Two-Mode Data Distribution Scheme for Heterogeneous Storage in Data Centers

Wei Xie*, Jiang Zhou*, Mark Reyes*, Jason Noble† and Yong Chen*
*Department of Computer Science, Texas Tech University, Lubbock, TX 79413
†Nimboxx, Inc., B #150, 1825 Kramer Ln, Austin, TX 78758
Email: wei.xie@ttu.edu, jiang.zhou@ttu.edu, mark.reyes@ttu.edu yong.chen@ttu.edu, jason.noble@nimboxx.com

***Abstract*—Fast growing "Big Data" demands present new challenges to the traditional distributed storage system solutions. In order to support cloud-scale data centers, new types of distributed storage systems are emerging. They are designed to scale to thousands of nodes, maintain petabytes of data and be highly reliable. The support for virtual machines is also becoming essential as it is one of the most important technology that supports cloud computing. To meet these needs, these distributed storage systems are implemented with advanced data distribution schemes. Data are striped and distributed across the storage cluster based on distribution algorithms instead of mapping tables. The existing algorithms usually balance the data distribution across nodes proportional to their capacity. However, they overlook distinct performance characteristics across different nodes and devices in the emerging heterogeneous storage environment. We propose a *two-mode data distribution scheme* in this study to maximize the overall performance and keep data balanced across the storage cluster at the same time. The working principle of the two-mode data distribution scheme is provided. We also present a new data read and write strategy to work with the two-mode scheme. We evaluate the computation time for data distribution using two-mode scheme and analyze its implication on the overall IO performance. We expect significant performance improvement while it still needs more analytical and experimental evaluation to further examine the details.**

I. Introduction

The data explosion in the trend of big data applications exposes great challenges to the underlying storage systems. Recently, major Internet Service companies like Google and Yahoo! have built up their cloud-scale data centers to accommodate the massive storage and computing needs of their applications. One of the biggest challenges for cloud-scale data centers is the management of data on a large number of storage nodes. Traditional distributed file systems support a global hierarchical namespace for the files contained in the storage cluster. They require a centralized or distributed meta-data server to handle the mapping between files and storage nodes. The meta-data server could be a performance bottleneck of the storage system. It also limits the system's reliability and scalability.

Virtualization technology has been the foundation of cloud computing. It unifies the data center's computing, network and storage hardwares together and offers high efficiency, low cost, fast application provisioning and high availability. The storage requirement for virtual machines is different from the traditional distributed file systems. Virtual machines use a Virtual Machine Disk Image (VMDI) file to emulate the Virtual Block Device (VBD). IO accesses in guest virtual machine file systems translate to IO operations on the VMDIs, which have completely different patterns [1]. Existing virtual machine software usually implement block-level IO drivers that allow it to hook up with different storage systems instead of just flat files in the file system [2]. This design makes it much easier for virtual machines to utilize large-scale distributed storage systems. Those systems just need to implement a block storage service according to the interface defined by the virtual machine. Several commercialized or active-developing projects are following this path [3]–[5].

In order to run a large number of virtual machines up on the cloud-based data centers, there is a high demand on the availability and scalability [6]. In order to achieve high availability, the storage system should work non-stop even with the failure of some nodes. Data should be appropriately replicated and placed in different nodes to ensure the maximum data availability. The storage system should also scale out well to expand the capability of data centers as the application demand grows. Traditional distributed storage systems use a centralized table to manage data-node distribution [7], [8]. However, as the size of the system grows, the size of the management table becomes enormously large. The sharing and synchronization of such large tables will also be very time-consuming. In addition, scaling up to cloud-scale also involves data migration when new nodes are added or old ones removed. The traditional way of data migration does not scale well to the level that current cloud-scale data centers demand [9], [10].

The object storage [11] and distribution algorithm [12] are usually involved in addressing the scalability and availability challenge. The object storage uses a flat, instead of hierarchical structure used in traditional file systems. This key-value storage architecture fits well with unstructured data and provides features like application programmable interface [13], unified namespace across different nodes, and data management functions like data replication and data distribution at object-level. The algorithm management of data-node distribution eliminates the need of a centralized table for mapping data to nodes. It uses an algorithm, for example Consistent Hashing [14] or pseudo-random number generator [15], to determine the data placement on the storage nodes. The benefit of a distribution algorithm compared to the mapping table includes better performance, high scalability, easy management, and inherent data replication capability. Storage systems using the object storage and algorithm-based distribution are becoming popular, for example Dynamo [12], Cassandra [16], Ceph [4],

Sheepdog [5], SPOCA [17] and ASURA [18].

One of the main problems in the distributed storage system is the distribution of data on nodes. In the consistent hashing algorithm, data are simply distributed randomly and uniformly on each node. This design ensures a balanced data distribution assuming each storage node has the same specification, namely capacity and throughput.

However, recent cloud-scale data centers are adopting heterogeneous storage solutions, which means that both fast solid state drives and slower hard disk drives will be used. As the technology of solid state drive is growing fast, devices with five times better performance are appearing in the market. It may introduce even more diversity in the storage devices in data centers. Assuming the storage nodes having similar specifications will be not a good choice any more.

Heterogeneous storage combines the benefit of cost-effective hard disk drives and fast-but-expensive new types of storage devices like solid-state drives and other storage class memories (SCM). The general goal of designing a heterogeneous storage system involves making the best of use of fast storage while at the same time exploiting the affordable massive capacity of hard disk drives.

Numerous data distribution algorithms were recently proposed to consider the difference of the capacity in different storage nodes. For instance, Ceph, SPOCA and ASURA take the storage capacity into consideration and make the data distribution proportional to it. This strategy solves the problem of data balancing on storage nodes. However, the limitation of this strategy is that the capacity-based load distribution ignores the performance of different storage devices. It is observed that high-performance storage devices, such as solid-state drives, usually have a much smaller capacity than the slower hard disk drives. The capacity-based distribution may under-utilize the faster solid-state drives because of their lower capacity. Additionally, when the IO access is intensive, the high-capacity hard disk drives may be overwhelmed because they are assigned with a large proportion of IO load.

To address this problem, we design a *two-mode data distribution scheme* for heterogeneous distributed storage systems. The data distribution works in two modes: performance mode and capacity mode. In the performance mode, the proportion of data placement is based on the throughput of each storage node. Faster nodes, for example SSD-equipped ones, handle more IO requests in this mode. In contrast, the capacity mode works by assigning proportions of IO loads to each node according to its capacity. This mode allows re-distribution of data so that no node will run out of space earlier than others. The algorithm switches between these two modes based on the data uniformity across the cluster and the user IO intensity. It works in performance mode to allow for maximized performance. Then it switches to capacity mode so that data are migrated across nodes to balance the use of the capacity. Since there is overhead generated by the data migration, the switch between the two modes should be controlled so that minimal data are migrated and migration does not have substantial impact on user IO. The read and write process also requires modification since data placement is not only determined by the algorithm and data ID, but also the mode. We do not store mode information for data, but instead, use a modified read and write policy so that data placement can be retrieved without breaking the simplicity of algorithm-based distribution.

In this study, we present the design of the two-node data distribution algorithm and the analysis of its performance improvement and data distribution compared against the state-of-the-art data distribution algorithms.

The contribution of this study includes:

- We propose an innovative data distribution scheme for distributed storage systems to maximize the storage throughput while maintaining balanced data distribution.
- We design data read and write schemes to adapt existing distribution algorithms to a two-mode scheme.
- We conduct analysis on the data distribution performance and the improvement on overall system throughput.

II. Background

A. Distributed Storage Systems

To manage a large amount of data in enterprise data centers, distributed storage systems are needed. Traditional distributed storage systems are usually composed of application servers, storage nodes, and metadata servers. These components are usually connected with a high-speed network. The application servers run applications and send the data requests to the metadata servers. The metadata servers contain the mapping information between data chunks and storage nodes. When accessing data, the location of data can be retrieved by accessing the metadata servers. Data files are usually split into smaller chunks. These data chunks are distributed across multiple storage nodes. The mapping information contained in the metadata server gives a logical view of the storage system as if all data are stored in a single node. This design hides the complexity of data distribution and management from the applications so that they do not need to be modified in order to run on distributed storage.

Data are usually replicated across storage nodes to allow for better fault-tolerance. When the failure of storage nodes occurs, there will be no data loss due to the replicated data. A recovery scheme is usually implemented to allow fast recovery from storage node failures. When nodes are added or removed due to failure/recovery or reconfiguration by administrators, data need to be rebalanced across the nodes. This means that some existing data need to be migrated among data nodes. This process incurs performance impact to the storage system and may also limit the scalability [19].

There is a new type of distributed storage system, which is derived from P2P network. In this type of distributed storage, there is no metadata server. Instead of retrieving data location from the mapping table in metadata servers, data distribution is consistently and solely determined by a certain algorithm, such as consistent hashing [14], CRUSH [20], and ASURA [18]. Since there is little information needed to be memorized for an algorithm's based distribution (usually the parameters of the algorithm), the storage nodes are symmetric to each other. This design means that each node is aware of the location of all the data in the cluster.

One of the advantages of the algorithm-based distribution is the elimination of metadata server. Metadata server works fine with small to moderate sized distributed storage. However, when the scale of the cluster becomes larger, the size of the metadata grows prohibitively large. The metadata server becomes the bottleneck when maintaining a large amount of data. It uses an enormous amount of memory and storage to contain the metadata. In addition, sharing and synchronizing the metadata compete for the storage and network bandwidth resource in the storage system. On the other hand, the algorithm-based distributed storage does not rely on metadata for data distribution. It allows a much better scalability, for example to scale to thousands of nodes and petabytes of data.

The algorithm-based distributed storage system is highly dependent on the algorithm design to distribute data across the cluster. Since every node in the cluster knows the distribution algorithm, it is aware of the data locations of all the data in the cluster. It means that each storage node can be a data server and a data client at the same time. This design makes the job of adding and removing nodes from the cluster easy, because each node just needs to migrate data according to the algorithm known to it. The design of the distribution algorithm should also consider calculation time, the resource utilization, and distribution uniformity.

Structured data are usually stored in distributed database, like BigTable [16], Dynamo [12] and Apache Cassandra [21]. They provide cloud-scale storage for structured data and also offer high-performance database transactions. Unstructured data are usually stored on distributed file systems, like Hadoop Distributed File System(HDFS)[8], Lustre [7], Ceph [4] and Sheepdog [5]. They provide normal hierarchical file system functions (HDFS, Lustre and Ceph), or block storage interface to be used for virtual machines (Ceph and Sheepdog), or object storage interface (Ceph and Sheepdog).

B. Virtual Machine and Cloud-Scale Storage

Virtualization technology made the cloud-scale data center possible. Virtual machines running in the data center helps to maximize the resource provisioning. The users could have full control of their own virtual machines and configure it the way they want. It has been successful in the business world that most major companies and institutions are using virtualization to meet their computing and storage demands.

The cloud service providers usually build very large scale compute clusters. This cloud-scale storage is provided to the virtual machine, via distributed storage system. Since virtual machines emulate storage devices using Virtual Machine Disk Image (VMDI) files, which are generally unstructured data, providing block storage interface to virtual machine is a widely adopted solution.

Several existing distributed storage systems have been designed to provide block storage for virtual machines. GlusterFS, Ceph and Sheepdog all provide this functionality. For example, Sheepdog provides a block IO interface for QEMU [2] virtual machines. It also supports virtual machine disk image features such as snapshot, copy-on-write, etc. These distributed storage systems usually have P2P architecture and algorithm-based data distribution to allow high scalability and easy and fast node addition/removal management. We target on improving the data distribution algorithm used in these systems because they are becoming important and essential part of the cloud-scale data centers.

C. Heterogeneous Storage and Tiered Storage

The aforementioned existing distributed storage systems determine the portion of data put in each node based on their capacity. It works fine if we assume that each node has the same type of storage device and similar performance specification. However, there exist faster storage devices or systems, such as disk drives formed in RAID and Solid State Drives (SSDs). These devices could offer much higher throughput than the traditional hard disk drives (HDDs). However they could be much more expensive than hard disk drives to prevent them from replacing HDDs. For example, a mainstream SSD could easily offer more than ten times larger sequential read and write performance comparing to a high-end hard disk drive [22]. The random access of SSDs outperforms the hard disk drives even more.

Based on this observation, considering the performance specification in each storage node is necessary to fully utilize the high-performance storage devices in the heterogeneous environment. There has been several studies on tiered storage in the heterogeneous environment [23]. Hystor [10] tried to find the optimized way to place data on SSDs and hard disk drives. They designed algorithms to detect performance critical data blocks based on workload characterization and place those data on SSDs for better performance. Janus [9] creates two storage tiers for SSDs and HDDs, respectively. Data are placed on SSDs first and moved to HDD tier later based on First-In-First-Out or Least-Recently-Used policy. They optimize the storage system for maximized data read from SSDs, based on the workload trace collected from the distributed storage. These solutions optimize the performance of heterogeneous storage, but none of them could fit in the algorithm-based distributed storage system like Sheepdog and Ceph. Compared to these studies, what we propose is a universal solution to the data distribution problem in algorithm-based distribution in order to satisfy both the performance and data balance requirements. We do not target on optimal data placement based on data access information. Instead, we *focus on offering a simple way to offer maximized throughput while not breaking the simplicity, scalability, and reliability of algorithm-based data distribution at the same time.*

III. Design of Two-Mode Data Distribution Scheme

In this section we describe the design of the two-mode data distribution algorithm. Our algorithm can be applied to any algorithm-based data distribution strategies. In this paper, we select the ASURA algorithm to demonstrate the working principle of our scheme.

A. Design of Two-Mode Data Distribution Scheme

We present the overall design of the two-mode data distribution scheme. As seen in Figure 1, there are three main components in this scheme: distributor selector, performance-based distributor, and capacity-based distributor. The distributor selector monitors the workload intensity and remaining

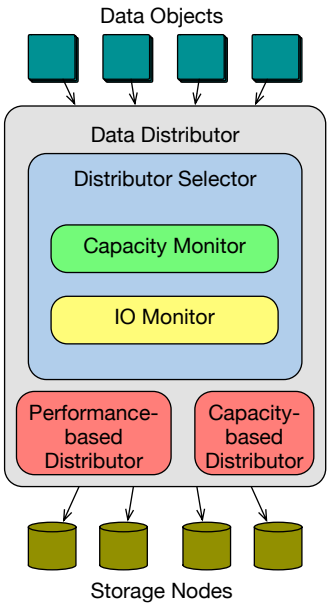

Fig. 1. Components of two-mode data distrbution scheme

capacity of nodes in the cluster. The workload intensity is monitored so that performance-mode is only turned on when it predicts that there is significant IO in the near future. It also keeps track of the remaining capacity of the storage nodes since the use of performance mode will eventually cause data imbalance across nodes, in which case capacity mode should be forced to re-balance data distribution.

The system starts with capacity mode using the capacity-based distributor. Once the monitor detects a significant IO user input, the performance mode kicks in. User write requests can benefit from the high throughput and low latency of the faster storage devices such as solid-state drives. Subsequent read requests to the written data can also get maximized overall performance. However, keeping the system in performance mode will result in unbalanced data placement on the nodes: some nodes with smaller capacity may contain more data than those with larger capacity. When the monitor identifies a period that there is no significant user throughput, it switches to use capacity-based distributor. After switching the mode, not only new write data will be placed based on the capacity-based distributor, but also some existing data will be migrated because the distribution policy has changed. Most of the data distribution algorithms, such as CRUSH, Consistent Hash and ASURA, ensures that minimal data are migrated. The system should stay in capacity-mode for load-balance until the monitor predicts that there will be significant user throughput coming.

For systems that rarely having idle period, the system could keep working in performance mode since the monitor is not able to find any low IO period. In this case, the utilization of the capacity of each node should be monitored. If the degree of imbalance across the cluster exceeds a certain threshold, the capacity mode must be forced until the cluster reaches a certain degree of balance.

Since there are two different rules of placing data in this scheme, data could be missing if only using the distributor in use. For example, a data item could be written in capacity mode but read in performance mode. Thus the read and write operations are handled differently in two-mode data distribution scheme, compared to the existing single-capacity-mode algorithms. The requested data objects will find its locations based on the calculated value by the distribution algorithm. However, the distributor does not always give the correct locations in two-mode scheme. This is because it could be written with a different distributor and such placed on other sets of nodes. In order to solve the problem, we design a new read and write scheme for our two-mode scheme, which is described in Section III-B.

B. Read and Write Operation in the Two-Mode Data Distribution Scheme

Since there are two modes of data distribution, there could be an inconsistency issue for read and write: the nodes determined by current chosen distributor may not be the correct ones that the data object is located. In order to solve the inconsistency problem, we design a naïve read and write scheme as shown in Algorithm 1 .

The read operation first calculates the data locations using the distributor that is currently being used. If data can not be fetched from any of the nodes calculated, it tries to find the data at the nodes determined by the other distributor. Since the data object must be distributed by one of the distributors, it is guaranteed to be found eventually. It is noticed that data objects are usually replicated and there is very likely overlap between the two set of destined nodes calculated by two mode distributors. This overlap means that there will be one or more nodes determined by the current distributor to be correct. Because reading only requires a single data source, it is adequate to complete the read request.

For the write operation, the destination node is simply calculated by the current distributor. However, this could leave dirty data if the write operation overwrites existing data that reside on different nodes. Like the read request, it first utilizes the current distributor to find the destination nodes. Then it follows the procedure described in Algorithm 1. It ensures that

Algorithm 1 Naïve Data Object Write Scheme

INPUT: Current mode M_c and the other mode M_o, data object ID to write oid ;

1: get the nodes $\{nd_1, nd_2, ...nd_r\}$ oid is placed on based on the distributor in current mode M_c;
2: search oid in $\{nd_1, nd_2, ...nd_r\}$;
3: **if** oid does not exist on more than one of the nodes in $\{nd_1, nd_2, ...nd_r\}$ **then**
4: find oid in nodes $\{ond_1, ond2, ..., ond_r\}$ generated by the distributor in mode M_o;
5: **if** oid does not exist on more than one of the nodes in $\{ond_1, ond_2, ...ond_r\}$ **then**
6: this is new data;
7: **else**
8: invalidate oid in $\{ond_1, ond_2, ...ond_r\}$;
9: **end if**
10: oid is placed on $\{nd_1, nd_2, ...nd_r\}$;
11: **end if**

the written data are placed on the nodes calculated by current distributor mode and the old data, no matter located on what nodes, are invalidated or deleted.

IV. Analysis of the Two-Mode Data Distribution Scheme

This section analyzes the calculation time, overall performance of our two-mode distribution scheme. Since our scheme is based on existing data distribution algorithms like consistent hashing, CRUSH or ASURA, we evaluate the impact of our two-mode scheme based on these algorithms.

A. Calculation Time

We first review the existing data distribution algorithms that distribute data only based on node capacity.

Consistent hashing involves generating hash values for all data nodes and distributing data by calculating the hash values of data objects. The first stage is to generate hash values for each node. Since Virtual Nodes are used for each node to allow multiple entries to exist on the hash ring, multiple hash values need to be generated for each node. Assuming the number of node is N and the number of Virtual Nodes is V, the time complexity of calculating all the hash values will be $O(NV)$. Then all the hash values need to be sorted for later lookup, it takes at least $O(NV \times log(NV))$. To add these together, the total time complexity of the initial stage is $O(NV \times log(NV))$. In the second stage, when the hash value of a data object is calculated, it needs to be searched in the sorted hash values of virtual nodes. Calculating the hash value takes $O(1)$ time; while searching for it takes at least $O(log(NV))$.

The CRUSH algorithm calculates the hash value for each data object and node pair. It selects the node with the highest hash value among all nodes to place the data object. For each data object, it needs to search for the highest value among N nodes. Then the time complexity of it is $O(N)$.

For ASURA, it has been proven that it takes constant time for calculating the data distribution [18]. The initial stage involves assigning segments for each data node, which takes $O(N)$ to calculate. This is only calculated once when the cluster is configured so it is a reasonable time.

The two-mode data distribution scheme differs with the original aforementioned algorithms by using two sets of distribution scheme at the same time. In the initial stage, it means double calculation time. For consistent hashing for example, it generates two different hash rings, each with different virtual node numbers of data nodes according to capacity and performance, respectively. For the other algorithms, it is similar where the only difference is using hash ring, straw bucket or segment.

In the data distribution stage, the value calculated by data object is searched based on the distributor (hash ring, straw bucket or segments) in the current mode. It only needs to search the other mode's distributor in cases that data can not be found in all nodes for read operation, or not found in more than one nodes for write operation. Assuming the total number of objects is n, there are $\frac{n}{N}$ data objects to be searched in each node. Searching for a data object would require $O(log(n/N))$ time.

For consistent hashing, its data distribution calculation time is $O(log(NV))$. Since both of the data distribution time and local object search time are logarithmic, these two parts of computation should be comparable.

For CRUSH algorithm, as it has linear time complexity for data distribution time. It is only suitable for smaller scale storage clusters otherwise the data distribution time could be unacceptably large. We generally do not consider the local data search time will be the bottleneck for CRUSH algorithm.

For ASURA, according to the values gained from [18], it is distribution time is around $1\mu s$. But the local object search time is more than 10 time smaller than ASURA's distribution time with number of data objects in each node ranging from 1 to 1,000,000. This means that the failing to get the correct nodes based on current mode will not introduce significant overhead to ASURA's total data distribution time.

The probability of needing two distributors for locating data depends on the switch frequency of the two modes. If the frequency is controlled to be low, the extra overhead introduced by two-mode distribution could be minimized. We are planning to include a policy to control the switch frequency in the future.

B. Performance Improvement and Migration Overhead

The goal of our two-mode data distribution scheme is to allow maximized performance while at the same time maintaining balanced data distribution based on capacity. If ignoring the overhead of migrating data when switching between modes, the improvement of using two-mode scheme is straightforward. When user throughput is high, the performance distributor will guarantee maximum system throughput. We assume that the capacity of the data nodes is $c_1, c_2, ..., c_n$, respectively. The maximum throughput of each node is $t_1, t_2, ..., t_n$. The total user request throughput is R. Then the ideal total throughput in performance mode is:

$$T_{perf} = \sum_{i=1}^{n} min\{\frac{R \times t_i}{\sum_{j=1}^{n} t_j}, t_i\} \quad (1)$$

However, the total throughput of the system in capacity mode is:

$$T_{cap} = \sum_{i=1}^{n} min\{\frac{R \times c_i}{\sum_{j=1}^{n} c_j}, t_i\} \quad (2)$$

Under low user input R that does not saturate the throughput of any nodes, the capacity mode performs the same as the idea performance mode. But when the user input R continues to grow, the performance mode starts to outperform the capacity mode. However, considering the migration overhead, the performance mode may not as good as Equation 1 shows.

If considering the data migration overhead, the design of the monitor component is important. The monitor should detect different workload patterns and adaptively switch the mode in order to minimize data migration. For example, the monitor need to stick to the capacity mode if the workload is always write-intensive, because allowing the performance mode may introduce too much mode-switch overhead that outweigh the benefit of higher throughput. However, the two-mode scheme will fit the workloads with moderate IO intensity but short bursty period. The monitor will naturally choose performance

mode in the bursty period while switch to capacity mode when it finishes. In this case, the migration overhead will not affect the overall performance too much in the moderate IO period.

V. Conclusion and Future Work

In this paper, we propose a new two-mode data distribution scheme to improve the existing algorithm-based distribution strategies in heterogeneous environments. We present our design details including the incorporation of performance mode, the read and write scheme in two modes, and the transition between the two modes. The distribution calculation time is also analyzed for different underlying algorithms. We evaluate that incorporating two-mode does not add significant overhead to the calculation time for the three algorithms evaluated. It provides throughput improvement to existing heterogeneous storage system. The two-mode scheme is simple yet universal to be implemented with existing data distribution algorithms without breaking their advantage in scalability.

There are still two limitations of our scheme though. The first limitation is that we do not consider data access patterns in our scheme. The migration of data among nodes is simply based on the value generated by algorithms. It does not consider data hotness and intelligently place data to maximize the performance benefits like in [9], [10]. We plan to incorporate a data hotness detection component so that frequently accessed data will stay on faster nodes so that future read access could benefit from their features of high bandwidth and low latency. The second limitation is that we still haven't implemented the scheme on real system. We are planning to implement it on Sheepdog storage system and conduct more experiments to evaluate how our scheme works in real situations.

Acknowledgment

This research is supported by the National Science Foundation under grant IIP-1362134 (through the Nimboxx membership contribution) and CNS-1338078.

References

[1] S. Ma, H. Chen, H. Lu, B. Wei, and P. He, "MOBBS: A Multi-tiered Block Storage System for Virtual Machines Using Object-Based Storage," in *High Performance Computing and Communications, 2014 IEEE 6th Intl Symp on Cyberspace Safety and Security, 2014 IEEE 11th Intl Conf on Embedded Software and Syst (HPCC,CSS,ICESS), 2014 IEEE Intl Conf on*, Aug 2014, pp. 272–275.

[2] F. Bellard, "QEMU, a Fast and Portable Dynamic Translator," in *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, ser. ATEC '05. Berkeley, CA, USA: USENIX Association, 2005, pp. 41–41.

[3] Amr Abdelrazik, Greg Bunce, Kathy Cacciatore, Kenneth Hui, Sridhar Mahankali and Frans Van Rooyen, "Adding Speed and Agility to Virtualized Infrastructure with OpenStack." [Online]. Available: http://www.openstack.org/assets/pdf-downloads/virtualization-Integration-whitepaper-2015.pdf

[4] 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*, ser. OSDI '06. Berkeley, CA, USA: USENIX Association, 2006, pp. 307–320.

[5] "Sheepdog project," https://sheepdog.github.io/sheepdog/, accessed: 2015-07-08.

[6] D. A. Maltz, "Challenges in Cloud Scale Data Centers," in *Proceedings of the ACM SIGMETRICS/International Conference on Measurement and Modeling of Computer Systems*, ser. SIGMETRICS '13. New York, NY, USA: ACM, 2013, pp. 3–4.

[7] F. Wang, S. Oral, G. Shipman, O. Drokin, T. Wang, and I. Huang, "Understanding lustre filesystem internals," 2009.

[8] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop Distributed File System," in *Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, ser. MSST '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 1–10.

[9] C. Albrecht, A. Merchant, M. Stokely, M. Waliji, F. Labelle, N. Coehlo, X. Shi, and C. E. Schrock, "Janus: Optimal Flash Provisioning for Cloud Storage Workloads," in *Proceedings of the 2013 USENIX Conference on Annual Technical Conference*, ser. USENIX ATC'13. Berkeley, CA, USA: USENIX Association, 2013, pp. 91–102.

[10] F. Chen, D. A. Koufaty, and X. Zhang, "Hystor: Making the Best Use of Solid State Drives in High Performance Storage Systems," in *Proceedings of the International Conference on Supercomputing*, ser. ICS '11. New York, NY, USA: ACM, 2011, pp. 22–32.

[11] A. Azagury, V. Dreizin, M. Factor, E. Henis, D. Naor, N. Rinetzky, O. Rodeh, J. Satran, A. Tavory, and L. Yerushalmi, "Towards an object store," in *Mass Storage Systems and Technologies, 2003. (MSST 2003). Proceedings. 20th IEEE/11th NASA Goddard Conference on*, April 2003, pp. 165–176.

[12] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: Amazon's Highly Available Key-value Store," *SIGOPS Oper. Syst. Rev.*, vol. 41, no. 6, pp. 205–220, Oct. 2007.

[13] J. Baron and S. Kotecha, "Storage Options in the AWS Cloud."

[14] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, and D. Lewin, "Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web," in *Proceedings of the Twenty-ninth Annual ACM Symposium on Theory of Computing*, ser. STOC '97. New York, NY, USA: ACM, 1997, pp. 654–663.

[15] M. Matsumoto and T. Nishimura, "Mersenne Twister: A 623-dimensionally Equidistributed Uniform Pseudo-random Number Generator," *ACM Trans. Model. Comput. Simul.*, vol. 8, no. 1, pp. 3–30, Jan. 1998.

[16] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: A Distributed Storage System for Structured Data," *ACM Trans. Comput. Syst.*, vol. 26, no. 2, pp. 4:1–4:26, Jun. 2008.

[17] A. Chawla, B. Reed, K. Juhnke, and G. Syed, "Semantics of Caching with SPOCA: A Stateless, Proportional, Optimally-consistent Addressing Algorithm," in *Proceedings of the 2011 USENIX Conference on USENIX Annual Technical Conference*, ser. USENIXATC'11. Berkeley, CA, USA: USENIX Association, 2011, pp. 33–33.

[18] K.-i. Ishikawa, "ASURA: Scalable and Uniform Data Distribution Algorithm for Storage Clusters," *ArXiv e-prints*. [Online]. Available: http://adsabs.harvard.edu/abs/2013arXiv1309.7720I

[19] A. Gulati, C. Kumar, I. Ahmad, and K. Kumar, "BASIL: Automated IO Load Balancing Across Storage Devices," in *Proceedings of the 8th USENIX Conference on File and Storage Technologies*, ser. FAST'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 13–13.

[20] S. A. Weil, S. A. Brandt, E. L. Miller, and C. Maltzahn, "CRUSH: Controlled, Scalable, Decentralized Placement of Replicated Data," in *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*, ser. SC '06. New York, NY, USA: ACM, 2006.

[21] A. Lakshman and P. Malik, "Cassandra: A Decentralized Structured Storage System," *SIGOPS Oper. Syst. Rev.*, vol. 44, no. 2, pp. 35–40, Apr. 2010.

[22] F. Chen, R. Lee, and X. Zhang, "Essential Roles of Exploiting Internal Parallelism of Flash Memory based Solid State Drives in High-speed Data Processing," in *High Performance Computer Architecture (HPCA), 2011 IEEE 17th International Symposium on*. IEEE, 2011, pp. 266–277.

[23] J. Chen, J. Liu, P. Roth, and Y. Chen, "Using Working Set Reorganization to Manage Storage Systems with Hard and Solid State Disks," in *The 7th International Workshop on Parallel Programming Models and Systems Software for High-End Computing (P2S2), in conjunction with The 43rd International Conference on Parallel Processing (ICPP)*, 2014.