cache, saving more storage resources to store other images, which ultimately saves users the latency of images pulling.

2. Related work

Existing research on accelerating container startup is generally divided into two parts: container deployment optimization and container cold start optimization. Container deployment time refers to the time taken to pull the image from the Docker registry backend storage to the local. Container cold start Time refers to the time overhead for the Docker engine to allocate resources and namespace to start the container after the image is pulled locally. The related work is as follows:

2.1 Container deployment optimization

Harter[9] et al. researchers found that a container can be started with only 6.4% of its data set, so Slacker was designed to transmit the smallest image data set to the Docker engine at the initial moment of starting the container, and use lazy cloning and lazy propagation to Extracting the remaining required mirror data from NFS storage not only significantly reduces network I / O, but also improves cache sharing, thereby reducing container startup time. However, this solution puts

forward higher collaboration requirements for the Docker client and the Docker registry. Due to the lazy pull of the image, the entire container build time increases, so the continuous connection between the client and the registry will consume certain network resources.

2.2 Container cold start optimization

Oakes et al[10] studied high-level languages such as python through serverless platforms, even if it runs slower than C. Developers usually use a microservice architecture to split the application into multiple function instances and run them in isolated containers. For containers with short life cycles, frequent restart operations bring large overhead. Therefore, the author redesigned SOCK for Docker, used more lightweight isolation technology to break the performance bottleneck of network namespace and mount namespace, and used Zygote preprocessing technology to reduce the overhead of python installation.

3. R-cache strategy

3.1 Traditional image pulling

The native Docker ecosystem contains so many components that http request of image pulling often need to traverse all the components, and the high latency of multiple request jumps and forms a bottleneck for rapid container startup. The general pull request process is as follows:

1.After receiving the user's pull command, the Docker daemon process in the Docker engine sends a Get / v2 / <name> / manifests / <reference> request to the registry to obtain the metadata file and verify the data stored locally

2.The daemon process checks the layers field and does not change the mirror layer that has been stored locally. For the missing mirror layer that needs to be pulled, the daemon sends a Head request method to query whether there is the mirror layer in the back-end storage of the registry.

3.Get the redirect URL from the registry server through the Get request method, retrieve the mirror layer in the back-end storage through the new URL, and verify it locally using the sha256 algorithm to ensure the integrity of the mirror layer.

3.2 R-cache improves Docker registry

Docker Registry is the core component of the Docker ecosystem and is responsible for storing and distributing images. The registry not only allows developers to store images, but also acts as a mapping library and is deployed to production business process coordinators. In addition to the officially maintained Docker Hub, different cloud service providers provide public registry services for different image collections. Enterprises can also have a local dedicated registry for internal use only by the organization.

In the traditional Docker registry architecture, the registry is composed of components such as a load balancer, registry server, and object storage server. After the user issues a build container instruction, the Docker client sends a request to traverse all components. Figure 1 shows that in the traditional mirror pull mode, the multiple hops of the request largely determine the lower limit of the delay, especially when the back-end storage server is under high load, the performance degradation will greatly affect the efficiency of image pulling.

The R-cache strategy stores a part of the image in the registry memory, and rationally uses the free resources of the registry to reduce the load of the back-end storage server. At the same time, R-cache judges that the request hits the registry cache and returns to the mirror layer. Compared with the traditional request process, it reduces the number of request jumps and greatly reduces high latency. In order to ensure the performance of the R-cache strategy, we synchronously detect the memory usage of the object storage server, and replace the existing stored image in the registry server with a new hotspot image.

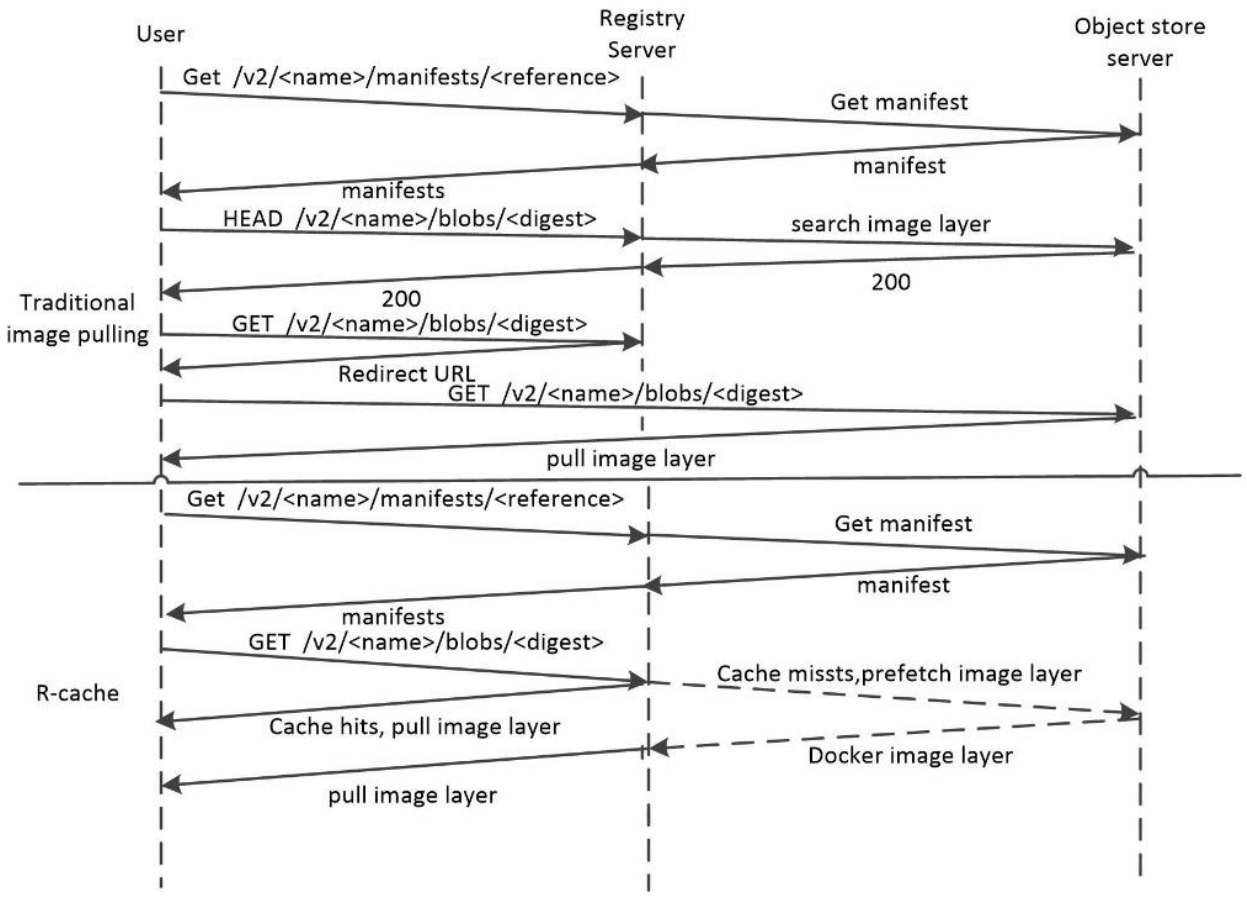

Fig. 1 Traditional image pulling and R-cache strategy

4. Experiment

4.1 Experimental environment and trace data

This experiment uses Inter Core i7-6700 CPU, 24GB DDR3 RAM, and 7200RPM 500GB SATA hard disk, and installs the operating system CentOS7.3, Linux kernel version is 3.10.0, Docker version is 1.12.0, corresponding API version is 1.27 , The compiled language python version used is 3.5.2. The specific parameters of the experimental environment are shown in Table 1.

Table 1 The parameters of experiment environment

Component	Description
CPU	Inter Core i7-6700
Memory	24GB
OS	CentOS 7.3
Kernel	3.10.0-514.16.1.e17
Docker	API Version 1.27

The dataset uses SNIA trace data files to test the performance of R-cahce. SNIA trace data files is a real workload data set collected from the IBM cloud registry, including from June 20, 2017 to February 9, 2017 The five different geographical registries (the other two are the registries used internally by IBM) processed a total of more than 38 million requests, and the total number of transmissions exceeded 181.3 TB. Since some information in the original log could not be made public, redundancy was removed After the fields are processed anonymously, the total size of the dataset is 22.4GB. The data set consists of http requests. Each request has fixed attributes, including the host number of the registry server that processes the request, the response time of the request, the HTTP request method, the user IP address, the requested URL, the Docker version used by the user, Request status, request data size, request id, request timestamp. Because the prefetch strategy

proposed in this article only involves the operation of the metadata file and the mirror layer, we only retain the data pull and upload requests after processing the data set.

4.2 Experimental results

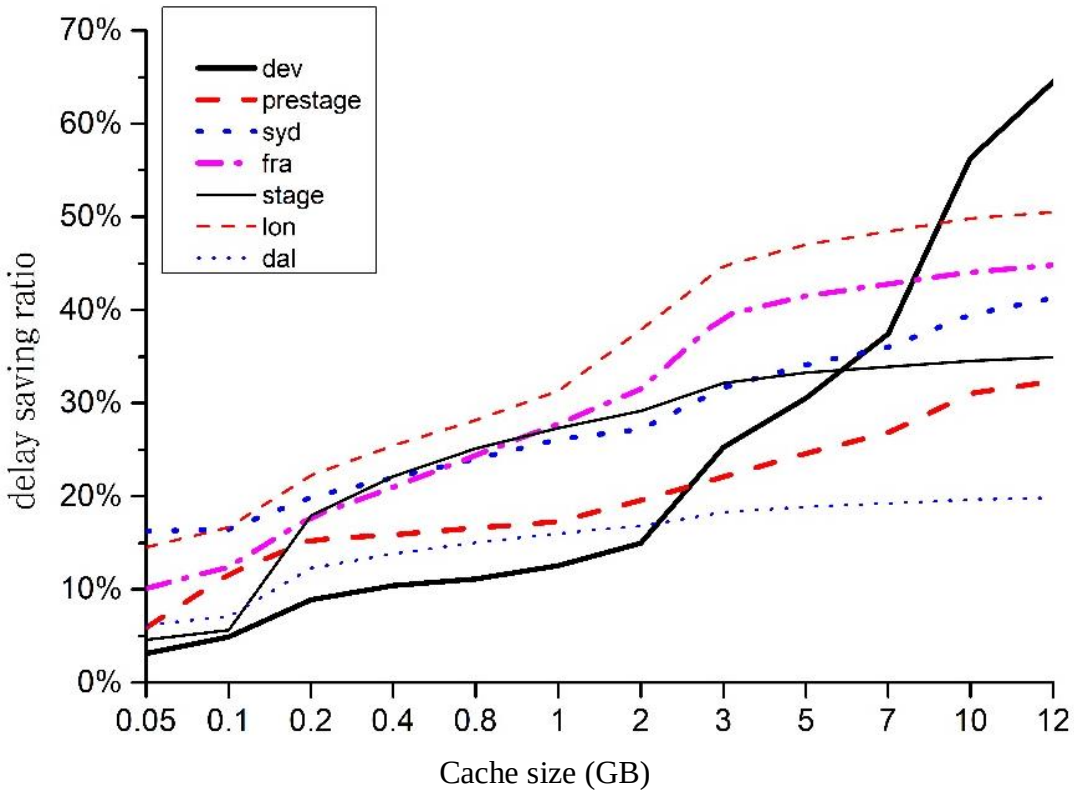

Fig. 2 test R-cache in 7 data trace

Figure 2 shows the latency savings rate of the R-cache strategy in 7 data sets, and the cache size is set to 50M to 12G. It is flexible to show the performance of the R-cache strategy when the registry server resources change, and at the same time the traditional. The mirror pull process is used as a benchmark.

The experimental test results show that from a horizontal perspective, as the cache capacity continues to increase, the R-cache strategy performs well in all data sets, and most data sets gradually approach the limit of the latency savings rate. In dev data, Centralized, the performance of R-cache increases with the increase in capacity. This is because the dev data set is the registry data used internally by IBM. The internal developers of the company use a small part of the mirror to form a relatively large amount of memory in the registry. Many hotspot mirrors. From a vertical perspective, even if the cache capacity is limited to a very small amount, the R-cache strategy can still save users an average of 10% latency, which reflects the value of this research. Based on comprehensive experimental results, R-cache can save 10.2% to 36.8% latency overhead compared to the traditional image pull process.

5. Conclusion

With the development of virtualization technology, more and more scenarios use Docker for task deployment. Because the container cold start needs to retrieve the image from the storage backend of the Docker registry, this process causes a huge delay overhead and severely slows down the business process. The existing solutions have certain shortcomings. This solution focuses on the registry components and proposes an R-Cache strategy. The hotspot image is cached from the object storage server to the Docker registry to simplify the pull process, and the storage server memory is removed. Redundant to hold more valid images. The experiment proves that the R-cache strategy effectively

reduces the image pull latency, which saves 10.2% ~ 36.8% latency overhead compared to native Docker, and effectively speeds up task deployment in containers.

References

[1] Hayes, Brian. "Cloud computing." (2008): 9-11.

[2] Gunning D. Explainable artificial intelligence (xai)[J]. Defense Advanced Research Projects Agency (DARPA), nd Web, 2017, 2.

[3] Zaharia M, Xin R S, Wendell P, et al. Apache spark: a unified engine for big data processing[J]. Communications of the ACM, 2016, 59(11): 56-65.

[4] Botta A, De Donato W, Persico V, et al. Integration of cloud computing and internet of things: a survey[J]. Future generation computer systems, 2016, 56: 684-700.

[5] Suzuki A T, Johnson N S, Gillette J, et al. System and method for deploying a virtual machine: U.S. Patent 9,547,485[P]. 2017-1-17.

[6] Felter W, Ferreira A, Rajamony R, et al. An updated performance comparison of virtual machines and linux containers[C]//2015 IEEE international symposium on performance analysis of systems and software (ISPASS). IEEE, 2015: 171-172.

[7] Oakes E, Yang L, Zhou D, et al. {SOCK}: Rapid task provisioning with serverless-optimized containers[C]//2018 {USENIX} Annual Technical Conference ({USENIX}{ATC} 18). 2018: 57-70.

[8] Wang L, Li M, Zhang Y, et al. Peeking behind the curtains of serverless platforms[C]//2018 {USENIX} Annual Technical Conference ({USENIX}{ATC} 18). 2018: 133-146.

[9] Harter T, Salmon B, Liu R, et al. Slacker: Fast distribution with lazy docker containers[C]//14th {USENIX} Conference on File and Storage Technologies ({FAST} 16). 2016: 181-195.

[10] Oakes E, Yang L, Zhou D, et al. {SOCK}: Rapid task provisioning with serverless-optimized containers[C]//2018 {USENIX} Annual Technical Conference ({USENIX}{ATC} 18). 2018: 57-70.