InterFS: An Interplanted Distributed File System to Improve Storage Utilization

Peng Wang [*]
Peking University
wang_peng@pku.edu.cn

Le Cao Chunbo Lai
Baidu Inc.
{caole, laichunbo}
@baidu.com

Leqi Zou Guangyu Sun [*]
Peking University
{zlqiszlq, gsun}
@pku.edu.cn

Jason Cong [†]
University of California,
Los Angeles and
Peking University
cong@cs.ucla.edu

Abstract

Resource under-utilization is a common problem in modern data centers. Though researchers have proposed consolidation techniques to improve utilization of computing resources, there still lacks an approach to mitigate particularly low utilization of storage capacity in clusters for online services. A potential solution is to "interplant" a distributed storage system together with the services on these clusters to leverage the unused storage. However, avoiding performance interference with existing services is an essential prerequisite for interplanting. Thus, we propose InterFS, a POSIX-compliant distributed file system aiming at fully exploiting the storage resource on data center clusters. We adopt intelligent resource isolation, peak load dodging, and region-based replica placement schemes in InterFS. Therefore, it can be interplanted with other resource-intensive services without interfering with them, and amply fulfill the storage requirements of small-scale applications in the data center. Currently InterFS is deployed in 20,000+ servers at Baidu, providing 80 PB storage space to 200+ long-tailed services.

Categories and Subject Descriptors H.3.4 [*Information Storage And Retrieval*]: Systems and Software

Keywords distributed file system, resource utilization, stroage

[*] Peng Wang and Guangyu Sun are also members of the PKU/UCLA Joint Research Institute in Science and Engineering.

[†] Jason Cong is also a distinguished visiting professor at PKU, and co-director of the PKU/UCLA Joint Research Institute in Science and Engineering.

APSys 2015, July 27–28, 2015, Tokyo, Japan.
Copyright © 2015 ACM 978-1-4503-3554-6/15/07…$15.00.
http://dx.doi.org/10.1145/2797022.2797036

1. Introduction

With rapid advance of Internet services, which include web search, e-commerce, social networking, etc., the number of services hosted in modern data centers keeps increasing. It also results in an increasing demand for both computing resources (e.g. processors and memory) and storage resources (e.g. disks) employed in the data-centers. Unfortunately, the utilization of these resources remains at a very low level in many modern data centers. A recent investigation shows that the total CPU utilization in Twitter's and Google's data center stay at 20%–30% without significant improvement in the past decade [4–6]. The current utilization of storage capacity is about 33% in Baidu's online-service clusters. Such low resource utilization poses a serious challenge on reducing the total cost of ownership (TCO) of data centers [14]. For the sake of the cost-efficiency, it is critical to improve the resource utilization in data centers.

One major reason for resource under-utilization is that most Internet companies usually employ a huge number of resources over-provisioned for many important services. We call these services as *high priority services*, such as web search, social networking news feeds, online ad recommendation, real-time anti-cheating analysis, etc. In order to handle the peak traffic in the worst case, the total number of servers deployed is determined by the maximum resource demand in some special scenarios (e.g. World Cup Final, Black Friday, DDoS attack). Over-provisioning for the worst-case results in serious resource under-utilization in normal working time. The storage utilization is especially low for online processing clusters, usually less than 25% in Baidu, since copious disks are installed in a single server to comply with the IOPS requirement of the worst case. The imbalance between disk throughput and storage capacity commonly occurs in various data centers.

Recently, several approaches have been proposed to improve utilization of computing resources (e.g. CPU and memory) through co-locating a mix of services on the same server [6, 10, 16]. However, how to improve the utilization of storage capacity is not well addressed yet. In fact, it is impractical to mix-deploy these high-priority services to improve utilization of storage capacity. The major concern of services co-location is about the quality-of-service (QoS). Since most software of these services are developed without considering resource sharing, resource competition cannot be fully avoided even with careful management, especially when multiple services on the same server enter the peak traffic scenarios at the same time. Thus, it may result in substantial performance degradation.

Instead of mixing these services together, it is more practical to co-locate a special storage system with a primary service on the same server. To minimize the interference, the storage system should monitor run-time resource requirement of high-priority services so that it can adapt its own storage service on this server to avoid resource competition. At the same time, the quality of this storage service should also be maintained in an acceptable level. To this end, the storage system should only serve those applications with low I/O throughput intensity. Interestingly, co-locating such a storage system with high-priority services on the same server is quite similar to the practice of interplanting two different crops, which are carefully selected to avoid competing with each other.

Fortunately, we have observed that there exist a variety of small-scale applications that can be served by this interplanted storage system, such as mobile apps and lightweight web applications. They are named as “long-tailed” applications in this work. The reasons are explained as follows. First, the data volume of one long-tailed application is normally relatively low, compared to other workloads. However, the total number of these long-tailed applications is quite dominating (e.g. about 90% in Baidu’s data centers) in a data center. An illustration is shown in Figure 1. For example, most mobile apps need a backend in data centers to store their data. Thus, the total data volume of these applications is still considerable (e.g. about 15% in Baidu’s data centers). Second, the development and maintenance of most long-tailed applications heavily rely on the usage of files. In addition, they normally operate on small files and require low I/O throughput. The breakdown of these long-tailed applications serviced by InterFS is also presented in Figure 1.

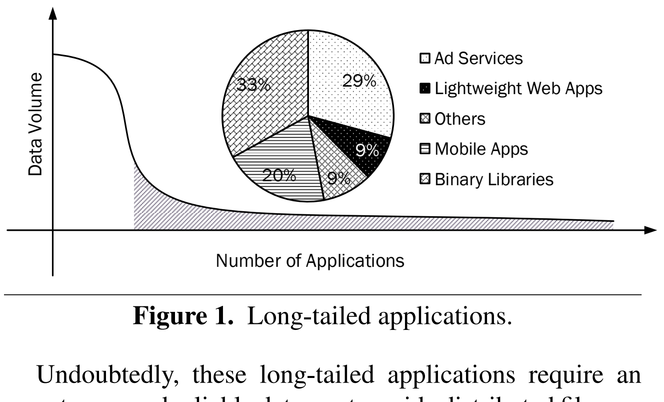

Figure 1. Long-tailed applications.

Undoubtedly, these long-tailed applications require an easy-to-use and reliable data-center-wide distributed file system. On the other hand, the storage capacity utilization can be improved if the distributed file system can be interplanted with high-priority services with negligible interference. To achieve a double-win, we have developed InterFS, a distributed file system that can be co-located with other workloads in online-processing clusters to improve the storage utilization. The InterFS is almost POSIX-compliant, and it is tailored for the access patterns of long-tailed applications: numerous small files, multiple-reads after an initial write, etc. More importantly, we adopt intelligent resource isolation, file server dodging, and replication placement schemes in InterFS. Therefore, it can be interplanted with other resource-intensive services and amply fulfill the storage requirements of long-tailed applications in the data center. Currently InterFS systems have been widely deployed in 20,000+ servers at Baidu, providing 80 PB storage space to 200+ long-tailed services. The largest one has over 6,000 nodes, 30 PB of storage capacity, and 90 million files.

2. System Overview

In this section, we first introduce basic features of InterFS and its overall architecture.

2.1 Features

The design of InterFS follows several disciplines, which are listed as follows,

- **Interface**: InterFS provides an almost POSIX-compliant file system interface to its clients. Most of the Linux file system APIs are supported, such as *read, write, sync, mkdir*. Only few uncommon functions (e.g. *truncate*) are not supported.
- **Mixed Deployment**: InterFS is designed to be interplanted with other high-priority services. Only the master node requires a dedicated machine. It means that InterFS file server, the clients, and other online services may run in the same server.
- **File Size**: Most files stored in the InterFS have relatively small sizes. According to Baidu’s statistics, the percentage of files smaller than 1 KB is 49.3%, and only 0.8% files are larger than 10 MB. The total number of files is more than 10 million in this cluster. Unlike other file systems targeting large scale files (e.g. GFS [8]), InterFS is designed to provide efficient process of small files. Thus, for applications served by InterFS, the I/O throughput is not our chief concern.
- **Access Pattern**: Although InterFS supports the random access on files, the principal access pattern is the multiple reads after an initial write, and the append write. The number of opened files is very small (less than 1%) compared to the total number of files for a cluster at a given time. Thus, from the data-center-wide perspective, the total amount of computing resources utilized by InterFS is very low. This access pattern makes InterFS feasible for interplanting.

- **File Opening Mode**: Considering the fact that a single file opened for write by multiple applications is really rare, InterFS only allows a single file to be opened once for write by one client (application) at the same time. This feature can be utilized as a global lock.
- **Strong Consistency**: Many long-tailed applications desire strong consistency, which means once the file is successfully written, all clients can read the contents of this operation in real-time. InterFS provides the strong consistency mode to make the distributed file system transparent to long-tailed applications.

2.2 Architecture of InterFS

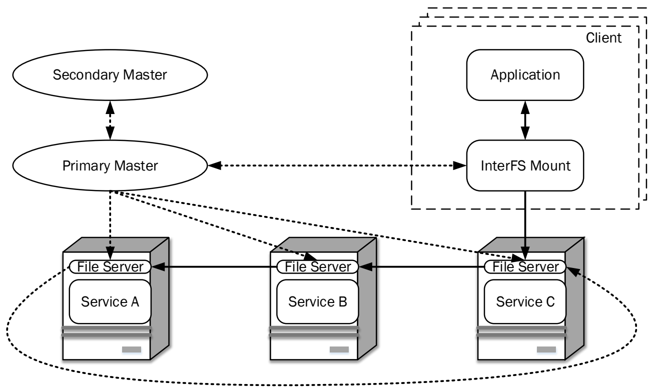

Figure 2. InterFS system architecture.

As shown in Figure 2, an InterFS cluster is composed of a single master node and multiple file server nodes. It can be accessed by multiple clients, on which the long-tailed applications are running. Only file servers and clients are able to be interplanted with other high-priority applications on the same server.

Similar to other single-master-based distributed system (e.g. GFS), each file in InterFS has multiple replicas (typically three) scattered on different file servers. Each replica, contained in a *file unit*, is stored in a single file server. Each file server contains multiple file units, the number of which relies on the disk space and memory capacity available on this server. For example, five million files require less than 2 GB memory to store the metadata. There is a *file engine* in each file unit, which is a storage engine to flush the file unit to the local persistent storage. We reuse the log-structured-merge tree (LSM-tree) based file engine in the CCDB Table system of Baidu to simplify our design [13]. The file unit also takes care of replication.

As mentioned before, the sizes of most files stored on InterFS are in the scale of several KBs. Thus, for these less than 1 MB, each file only occupies the storage space equal to its real file size. File stripping is also supported in InterFS. When the file size is larger than a threshold α (default 1 MB), each file is stripped into blocks with a maximum size of α. Note that the last block less than $alpha$ is not extended to a "full block". The block size can be adjusted dynamically in the range of $1 \sim 16$ MB. Obviously, increasing the block size can help reduce overhead of metadata. However, the I/O pressure of transferring a block also increases at the same time. This is the reason why the block size is limited to 16 MB, which is much less than those in other distributed file system (e.g. GFS).

We minimize master's involvement in data access to prevent it becoming a bottleneck. The data never go through the master. Instead, a client inquires of the master about the file locations. It then contacts the file server directly for subsequent operations. Note that the mapping between file units and blocks are not preserved in the master node. Because each replica of the file is only stored in one file server, the block mapping metadata are stored in the same file server. Other types of metadata, such as *mtime*, *ctime*, file size, execution bit, etc., are also stored in the same node. This is reasonable since the small files consist of few blocks. Such design provides several advantages: 1) It substantially reduces the memory usage of master node; 2) Many metadata query operations bypass the master and fall onto the file servers to further reduce the pressure of master. We move the metadata operations on files, *getattr*, to file server and further reduce the total operations of master by 9% in the system; 3) Since all read/write requests from clients are directly handled by file servers, saving the other metadata like *mtime*, *ctime*, file size in file servers will further avoid the interaction with master.

The master maintains three types of metadata: the namespace, access control list, and the location information of file. The namespace stands for the overall directory tree structure of the file system, and the location information is the mapping from files to it replicas on file servers. The masters keep all metadata in memory, but all changes will be persistently recorded. The reason is that InterFS is designed for numerous small files in the online clusters. It is unacceptable to take as long as ten minutes to gather location information from thousands of the file servers after master restarting. Master also manages the access control list and checks the permission for each access.

In the following sections, we introduce detailed optimization techniques to avoid interference in interplanting.

3. Optimization for Interplanting

In order to co-locate InterFS with high-priority services without interference, we propose several optimization schemes, which are introduced in this section.

3.1 Resource Isolation Scheme

InterFS uses an adaptive scheme of resource isolation, which can dynamically adjust the threshold of each kind of resource in file servers according to the current load of file servers and the service-level agreement (SLA) of services co-located within the same server. There is a monitoring process called *file server agent* to manage the file server daemon process on each file server node. Isolation techniques for different resources on file servers are described as follows,

- **CPU**: InterFS uses *taskset* to bind its daemon process to a specific CPU core. Since InterFS background process is a storage service, it usually has little CPU usage and can only occupy at most single core.
- **Memory**: Since the metadata is stored in the file server, the memory usage is about several hundreds of MBs. We use *cgroup* to set a limit on the total memory usage.
- **Network bandwidth**: InterFS adopts a speed limit kernel module based on token bucket algorithm to restrict total TCP traffics that file server generates. InterFS also has the internal traffic control mechanism.
- **Disk throughput**: InterFS constantly monitors its throughput. It employs a fine-grained token-based throttling mechanism for disk throughput control. The tokens are generated by file servers. A client needs to acquire a token before accessing files on a file server. The rate of generating tokens is adapted according to the run-time throughput. When the throughput reaches a threshold, the client will not acquire a token but receive an error message from the file server. Then the client will slow down the rate of retry. This is a complete negative feedback process.
- **Disk space**: InterFS sets an individual quota for each disk in file server. When the high-priority services on a file server require more disk space, the rebalance process will be triggered, and the file replicas of InterFS will be migrated to another file server. When the disk space fall below a threshold, the file server will be killed by file server agent and the data on this server will be deleted.
- **Number of *inodes***: When the number of inodes of a disk exceeds a given limit, this disk is no longer used.
- **Page cache**: InterFS can be switched to *no-caching mode* in which page cache provided by OS is bypassed since some performance sensitive services require exclusive usage of the page cache.
- **Cross-data-center bandwidth**: Although most of the traffic is inside the cluster, InterFS also has a monitoring module to restrict the cross-data-center bandwidth usage to a safe range.

3.2 Peak Load Dodging Scheme

With the resource isolation schemes, InterFS limits the resource usage on the file servers to ensure the original high-priority services function normally. However, the online services may have unpredictable traffic occasionally. In such scenarios with in-rush traffic, the InterFS file server will relinquish its resources to the high-priority service. The file server will automatically revive after the peak traffic period has passed.

After starting the file server, a file server agent watches the overall load and resource usage in the background. When it finds the resource usage is higher than a threshold value (denoted as Th_{high}) during several continuous sampling windows, the agent will shut down the file server and report the event to the master. Master will delay the recovery procedure with other replicas in order to wait for the self-recovery of this suicidal file server. When the agent notices the resource usage is lower than a threshold value (denoted as Th_{low}), it will restart the file server process and resume the service.

If a file server uses the shutdown interface to dodge the peak load, it will retain all status for fast recovery. If this server is restarted during the period called *surviving windows*, it can be fully restored to the original state without notifying the master. The clients that use this file server will only experience a small increase of latency, which is very helpful for smooth upgrade. If this server does not revive during this period, the master will redirect the affected clients to other replicas and rebuild the replication chain for future requests.

On the contrary, if a file server is terminated abnormally, it cannot rejoin the cluster automatically. The file server will try to intercept all exception signals and send the error message to the master before it exits as far as possible. If the master receives the abnormal status report from a file server, or does not detect the HeartBeat message, the master will regard this file server as dead and start the recovery process. In InterFS, the recovery on a file server is divided as two phases: metadata recovery phase and data recovery phase. Only the metadata recovery phase will block the write requests for a short time. During the data recovery phase, the write request can be handled since the file engine is log-structured.

3.3 Region-based Replica Placement Scheme

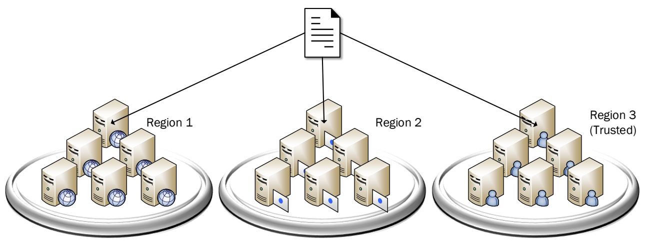

Figure 3. An illustration of region-based placement.

We observed that the activity patterns of the servers in data center are diversified. The maintenance activities, such as software updating, upgrading, machine restarting, or OS reinstalling, are often conducted in the granularity of a product line, i.e. the machines belong to the same product line are usually operated simultaneously. Furthermore, the faults caused by software bugs mostly affect servers of the same product lines. Thus, if we locate all replicas in the machines owned by the same product line, it may result in data loss since all replicas might be down at the same time.

Thus, as shown in Figure 3, we group the machines into *regions*, similar to fault domains, according to the product lines. InterFS never puts all replicas in the same region. Instead, it guarantees one region contains no more than one replica of the same file. Consequently, data will be safe even all machines in one region crash down.

Note that the availability of the region differs significantly. Some regions have a higher availability than others. We call them *trusted regions*. For a specified file, the client can configure the number of replicas and how many regions it should be distributed into. InterFS ensures that at least one replica is allocated to a trusted region. For files frequently accessed, we can allocate them to light-load or dedicated regions.

Let p_i denote the probability of failure in the ith region, q_i denote the conditional probability of each server failure in this region. If we place three replicas in three different regions, the probability that all three servers fail will be:

$$P_1 = \sum_{\substack{1 \le i,j,k \le n \\ i \ne j, i \ne k, j \ne k}} \frac{p_i q_i \cdot p_j q_j \cdot p_k q_k}{\binom{n}{3}}$$

If we keep all three replicas in the same region, the probability of failure will be:

$$P_2 = \sum_{i=1}^{n} \frac{p_i q_i^3}{n}$$

We can easily prove that $P_1 < P_2$, which shows region-based replica placement scheme has lower probability of losing data.

When file units are created, master will select a proper file server according to its available storage capacity, the number of total replicas, and the region maps. The file server with more idle resources has a higher priority. InterFS also has a rebalancing mechanism to make the data volume and the number of files on each file server remains balanced. Furthermore, InterFS also tries to keep the number of primary unit on each file server balanced so that the read requests are distributed across the whole cluster uniformly.

4. Evaluation

In this section, we first describe our experimental setup for evaluation of InterFS. Then we present basic evaluation results of InterFS without interplanting. After that, we measure the performance of InterFS under the circumstances of mixed deployment. Finally we show real statistics numbers from the production clusters used in data center of Baidu.

4.1 Experimental Setup

We measured performance on a InterFS cluster consisting of one primary master, one secondary master, 19 file servers, and up to 50 clients. All the machines are configured with one 2.40 GHz Intel Xeon E5620 8-core processor, 32 GB memory, ten 2 TB 7200 RPM disks, and 1 Gbps full-duplex Ethernet connection.

4.2 Evaluation Results without Interplanting

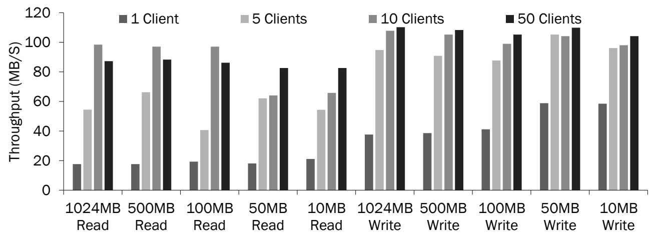

Figure 4. Aggregate throughput of InterFS with various numbers of clients.

We first evaluate the performance of InterFS without interplanting with other high-priority services. As shown in Figure 4, we evaluate the aggregate throughput of InterFS using different numbers of clients. Up to 50 clients download (read) or upload (write) files simultaneously using InterFS. The total size of data accessed is fixed at 10 GB. Data are partitioned uniformly to multiple files ($10 \sim 1000$) for different cases. Read and write performance is evaluated separately. The maximum throughput is about 100 MB/s for both read and write operations. We can find that InterFS has a limited throughput due to network and disk I/O limits brought by the resource isolation schemes. Write performance is a little bit lower than read performance due to overhead of replication.

We then compare InterFS with two other distributed file systems with POSIX-compliant interface, MooseFS [3] and GlusterFS [1], in respect of performance under non-interplanting circumstances. In other words, we compare the maximum performance of these file systems. All experimental results are shown in Figure 5 for different file sizes and client numbers.

Compared to MooseFS, InterFS outperforms MooseFS when the file size is small (e.g. 50 B $\sim$ 1 MB cases). InterFS also achieves comparable performance for large file (1 GB case). In fact, for the multiple client cases, InterFS cannot beat MooseFS when all clients read the same file. However, InterFS achieves better performance when the clients read different files on different file servers. It is because a single file must be located in the same file server in InterFS. Thus, InterFS cannot benefit from concurrent block operations of the same file over multiple servers like that in MooseFS. When compared to GlusterFS, InterFS achieves better performance for large files (1 MB and 1 GB).

We also measure the performance of InterFS's namespace operations and make a comparison with other distributed file systems using a 20-node cluster. Figure 6 shows the metadata throughput (ops/sec) of the four common namespace-related operations, including *ls*, *rename*, *create*, and *delete*.

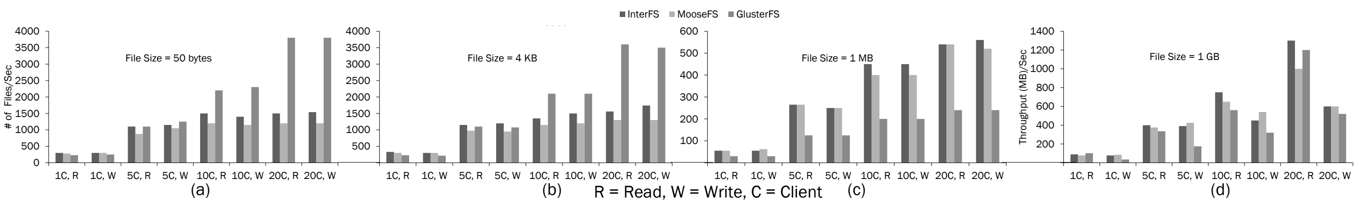

Figure 5. Comparison of performance without interplanting.

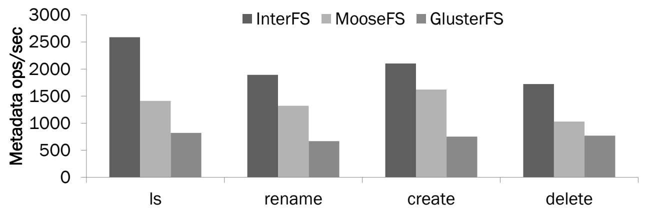

Figure 6. Comparison of namespace operation performance (ops/sec).

InterFS outperforms other systems in this evaluation. The *ls* operation that consists *readdir* followed by a *stat* of each file is an extremely common access pattern for long-tailed applications. There is an 1.83x speedup for InterFS over the MooseFS since the master is carefully optimized for queries of this type. On average, InterFS is 55% better than MooseFS in terms of namespace operations. InterFS achieves 2.75x higher metadata throughput than GlusterFS, since it is slow for center-less distributed file systems to modify the metadata across multiple machines.

To evaluate the recovery ability of InterFS, we record the whole recovery process when a file server is forced to shutdown. In real world system, there are about 0.5 million files stored on this server. After the failed server is detected, the total recovery time is less than 90 seconds. On average, more than 5,500 files can be recovered in one second.

4.3 Evaluation Results with Interplanting

In this subsection, we analyze the performance result of InterFS, MooseFS and GlusterFS in interplanting cases. We interplant these distributed file system with Memcached [7], which is a well-known in-memory key-value store widely used for online services, and LevelDB [9], a popular disk-intensive key-value store. We use Memcached and LevelDB to measure the impact of interplanting on network throughput and disk throughput, respectively. A sample workload extracted from Baidu's online service is used to evaluate these distributed file systems.

LevelDB's throughputs in 6 benchmarks are shown in Figure 7. The mixed deployment of LevelDB and InterFS has only 11% degradation of throughput on average, while the throughput of mixed deployment of LevelDB and MooseFS is 47% less than LevelDB alone. The throughput is 45% less in the mixed deployment with GlusterFS. Similarly, Figure 8 shows that the mixed deployment has a 10% degradation of the overall performance of Memcached, while the results of

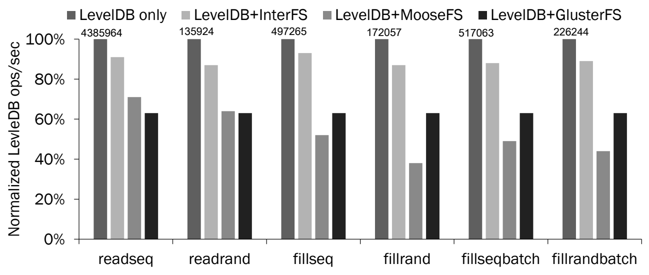

Figure 7. Impact on LevelDB performance when co-located with various distributed file systems.

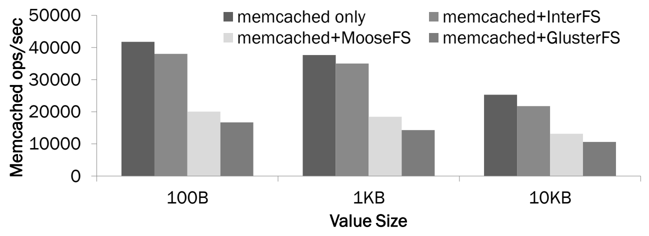

Figure 8. Impact on Memcached performance when co-located with various distributed file systems.

MooseFS and GlusterFS are 51% and 60%, respectively. It can be derived that InterFS has less interference of high-priority services than that of other two systems.

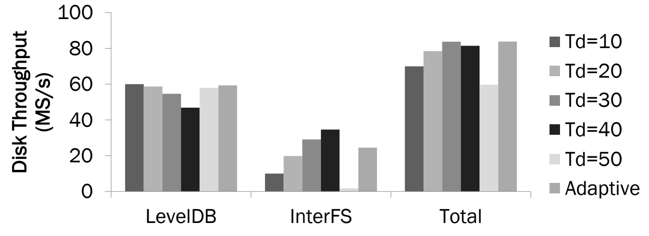

Figure 9. Impact of disk throughput threshold.

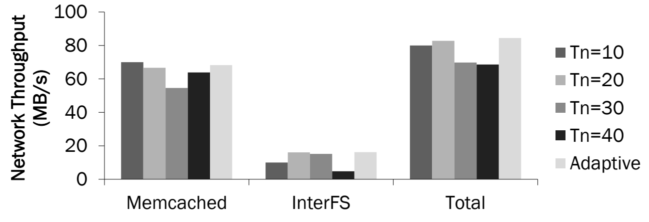

Figure 10. Impact of network throughput threshold.

We also compare the effect of fixed threshold of the disk throughput (T_d) and network throughput (T_n) with the adaptive threshold in InterFS. Figure 9 shows that the adaptive threshold achieves the best overall performance for one single file server. Note that when the threshold of disk throughput of InterFS is set to 50 MB/s, the total disk throughput will exceed the warning level sometime. Therefore, the peak load dodging mechanism is triggered and InterFS is shutdown soon afterwards. In this case, the throughput of InterFS is close to zero. The same trend on network throughput is shown in Figure 10. They both prove that the dodging mechanism takes effect and make sure that high-priority services can survive the peak pressure.

4.4 Statistics of InterFS in Real World

In this subsection, we provide real numbers for InterFS deployed in the real data center of Baidu. Detailed statistics of a typical InterFS cluster, such as number of nodes, number of files, disk space, etc., are shown in Table 1. The real IOPS, throughput and access latency are also presented.

File servers	556
Regions	11
Files	90 million
Available disk space	581 TB
Used disk space	388 TB
Total IOPS of file servers	20 283
Total throughput of file servers	808 MB/s
Average file access latency	0.47 ms

Table 1. Characteristics of a typical InterFS cluster.

Note that the space utilization of this cluster is 39.3%, which is 6% more than the average level. It demonstrates that InterFS can improve the storage utilization of data center. The gain is made within 8 months after the deployment of InterFS in this cluster. About 40 long-tailed applications have migrated to InterFS platform. We believe that the utilization will keep rising as more long-tailed applications embrace InterFS.

The number of InterFS operations over 24 hours on this cluster is shown in Figure 11. We can find that master OPs vary in the range of 2.2 to 4.5 million and file server OPs vary in the range of 0.7 to 1.8 million. We can also observe that the low request intensity happens during 3 pm and 11 pm.

The breakdown of operation in InterFS is shown in Figure 12. We can draw two conclusions. First, the *lookup* operation is dominating. Thus, it is worth applying optimization on its processing. Second, we can tell that a lot of *getattr* operations are offloaded from master to file servers. Note that the labels *mgetattr* and *fgetatter* are *getattr* operations on master and file servers, respectively.

Typical resource usage of file server in InterFS is shown in Table 2. The low resource usage guarantees InterFS can be interplanted with high-priority services.

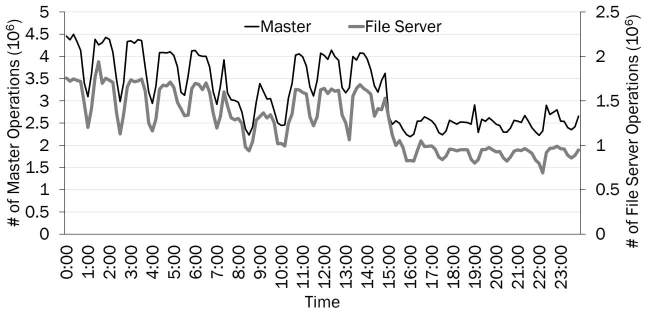

Figure 11. Number of InterFS operations in a real cluster.

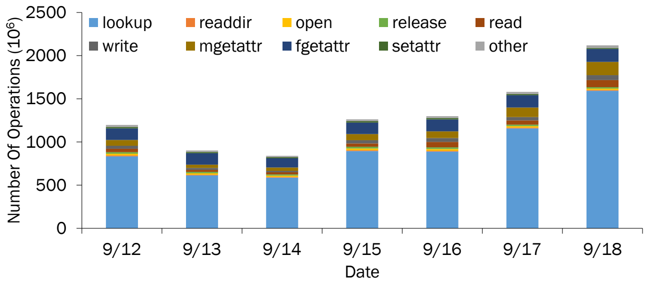

Figure 12. Breakdown of operations.

Resource	Usage
CPU	<10%
Memory	<2 GB
Network bandwidth	<15 MB/s
Disk throughput	<30 MB/s

Table 2. Typical resource usage of InterFS file server.

5. Related Work

Resource Utilization Many researchers have investigated strategies of improving resource usage of data centers. The QoS for latency-critical workloads is studied in [10]. Quasar [6] is a cluster management system to increase resource utilization. Google proposes Omega [16], a scheduler for large computing cluster. However, they mainly deal with the CPU and memory utilization, but the storage capacity has long been ignored.

Distributed File System Distributed file system has long been studied. Examples include Coda [15], Ceph [18], and Panassas [12]. GFS [8] proposed by Google and its successor HDFS [17] are highly optimized for Map-Reduce processing. It does not support random file access and POSIX interface. Ceph [18] presents a distributed file system using a hash-based distribution scheme which does not have a centralized node for improving the scalability, but this also makes its structure very complex and hard to deploy in production environment. Panassas [12] moves the RAID checksum computation process to the clients to reduce the pressure on storage nodes. Lustre [2] uses a distributed lock manager protocol in which clients, OSS, and the metadata manager all participate. Boxwood [11] provides a B-tree implementation, but the design favors strict consistency over scalability, limiting the scale to a few tens of machines. However, none of these systems take the resource sharing issues into account, which is our main design goal.

6. Conclusion

In order to improve storage space utilization of clusters running online services, we propose a distributed file system called InterFS. Compared to other file systems, the major advantage of InterFS is that it can be interplanted on these clusters without interfering with high-priority services. Because of this, only carefully selected applications can be served on InterFS. In real world, we find that there exist a lot of long-tailed applications, which are suitable for leveraging InterFS. Consequently, InterFS can significantly improve the storage space utilization in data centers. Now, InterFS has been widely deployed in Baidu's data centers.

Acknowledgments

We are grateful to the anonymous reviewers and our shepherd, Lakshminarayanan Subramanian, for their valuable feedback and comments. We also appreciate engineers at Baidu, especially Jianqiang Zheng, Yao Wang, Bo Guo, Lianjiang Lin, Miao Yu, for their helpful suggestions. This work was supported by the the National Natural Science Foundation of China (No. 61202072), National High-tech R&D Program of China (No. 2013AA013201), as well as by generous grants from Baidu.

References

[1] GlusterFS. `http://www.gluster.org/`.

[2] Lustre. `http://lustre.org/`.

[3] MooseFS. `http://www.moosefs.org/`.

[4] L. Barroso and U. Hölzle. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-scale Machines*. Synthesis lectures in computer architecture. Morgan & Claypool, 2009.

[5] L. Barroso, J. Clidaras, and U. Hölzle. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines, Second edition*. Synthesis Lectures on Computer Architecture. Morgan & Claypool Publishers, 2013.

[6] C. Delimitrou and C. Kozyrakis. Quasar: Resource-efficient and qos-aware cluster management. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '14, pages 127–144, New York, NY, USA, 2014. ACM.

[7] B. Fitzpatrick. Memcached: a distributed memory object caching system. *Memcached-a Distributed Memory Object Caching System*, 2011.

[8] S. Ghemawat, H. Gobioff, and S.-T. Leung. The google file system. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, SOSP '03, pages 29–43, New York, NY, USA, 2003. ACM.

[9] Google. LevelDB. `https://github.com/google/leveldb`.

[10] J. Leverich and C. Kozyrakis. Reconciling high server utilization and sub-millisecond quality-of-service. In *Proceedings of the Ninth European Conference on Computer Systems*, EuroSys '14, pages 4:1–4:14, New York, NY, USA, 2014. ACM.

[11] J. MacCormick, N. Murphy, M. Najork, C. A. Thekkath, and L. Zhou. Boxwood: Abstractions as the foundation for storage infrastructure. In *OSDI*, volume 4, pages 8–8, 2004.

[12] D. Nagle, D. Serenyi, and A. Matthews. The panasas activescale storage cluster: Delivering scalable high bandwidth storage. In *Proceedings of the 2004 ACM/IEEE conference on Supercomputing*, page 53. IEEE Computer Society, 2004.

[13] J. Ouyang, S. Lin, S. Jiang, Z. Hou, Y. Wang, and Y. Wang. Sdf: Software-defined flash for web-scale internet storage systems. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '14, pages 471–484, New York, NY, USA, 2014. ACM.

[14] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch. Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In *Proceedings of the Third ACM Symposium on Cloud Computing*, SoCC '12, pages 7:1–7:13, New York, NY, USA, 2012. ACM.

[15] M. Satyanarayanan, J. J. Kistler, P. Kumar, M. E. Okasaki, E. H. Siegel, and D. C. Steere. Coda: A highly available file system for a distributed workstation environment. *Computers, IEEE Transactions on*, 39(4):447–459, 1990.

[16] M. Schwarzkopf, A. Konwinski, M. Abd-El-Malek, and J. Wilkes. Omega: Flexible, scalable schedulers for large compute clusters. In *Proceedings of the 8th ACM European Conference on Computer Systems*, EuroSys '13, pages 351–364, New York, NY, USA, 2013. ACM.

[17] K. Shvachko, H. Kuang, S. Radia, and R. Chansler. The hadoop distributed file system. In *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*, pages 1–10, May 2010.

[18] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. E. Long, and C. Maltzahn. Ceph: A scalable, high-performance distributed file system. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*, OSDI '06, pages 307–320, Berkeley, CA, USA, 2006. USENIX Association.