**HotSpot & Cache: An Optimization Method for Small Objects Storage in SWIFT of OpenStack Cloud**

Xueming Qiao\(^{1,a}\), Xue Han\(^{2,b}\), Dongjie Zhu\(^{3,c,*}\), Yuan Zhang\(^{1,d}\), Haifeng Sun\(^{1,e}\), Mingli Yin\(^{1,f}\), and Xiangkun Zhang\(^{1,g}\)

\(^1\) State Grid Shandong Electric Power Company, WeiHai Power Supply Company
\(^2\) Heihe University
\(^3\) School of Computer Science and Technology, Harbin Institute of Technology, Weihai

\(^{a}\) 13561832200@163.com, \(^{b}\) hanhuaer@yeah.net, \(^{c}\) zhudongjie@163.com, \(^{d}\) 5118518@qq.com, \(^{e}\) 1399151689@qq.com, \(^{f}\) 13305310986@163.com, \(^{g}\) 1372160955@qq.com

*corresponding author’s email: zhudongjie@163.com (Dongjie Zhu)

**Keywords:** Cloud computing, small files, performance optimization, SWIFT

**Abstract:** In the big-data era, cloud platforms such as OpenStack aim at the optimization of storage performance for large objects. However small file severely hurdles its performance. In this paper, a framework, namely HotSpot & Cache, is proposed to optimize the storage performance for small objects in SWIFT of OpenStack cloud. Double-proxy architecture is adopted for HotSpot & Cache architecture, including an external proxy node, an internal proxy node, and a dedicated cache server. Storage performance optimization for small objects is accomplished in the unit of partitions in storage nodes, hot-spot partitions are predicted and cached in the dedicated cache server. A series of facilities are implemented for small objects storage performance optimization, e.g. Sampling & Statistics, Hot-Spot Prediction, and a Request Redirection. The access frequency and the size of a partition is periodically sampled and statistically accumulated to predict the hot-spot of a partition for the next interval. Experimental results demonstrate the effectiveness of the proposed HotSpot & Cache architecture.

1. **Introduction**

It is estimated that data will increase at a 58 percent rate each year, which means that in the year 2020 the amount of global data will exceed 40ZB that is forty-four times of the year 2010. For big data, small files with size less than 1MB takes an overwhelming dominance. Global data is attacked and stored in cloud platforms, small files such as data, image and audio takes over a third of the overall data, in particular small files become the hot-spot for prevalent applications [1]. For example, the images and static contents of Internet surfing, the pictures in Taobao with an amount of over twenty billion with average size of 15KB, the small objects in Facebook with the amount of over sixty billion, and the large amount of thumbnails in Youtube [2][3].

The following factors contribute to the storage performance bottleneck for small files. One the one hand, cloud platforms and distributed storage systems, such as GFS, HDFS and OpenStack aims at optimizing the storage performance for large files. On the other hand, local file system such as EXT
and XFS, are designed as well to optimize large files performance. Consequently, random read and write operations for large amount of small files will incur a great performance overhead [4][5].

In this paper, a framework, namely HotSpot & Cache, is proposed to optimize the storage performance for small objects in SWIFT of OpenStack cloud. A double-proxy architecture is used, which includes an external proxy node, an internal proxy node, and a dedicated cache server. Optimization is accomplished in the unit of partition, hot-spot partitions of storage nodes are predicted and cached in a cache server. A series of facilities are implemented for small objects storage optimization, e.g. Sampling & Statistics, Hot-Spot Prediction, and a Request Redirection. The access frequency and the size of a partition is sampled and accumulated periodically to predict the hot-spot partition for the next interval, for which multiple intervals are adopted. The overall architecture and detailed implementation is depicted in the following sections.

2. HotSpot-Cache Architecture

2.1. Double-Proxy Architecture

The primary idea of storage performance optimization for small objects is to predict the hot-spot data in the storage nodes according to statistics, then hot-spot partitions in storage nodes are cached in a dedicated Cache Server. Accordingly the hot-spot requests are redirected to Cache Server instead of Storage Nodes, greatly relieving the request pressure of the Storage Nodes and optimizing the storage performance with in-memory access time. Note hot-spot prediction is made in the unit of partitions in storage nodes.

The overall architecture of HotSpot & Cache framework is depicted in Figure 1. In contrast to the original architecture of OpenStack SWIFT a double-proxy architecture is introduced, including an External Proxy, an Internal Proxy with a dedicated Cache Server employed. The significance of double-proxy is two fold. First, the External Proxy is of great importance to the security of the architecture in terms of Authentication. Second, the External and Internal proxies perform in a cooperative manner accomplishing in-memory performance by following steps such as hot-spot sampling, hot-spot value quantitative prediction and request redirection. In fact, External Proxy, Internal Proxy and Cache Server can be implemented in the form of a cluster.

![Figure 1 The Double-Proxy Architecture of Hotspot & Cache for Small Objects Storage Optimization.](image)

A double-proxy architecture is introduced, the following components are included such as External Proxy nodes, Internal Proxy nodes, and a dedicated Caching Server cluster.

The specific modules are described in Figure 2. A dedicated Cache Server is adopted to cache the hot-spot partitions in storage nodes of SWIFT. In addition, a series of supporting facilities are employed, including Sampling & Statistics, Hot-Spot Value Prediction, and Request Redirection. Sampling & Statistics is composed of two modules, e.g. Hot-Spot Sampling and Space-Sampling.
Hot-Spot sampling periodically samples the number of accesses for each partition in storage nodes. The Space-sampling module dynamically acquires the size of each partitions in the storage nodes. And size as well as number of accesses for each partition are fed into the Hot-Spot Prediction module, for which a multiple cycles statistical algorithm is employed to quantitatively predict the Hot-Spot for each partition and designates it as the hot-spot in following interval.

Figure 2 Function modules of Hotspot & Cache architecture. A cluster of dedicated Cache Servers is adopted to cache hot partitions in Storage Nodes of SWIFT. A series of supporting facilities are provided, including Sampling and Statistics, Hot-Spot Prediction, and Request Redirection. Sampling & Statistics is composed of two modules, namely Partition Hot-Spot Sampling and Partition Space Sampling respectively. The Hot-Spot Sampling module lies in external proxy, while the Space Sampling module is in the Internal Proxy.

After that a hot-spot partition list is generated in the form of a Hot-Spot Hash. In case of a request packet is received, it is parsed first and then according to the Hot-Spot Hash the request is redirected to Cache Servers or Storage Nodes by the Redirection Module in External Proxy. In each sampling interval the Cache Server(s) is updated according to the Hot-Spot Hash list as well. The cold partitions are invalidated and evicted and the hot partitions are inserted and updated.

2.2. Hot-Spot Sampling & Statistics

Hot-spot prediction is made in terms of partitions. Hot-spot Sampling & Statistics functionality is composed of two modules, namely Hot-Spot Sampling and Space-Sampling. The Hot-Spot Sampling module lies in the External Proxy, while Space-Sampling module in the Internal Proxy. Hot-Spot Sampling periodically gathers the number of accesses times for all partitions in the Storage nodes of SWIFT, whilst Space-sampling acquires the size of partitions.

The process of Hot-Spot Sampling acts as follows. The first time Hot-Spot Sampling module starts, an in-memory hash table is generated, namely HotSpot-Hash for hot partition statistics, which can be serialized to disk later. When a request is arrived, the requested object is first resolved by External Proxy. Then the object is transformed to corresponding Partition-ID. For each partition in Object Storage Nodes of SWIFT, an Partition-ID is generated using MD5 algorithm. The Partition-ID is hashed to index the in-memory HotSpot-Hash table. If the partition is not in HotSpot-Hash, it is inserted in it, otherwise its access frequency is accumulated and updated accordingly. Note that lock mechanism must be employed to the HotSpot-Hash table because of the multi-threading characteristic of OpenStack. In SWIFT, an object is stored in multiple replicas in different nodes. Accordingly, the Space-Sampling module should be carefully designed to accumulate the size of each partitions in storage nodes. The Space-Sampling is located in Internal Proxy and it is implemented using socket.

Above all, the Hot-Spot Sampling & Statistics accomplishes the information such as the size as well as the access times for each partition. Then it is fed into the Hot-Spot Prediction module, for
which a multiple interval statistics algorithm is employed to quantitatively predict the hot-spot value for each partition and designates it in the following time interval.

### 2.3. Hot-Spot Prediction

In present work the Binary Linear Regression algorithm is used for Hot-Spot Prediction. The Hot-Spot Sampling & Statistics is accomplished in multiple cycles, namely Multi-Cycle Statistics by us. For hot-spot prediction, different update time intervals are adopted. For example, three update frequencies are used: T1 is set to 60 minutes, T2 is set to 30 minutes, whilst T3 is set to 10 minutes in this work. Tsample is the sampling interval, e.g. 1 minutes, while Thot-spot is the Hot-Spot Prediction cycle, e.g. 15 minutes. The Hot-Spot Prediction process is depicted in detail.

```python
    while True:
        time_interval = int(time.time())
        if time_interval == T_sample
            update partition statistics in Binary Linear Regression algorithm
        if time_interval == T3
            update partition statistics in of T3
        if time_interval == T2
            update partition statistics in of T2
        if time_interval == T1
            update partition statistics in of T1
        if time_interval == Thot-spot
            Hot-Spot Prediction
        sleep(Tsample)
    done
```

### 2.4. Request Redirection

![Flow Chart of Request Redirection](image)

Figure 3 Flow Chart of Request Redirection
When a request packet is received, first it is parsed by the external proxy. According to Hot-Spot Hash, it is either redirected to the Cache Server in case of a hot-spot partition is requested, otherwise it is forwarded to Storage Nodes. The revised flow chart after Redirection is depicted in Figure 3. Due to space restriction, the detailed implementation is introduced in the future.

3. Evaluations

In this work, Cache Server is implemented using Varnish with the cache size of 200 M memory, for which a 20 percent of hot partitions are cached. In each test, 1000 small objects are created using Cosbench each with size less than 1MB [6]. The experiment results are depicted in Figure 4.

![Figure 4 Response Time of the Optimized Architecture](image)

Figure 4 Response Time of the Optimized Architecture. X coordinate designates concurrency, Y coordinate describes response time in unit of millisecond.

As expected the response time increases with concurrency. Experimental result demonstrates its effectiveness. Optimization effect increases with concurrency because of more requests are redirected and forwarded to Cache server, with an optimal response time of about 40%. Analysis shows the following factor leads to the efficacy of HotSpot & Cache framework: the hot-spot request effectively eliminates the ring file accesses overhead twice, in addition the object is accessed with in-memory latency avoiding time consuming disk operations and network transmissions.

4. Conclusions

Aiming at access performance optimization for small files, a framework namely HotSpot & Cache is put forward. Double-proxy architecture is adopted in the framework, including an External Proxy, an Internal Proxy, and a dedicated Cache Server. Each component can be configured as a cluster, and the independence of cache server brings the HotSpot & Cache architecture much scalability. A series of facilities are employed, e.g. Sampling & Statistics, Hot-Spot Prediction, and Request Redirection. Hot-spot prediction is made in terms of partitions. Two kinds of parameters are sampled and accumulated statistically: the number of accesses times and size of partitions. Then a multi-cycle prediction is made to prediction the hot partitions. Experimental result demonstrates that the optimized architecture acquires satisfactory efficacy.

Acknowledgments

Thanks to the students of HIT at Weihai and staff of HuaQiang Co. This work is supported by project under Grant no. 520613170002, SGSDWH00YXJS1700522, SGSDWH00YXJS1700270, the Fundamental Research Funds for the Central Universities (Grant No.HIT.NSRIF.201714), Weihai Science and Technology Development Program (2016DXGJMS15) and Key Research and Development Program in Shandong Provincial (2017GGX90103)
References