2017 IEEE 23rd International Conference on Parallel and Distributed Systems

Kinetic Action: Performance Analysis of Integrated Key-Value Storage Devices vs. LevelDB Servers

Manas Minglani, Jim Diehl, Xiang Cao*, Bingzhe Li, Dongchul Park†, David J. Lilja, and David H.C. Du
University of Minnesota - Twin Cities, Minneapolis, Minnesota, USA
{mingl001, jdiehl, lixx1743, lilja, du}@umn.edu
*School of Computing and Information Systems, Grand Valley State University, Allendale, Michigan, USA
caox@gvsu.edu
†Computer & Electronic Systems Engineering, Hankuk University of Foreign Studies, South Korea
dpark@hufs.ac.kr

***Abstract*—With the rise of cloud storage and many data intensive applications, there is an unprecedented growth in the volume of unstructured data. In response, key-value object storage is becoming more popular for the ease with which it can store, manage, and retrieve large amounts of this data. Seagate recently launched Kinetic direct-access-over-Ethernet hard drives which incorporate a LevelDB key-value store inside each drive. In this work, we evaluate these drives using micro as well as macro benchmarks to help understand the performance limits, trade-offs, and implications of replacing traditional hard drives with Kinetic drives in data centers and high performance systems. We perform in-depth throughput and latency benchmarking of these Kinetic drives (each acting as a tiny independent server) from a client machine connected to them via Ethernet. We compare these results to a SATA-based and a faster SAS-based traditional server running LevelDB. Our sample Kinetic drives are CPU-bound, but they still average sequential write throughput of 63 MB/sec and sequential read throughput of 78 MB/sec for 1 MB value sizes. They also demonstrate unique Kinetic features including direct disk-to-disk data transfer. Our macro benchmarking using the Yahoo Cloud Serving Benchmark (YCSB) shows that mid-range LevelDB servers outperform the Kinetic drives for several workloads; however, this is not always the case. For larger value sizes, even these first generation sample Kinetic drives outperform a full server for several different workloads.**

***Keywords*— Performance Evaluation, Data Center Storage Architecture, Key-Value Store, Cloud Applications**

I. Introduction

The amount of digital data is growing at an extremely rapid pace, and it is estimated that its volume will grow at 40%-50% per year [1]. Most of this data explosion is due to unstructured data. According to predictions from International Data Corporation (IDC), 80% of the 133 exabytes of global data growth in 2017 will be unstructured [2]. Managing such an enormous amount of data is a challenging task.

Most of the data in existing systems is stored and accessed using traditional file-based or block-based systems [3]. Unfortunately, these traditional systems are becoming inefficient as file-based access and hardware requirements limit their scalability. Therefore, there is a need for a data access method that is flexible and capable of horizontal scale-out [4].

Object storage overcomes limitations of file-based systems by offering scalability and being software defined [5]. The data is communicated as objects rather than files or blocks, and additional metadata can be stored alongside the object [3]. Object storage is flat structured and prevents higher-level applications from needing to manipulate data at the lowest level. Therefore, object storage has the flexibility to scale-out horizontally. When a unique identifier, called a "key," is used to access the object, or "value," this form of storage can be called a key-value store (KV store).

There are several key-value stores being deployed to support large websites such as Dynamo at Amazon [6], Redis at GitHub [7], and RocksDB at Facebook [8]. All these systems store ordered <key, value> pairs. Even though key-value stores address the problem of scaling and managing huge amounts of data for the above systems, the existing key-value stores also have several limitations. They run on top of multiple layers of legacy software and hardware, such as POSIX, RAID controllers, etc., designed for file-based systems [5]. Also, these huge systems consume significant amounts of power and rack space [9].

To help overcome these hardware and software limitations, Seagate has announced a new class of hard drives called Kinetic drives [5]. These drives have a built-in processor that runs a LevelDB-based key-value store directly on the drive [10]. Rather than the typical SATA or SAS interface, Kinetic drives communicate externally via TCP/IP over Ethernet. Each drive acts as a tiny server in itself. An important function of the drives is direct P2P (Peer-to-Peer) transfer that allows direct data transfer from one drive to another via Ethernet without the need to copy data through a storage controller or other server [11]. By replacing hardware and software layers, Kinetic drives can reduce the cost and complexity of a large-scale object storage system.

In this work, we seek to better understand the key functionalities, features, and performance of the drives. We use this information to compare Kinetic drives with other LevelDB-based servers and to derive insights about the possibility of replacing traditional hard drives with Kinetic drives. Furthermore, the specification sheets [12] do not provide a detailed analysis of the throughput, latency, and other features, especially in comparison to the other LevelDB-based servers.

We also study the ease of programmability of the Kinetic drives and share our experiences. To the best of our knowledge there are no prior works which evaluate Kinetic drives this thoroughly. To understand several salient features including P2P transfer, "Get," "Put," and others, we develop several tests which we use to identify bottlenecks and limitations of the drives.

Overall, these tests help by giving us experience using this latest class of drives from Seagate as well as help us determine throughput bounds and performance characteristics in different scenarios. Beyond academic interest, we believe the information in this paper may be useful to system-level engineers at data centers and for High Performance Computing or Big Data and Analytics systems.

Our contributions are the following:

- Conduct tests on Kinetic drives to understand the throughput and performance bottlenecks, limits, trade-offs, and characteristics
- Compare key-value store hardware (Kinetic drives) with servers running the LevelDB key-value store on conventional hard drives using micro and macro benchmarks (YCSB)
- Share the experience of using the Kinetic platform

The rest of the paper describes the configuration of the systems, performance results, and derived insights.

II. Kinetic Platform — Motivation and Architecture

Kinetic drives are the latest class of hard drives from Seagate and have a similar form factor to a conventional 3.5" drive. However, Kinetic drives are fundamentally different from traditional drives in two ways: Ethernet connectivity and an internal processor that converts the storage drive to a key-value store [13].

Each drive has two Ethernet ports (1 Gbps each) to transfer data to the clients or other drives using TCP/IP over Ethernet. The ports are for fault tolerance, and only one should be used at a time. Newer models (2nd generation) are expected to have two 2.5 Gbps ports. However, after inquiring, these newer models are not available for the general public, so we cannot test them. Furthermore, Kinetic drives do not use the SCSI interface and instead have a key-value interface. Applications interact with the drives directly using this interface and a standard Ethernet connection. IP addresses are assigned to the drives by a DHCP server [13]. The use of Ethernet allows the storage system to be expanded without the need to install a separate Storage Area Network. Figure 1 is a simplified comparison of the hardware and software stack of a traditional storage system with that of a Kinetic system.

The second major benefit of using these drives is that they encapsulate the concept of in-storage processing by running a LevelDB-based key-value store in the drives. There is a single-core System-on-Chip (SoC) processor inside the drives that runs a custom LevelDB and manages the data on the disk itself. This can reduce the software I/O complexity for applications and simplify the storage rack by eliminating the need for separate storage controllers [9].

Seagate provides a software platform to help write programs for these drives. This Kinetic API comes with libraries for several popular programming languages, a simulator, installation instructions, some test utilities, and some basic test code [5]. The simulator allows the test and custom programs to run without the actual drives (but we run all our experiments on actual drives). The applications or programs are created using the functions and protocols defined in the API library. The protocols are responsible for the actual connection and communication with the Kinetic drives via messages (Figure 2). The Linux Foundation now maintains this Kinetic API [5].

The drives have a built-in capability to support multiple threads and sessions. While the Kinetic API abstracts away most of the internal details of the drives, there are set limits to certain parameters. These limits are listed in Table I. Table II lists the main API commands used to interact with the key-value store on the drive. In addition, the API allows the programmer to perform write synchronization in several different modes as described in Table III. These modes allow the programmer to force immediate synchronization or allow the drive to control the write operations.

Moreover, the Kinetic drives come with unique features, such as P2P transfer, which set these drives apart from regular hard drives. With P2P transfer, one drive can directly transfer its data to another drive via Ethernet; hence, it could be utilized for direct data replication. The process initiates when a client issues a "P2P Put" or "P2P Get" command to a particular Kinetic drive. This drive takes control of the process and automatically (without the client's involvement) performs "Put" (write) or "Get" (read) operations to/from the target drive using Ethernet. After this process completes, both the drives have identical copies of those key-value pairs. These instructions avoid several stack layers typically required to

TABLE I
Kinetic Drive Limits

Device Feature	Limit
max Key Size	4096 Bytes
max Value Size	1 MB
max Version Size	2048
max Tag Size	128
max Connections	100
max Outstanding Read Requests	30
max Outstanding Write Requests	20
max Message Size	1 MB
max Key Range Count	200
max Identity Count	1000
max Pin Size	200

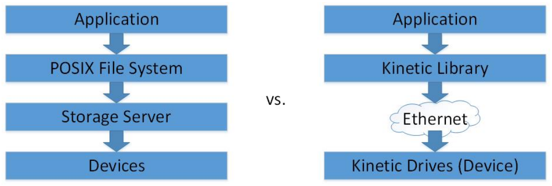

Fig. 1. Software stack of server with traditional drives vs. Kinetic drives.

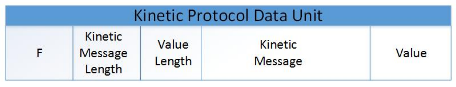

Fig. 2. Message protocol for Kinetic drive.

transfer data and can drastically reduce overall system I/O.

In the following sections, we describe how we use this Kinetic API to evaluate the performance trade-offs, limits, and behavior of these sample drives.

III. Configuration

In this section, we cover the three system configurations we employ to test and evaluate the Kinetic drives and compare them with LevelDB-based servers.

To benefit from the commands described in Table II, we have to leverage the Kinetic API. This API is installed in the client and is used to send commands to different Kinetic drives, which logically act as tiny independent servers as shown in Figure 3. The Kinetic API is designed for several programming platforms including C, C++, Java, and Python. Our testing primarily uses the C library of the API.

The client machine is a Dell R420 running Ubuntu Server 14.04 LTS 64-bit (kernel 3.13) on two quad-core Intel Xeon E5-2407 CPUs @ 2.20 GHz and 12 GB of RAM. This machine uses a 10 Gbps Ethernet PCIe network card to connect to a Kinetic disk enclosure containing four Kinetic sample drives. Internally, this enclosure contains redundant Ethernet switches with 1 Gbps ports for the Kinetic drives and a 10 Gbps uplink port. The Kinetic enclosure also has a separate Ethernet connection to allow access to a web interface that allows basic management of each drive (power cycle the drive, see its IP address, etc). The drives are model ST4000NK001, and we upgraded the firmware to version 03.00.04 using one of the included utility programs. These disks are 4 TB, 5900 RPM, and have 64 MB of disk cache plus another 512 MB of on-board RAM to run the modified versions of Linux and LevelDB on the added Marvell 370 SoC [12].

To compare the Kinetic drives with LevelDB-based servers, we set up two other systems: Server-SATA and Server-SAS. Server-SATA has the same specifications as the Kinetic client system described above plus has LevelDB installed and two additional traditional SATA drives, separate from the OS drive, connected directly to the motherboard via SATA. These two extra SATA drives, Seagate model ST4000VN000, are chosen because they are similar to the Kinetic drives, i.e., 4 TB, 5900 RPM, and 64 MB of disk cache.

Server-SAS also has LevelDB installed but is a more powerful and modern system. This machine runs Ubuntu Server 14.04 LTS 64-bit (kernel 4.1) on two six-core Intel Xeon E5-2620 v3 CPUs @ 2.40GHz and 64 GB of RAM. There is a PCIe LSI MegaRAID SAS-3 3008 adapter connected to an external Dell MD1400 disk enclosure containing 11 Seagate model ST6000NM0034 Enterprise Capacity SAS drives. Each drive is 6 TB, 7200 RPM, and has 128 MB of disk cache.

For both Server-SATA and Server-SAS, LevelDB is installed and db_bench.cc (or a slightly modified version) is run on the various drive configurations from the same machine, i.e., the client benchmark is run against a local server. For tests using two or more drives, Linux software RAID-0 is configured with default *mdadm* settings offered by the GNOME Disk Utility. The target location is then formatted with default ext4 settings

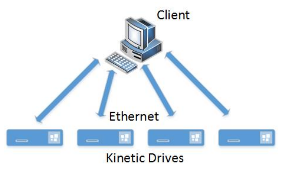

Fig. 3. Logical view with Kinetic drives as independent servers.

and mounted for use. A detailed description of how Server-SATA and the Kinetic client machine are used for YCSB testing is included in Section VIII.

IV. Methodology and Results for Kinetic Drives

This section details a variety of our tests executed on Kinetic drives. Performance results comparing Kinetic drives with LevelDB-based servers are presented in later sections. Our attempt is to first inform the readers about the raw performance metrics of Kinetic drives. This will help system engineers in designing their systems. We present the results of various random and sequential read and write tests for different value sizes as well as our experience with the Kinetic drive's unique P2P commands.

A. *Definitions*

Before delving into the tests, we define important terms which are used throughout the paper.

Sequential Key Order Writes (Sequential Order Writes): Write requests are issued by the client in increasing key ID order, i.e., from key number 1 to key number n.

Random Key Order Writes (Random Order Writes): The client issues write requests in random key ID order.

WriteBack-Flush: This write operation issues $n-1$ commands in the asynchronous WriteBack mode and then the n^{th} command is written in Flush mode to make all the commands persistent. This series of repeated asynchronous WriteBack writes followed by a single Flush command is what we call WriteBack-Flush mode.

B. *Comparison of WriteThrough vs. Flush Mode*

There are many applications, e.g., banking, that cannot risk data loss or inconsistency and must immediately have data stored persistently. Therefore, the Kinetic drive's multiple write modes, depending on the application, can be quite useful. There are two different modes for forcing synchronous write operations, WriteThrough and Flush, as mentioned in Table III. Of the two, WriteThrough operations tend to be faster than pure Flush operations. Flush mode must make sure all previously written commands in the asynchronous WriteBack mode are made permanent. However, requests made in WriteThrough mode are made permanent without checking the status of any other command.

Results of a test to illustrate this difference are shown in Table IV. In this test there are two variations, one where all writes are issued in WriteThrough mode and one where all writes are issued in Flush mode. In both cases, each write (or put) is made persistent before returning a completion signal

TABLE II
COMMANDS AND FUNCTIONS OF KINETIC DRIVES

Command	Function
put	Writes the specified key-value (KV) pair to the persistent store
get	Reads a KV pair
delete	Deletes the KV pair
getNext	Gets the next key that is after the specified key in the sequence
getPrevious	Gets the previous entry associated with a specified key in the sequence
getKeyRange	Gets a list of keys in the sequence based on the given key range
getMetadata	Get an entry's metadata for a specified key
secureerase	Securely erases all the user data, configurations, and setup information on the drive
instanterase	Erase all data in database for the drive; this is faster than secureerase
setSecurity	Set the access control list for the Kinetic drive
P2P Get	Kinetic drive reads directly from another Kinetic drive
P2P Put	Kinetic drive writes directly to another Kinetic drive

TABLE III
DIFFERENT WRITE MODES OF KINETIC DRIVES

Kinetic Write Mode	Description
WriteThrough	The request is made persistent before returning; this does not affect any other pending operations
WriteBack	The requests are made persistent when the drive chooses or when a FLUSH is sent to the drive
Flush	All the write requests that are pending and are not written yet will be made persistent to the disk

to the application. However, for Flush mode, the drives must ensure that there are no pending write requests waiting to be made persistent. For 10,000 key-value pairs and 100,000 key-value pairs, WriteThrough gives higher throughput than repeated Flush mode writes. This illustrates that WriteThrough should be used for synchronous writes, and it is bad programming to repeatedly use Flush mode for multiple writes.

Since throughput depends on the write modes, value size, and number of key-value pairs, we obtain higher throughputs in the following tests than we do in Table IV.

C. *Comparison of Throughputs for Write and Read Operation*

To help programmers understand the basic performance of these drives for various generic workloads, we test read and write performance for KV pairs with different value sizes in either random or sequential key order. In our terminology, Sequential (or Random) Order Reads/Writes mean Sequential (or Random) Key Order Reads/Writes.

Sequential Key Order Writes: This test consists of two experiments. First, after Sequential Order Write (explained in Section IV-A), the data is read back from key number 1 to key number n sequentially for Sequential Order Read. Second, after performing Sequential Order Write, the data is read back in random key ID order for Random Order Read.

Figure 4 shows the throughput obtained from the Kinetic drive for Random Order Read and Sequential Order Read of up to 1 TB of data sequentially written by key order. Since the write process is the same for both the experiments, it is shown only once in the figure. We vary the size of the value from 120 bytes to 1 MB but keep the key size constant at 16 bytes. We also keep the number of key-value pairs constant at 1,000,000. Thus, the x-axis value-size multiplied by one million will give

TABLE IV
COMPARISON OF THROUGHPUTS FOR DIFFERENT WRITE MODES

Value Size	Key-value pairs	Flush	WriteThrough
1 MB	10,000	5.41 (MB/sec)	8.99 (MB/sec)
1 MB	100,000	4.8 (MB/sec)	8.91 (MB/sec)

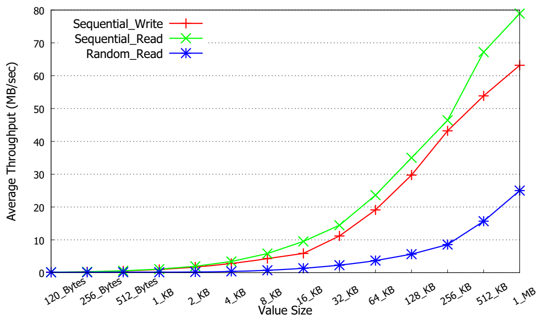

Fig. 4. Average throughputs of Sequential Order Write followed by Random Order Read or Sequential Order Read for 1,000,000 key-value pairs with varied value size and 16 byte key size.

the amount of data transferred for that particular portion of the experiment (i.e., 120 MB for the leftmost points and 1 TB to generate the points on the right).

We observe that the throughput increases with the value size. To understand this trend, we have to understand the capability of the Kinetic drive's internal processor, the client's processing capability, and throughput computation. The client sends requests to the Kinetic drive as fast as possible and can easily saturate the drive with messages. Due to the Kinetic drive's limited internal processing capability and the overhead to handle each request, it can process only a certain maximum number of read or write requests per second. When small value sizes are used, overhead dominates the Kinetic CPU and results in low throughput. As the value size is increased, the ratio of wasted overhead to data sent is improved and so is throughput. As is expected for hard disk technology, we also observe that Random Order Read throughput is lower than Sequential Order Read, because the drive's head has to be repositioned on the platter more often for Random Order Reads than Sequential Order Reads.

To save space, results for Random Order Reads after Random Order Writes on Kinetic drives are incorporated into the LevelDB server comparison in Figure 9.

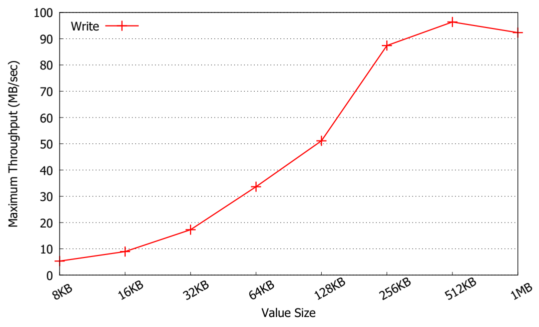

Fig. 5. Maximum write throughput for various value sizes.

D. Maximum Write Throughput for Various Value Sizes

While the previous subsection measured average throughput, application designers may often require the maximum performance that can be extracted from these drives. Based on the peak performance of the individual drives, the system engineers can estimate and try to optimize the raw performance that an entire system will be able to deliver. In this test we find the maximum write throughput that can be obtained for different value sizes with each key size equal to 16 bytes. Each data point on the x-axis in Figure 5 is a different experiment. For each experiment, we vary the number of key-value pairs and perform Sequential Order Write as explained in Section IV-A. The number of key-value pairs required to achieve the maximum throughput is different for all the experiments. Maximum write throughput increases with value size as shown in Figure 5. This trend is again because the CPU-bound (as demonstrated in Section IV-F) Kinetic drive wastes fewer resources on overhead with larger value sizes.

E. Impact of Key Size on Throughput Performance

So far we have varied the value size of the key-value pairs, but now we vary the size of the keys themselves. This gives application programmers an understanding of what to expect as the number of keys reaches its maximum or if they wish to use large keys to embed metadata or otherwise organize the KV pairs.

Figure 6 depicts the impact different key sizes have on write throughput. The comparison is made when KV pairs are written sequentially in WriteBack-Flush mode (as explained in Section IV-A) with two different key sizes, 4 KB and 16 Bytes. 1,000,000 key-value pairs are sequentially written for each value size from 120 bytes to 1 MB (the maximum value size).

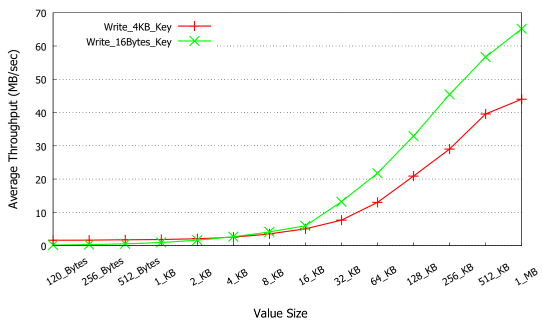

Fig. 6. Average throughputs for minimum and maximum key size.

While Seagate has a modified implementation, based on our understanding of LevelDB, we assume that the better throughput for smaller keys depends on the drive's internal RAM. Larger keys occupy more of the drive's internal RAM. The drive only has 512 MB of RAM for the entire onboard OS and LevelDB system. In this RAM, LevelDB keeps track of the largest and smallest keys of each SSTable. Each SSTable, which is a data structure that stores sorted KV pairs on disk, is 2 MB by default in stock LevelDB. As the value size of the KV pairs increases, fewer KV pairs fit in each fixed-size SSTable. As the number of SSTables increases to hold the larger values, there are more largest/smallest keys to maintain. When the key size is very large, this large number of largest/smallest keys no longer fits in the available memory. This introduces more disk activity which slows the overall throughput.

F. Throughput for P2P Transfer

This section presents the results for a unique feature, P2P transfer, which allows data transfer between two disks without extra network traffic or hardware burden on a separate machine. This feature is especially useful for propagating fault tolerance copies such as for replication. For example, if you want three separate identical copies of some key-value pairs in case one drive fails, you can have the Kinetic drives use P2P transfer in the background to spread these redundancy copies directly to other drives.

The client issues a P2P Get or P2P Put command to a Kinetic Drive (KD-Init) along with the IP address of the target Kinetic Drive (KD-Target). KD-Init establishes a connection over Ethernet to KD-Target and initiates put/write or get/read operations based on the command sent from the client. Once the operation is successfully completed, both the drives have these same key-value pairs. This data copying process takes place without interference from the client. The client only issues the initial instruction to begin the P2P transfer.

In Table V, we show the throughput and internal Kinetic CPU load while using this feature for small and large values. First, we write 512 key-value pairs to KD-Target. After that, we perform the P2P Get operation in which KD-Init copies 512 key-value pairs, each with 1 KB or 1 MB value size, from KD-Target. We see in Table V that the internal processor's utilization is almost 100% when P2P transfer is taking place.

Figure 7 shows the results of a similar experiment using the P2P Put command to write 25,000 KV pairs for various value sizes. We can see that non-P2P Put from the client machine offers better throughput. However, if the client first has to read data from a different Kinetic drive before writing, P2P is more efficient and less burdensome for the client and the network.

TABLE V
THROUGHPUT FOR P2P TRANSFER

Value Size	Read	No. of KV pairs	Internal CPU
1 KB	0.43 (MB/sec)	512	99.01%
1 MB	22.45 (MB/sec)	512	98.02%

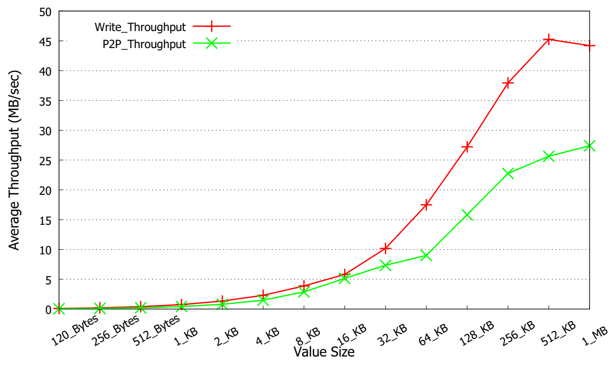

Fig. 7. Throughputs of client-based Kinetic write vs. P2P put for 25,000 KV pairs.

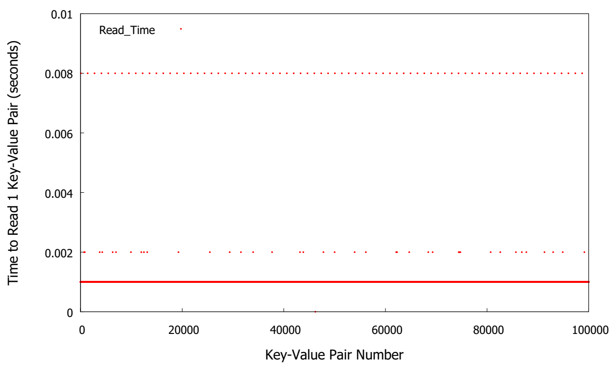

Fig. 8. Keys Not Found - 100k KV Pairs Searched in Random Order With 3.7 TB of Data Written in Sequential Order

G. Kinetic Drive Key Not Found Search Times

In this section, we study the time it takes when the searched keys are "not found" in a Kinetic drive. This aspect is important when a large number of drives are used to create a system. There is a strong possibility that users search for key-value pairs and may not find them in a particular Kinetic drive. Therefore, the users will have to search other Kinetic drives. The time it takes to perform the operation of searching key-value pairs in a particular server or Kinetic drive can affect the entire system's performance.

With 3.7 TB of Data Written: For the first experiment, we write 3.7 TB of data in sequential order in the form of 3,700,000 KV pairs with keys of 16 bytes each and value size of 1 MB. All the keys are unique and "kinetic-c-util" (an included test utility) is used to generate data for each of the 1 MB values. Following that, we request 100,000 non-existent key-value pairs from the Kinetic drive in sequential order. We ensure that the requested keys do not exist in the Kinetic drive. For each request, we record the time it takes for the Kinetic drive to return the status that the requested key is not found in the drive. We plot the "not found" response time of each request in Figure 8. We see that nearly all (99.916%) of the responses occur in one millisecond or 0.001 seconds, while a small number of responses take two or eight milliseconds. This implies there is mostly a network response time delay and only occasional disk seek latency, if any.

We next repeat the same operation; however, we request nonexistent KV pairs in random order. Again, the results look the same and are omitted. We also test the difference when nonexistent keys are inside the range of the written keys (e.g., write key 1 and key 3 and search for key 2 instead of key 4), but we see the same extremely low response time plot. Finally, we experiment with a smaller value size than 1 MB, but the results are generally the same. Small value sizes, such as 4 KB, show a few more requests high enough to be disk activity, but the overwhelming majority are still 1 ms. Because all these tests provide similar results, we cannot determine the effect of read or write randomness in searches for keys that do not exist. However, we can conclude that the Kinetic drive is able to quickly resolve unavailable key requests from its memory even when the drive is full.

These results make sense when you consider that LevelDB stores KV pairs in sorted key order inside 2 MB SSTable files. For each SSTable, the in-memory MANIFEST stores the file name, the largest key, and the smallest key stored in that SSTable. Because we are using a 1 MB value size in our KV pairs, each SSTable can only store two KV pairs. Thus, for 1 MB values, the MANIFEST acts as an in-memory index of every key stored, and requests for nonexistent keys can be processed immediately no matter the order of the initial writes or later reads. If the MANIFEST is not as helpful in determining if the requested key is stored, e.g., when we use smaller value sizes and search for keys in an overlapping range, the next layer is a Bloom filter. Only in the rare instance that the Bloom filter is unable to correctly determine that a key is not stored (i.e., a false positive) does LevelDB actually have to read the SSTable from the disk to check. We believe that some of the 2 ms read times are Bloom filter checks, and only the 8 ms searches involve actual disk accesses.

In summary, we can conclude that the time spent fulfilling a request for each key not found is significantly less than performing a successful read operation in the Kinetic drive when using 1 MB value sizes. When a key is requested that does not match the previously written keys, the in-memory MANIFEST and a fall-back Bloom filter can rapidly report the key as not found without the need for very many actual disk accesses.

V. Throughput Comparisons of Kinetic Drives and LevelDB Servers

This section compares the throughput of Kinetic drives against traditional servers running LevelDB (Server-SATA and Server-SAS). We believe it is important to understand the possible implications of replacing traditional hard drives with Kinetic drives for various workloads and disk configurations. As described in Section III, we have two different LevelDB-based server systems. Server-SATA has 12 GB of RAM and two extra 5900 RPM SATA disks with similar properties to the Kinetic drives, and the faster Server-SAS has 64 GB of RAM with several 7200 RPM enterprise drives.

We make comparisons when data is written and read back in a random key order (sequential results are omitted for space).

We use db_bench.cc, which comes with LevelDB, to perform random key order reads and random key order writes with various value sizes on a single drive as well as multiple drives.

Every point on the x-axis in Figure 9 shows several different experiments. For example, 120_B represents 1,000,000 KV pairs, each with value size of 120 bytes and key size of 16 bytes, first written in random key order for one experiment and then read back in random key order for another experiment. The total data transfer of each run is 120 MB. Similarly, 256_B represents another 1,000,000 KV pairs written and then read back, each with a value size of 256 bytes (total data transfer is 256 MB per run). It is also important to note that the y-axis is in log scale so that high throughputs do not overwhelm the plots. This happens because some of the server read experiments with smaller value sizes fit mostly in the server RAM and avoid many of the slower disk accesses. For read tests, the LevelDB benchmark first loads (writes) some data to read but does not purge any system caches before measuring the reads. It is likely that some (or, for small tests, all) of these previously written/loaded key-value pairs already exist in the server's cache when read performance is measured. Significant modifications to the LevelDB benchmark or adjustments to default settings, to hinder the servers by eliminating this caching, are beyond the scope of this research.

Looking at Figure 9, we notice that as the value size increases, the server's random write throughput decreases below 1 MB/sec. The larger value sizes were so slow that we were forced to abandon those tests. As the value size increases, so does the amount of data per experiment, the number of LevelDB files, the amount of key-value pair sorting, and overall disk activity. Moreover, irrespective of different RAM sizes, the write throughput for Server-SATA and Server-SAS both decline at similar rates as we increase the value size. When key-value pairs are written, they initially are written to a 4 MB memtable. Subsequently, the memtable is flushed to the drive in the form of 2 MB SSTables. Since such a limited amount of data is written in each memtable before flushing, the RAM of Server-SATA and Server-SAS has a limited impact on write performance.

The most interesting observation is that the Kinetic drive (Figure 9) can sustain higher throughput than a LevelDB server for large values sizes. Unlike the open-source API, the Kinetic drives themselves are closed-source black boxes. Therefore, the following hypothesis to explain their relative performance is difficult to verify. We assume that there is more write amplification during random key order writes for the default LevelDB configuration on the servers than for the optimized LevelDB in the Kinetic drive [14]. It is likely that the memtable and SSTable size is larger for Kinetic drives compared to LevelDB's default. This increased table size would create fewer files thus reducing read and write overhead. However, for small value size experiments, the servers with their large RAM size for reads and faster CPUs for writes can

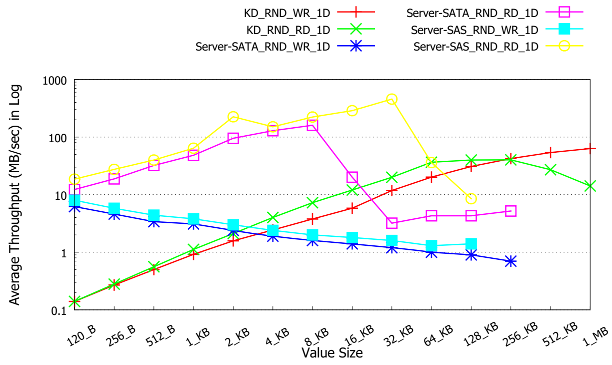

Fig. 9. Random Order Key (RND) Read (RD) and Write (WR) throughput comparison of a Kinetic Drive (KD), Server-SATA, and Server-SAS using One Drive (1D).

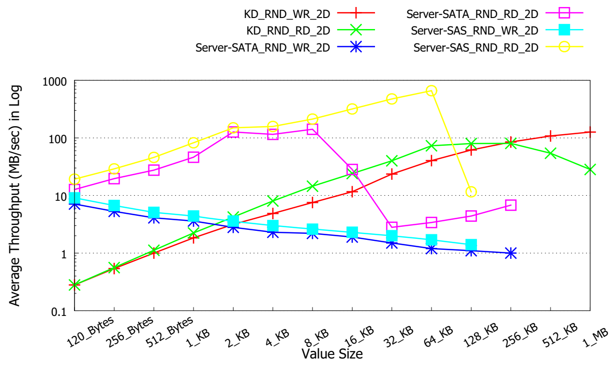

Fig. 10. Random (RND) Read (RD) and Write (WR) throughput comparison of Kinetic Drives (KD), Server-SATA, and Server-SAS using Two Drives (2D).

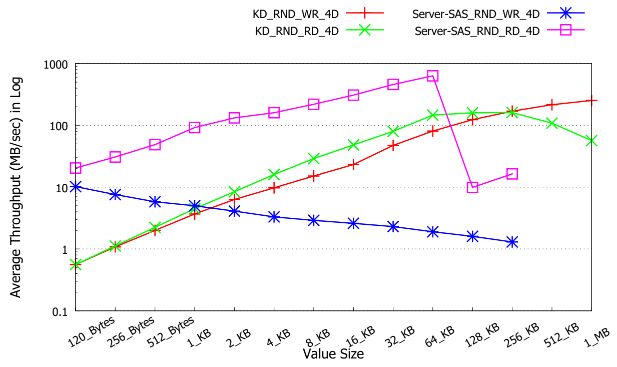

Fig. 11. Random (RND) Read (RD) and Write (WR) throughput comparison of Kinetic Drives (KD) and Server-SAS using Four Drives (4D).

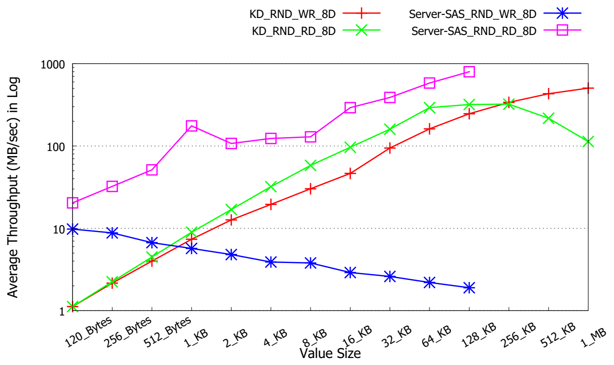

Fig. 12. Random (RND) Read (RD) and Write (WR) throughput comparison of Kinetic Drives (KD) and Server-SAS using Eight Drives (8D).

easily outperform a Kinetic drive. Overall, even ignoring the small experiments that mostly fit in server RAM, we can also conclude that read operations achieve higher throughput than writes for all of the three systems. During reads, the drives simply have to deliver data that has already been sorted by the systems. When there are many writes, LevelDB has to do compaction and level management that introduces write amplification affecting the throughput.

We extend these experiments by adding drives, striped in software RAID-0, to Server-SATA and Server-SAS as shown in Figures 10, 11, and 12. Only Server-SAS has enough disks for four and eight drive tests. The multiple drive Kinetic results in these figures are a linear extrapolation of the single drive results in Section IV-C. By conducting simultaneous parallel experiments on all four Kinetic disks, we verified that there is no individual performance degradation. Therefore, we assume the load is balanced evenly across the independent disks and the performance will scale linearly until the Kinetic enclosure's 10 Gbps network link becomes a bottleneck. While the LevelDB servers do show throughput increases in some cases as we add more disks, their performance does not scale up as much as expected.

In these figures, the larger memory size of Server-SAS allows it to demonstrate significantly higher read throughput than the Kinetic drives or Server-SATA for small value sizes. However, Server-SATA and Server-SAS both reflect a "memory cliff" trend in read throughputs. After the read throughputs increase continuously, they suddenly drop beyond a certain value size. As the amount of data transferred in each experiment increases with value size, the system memory cannot continue to hide slower disk accesses and throughput drops. At these points, the Kinetic drives usually take over and start giving better throughputs than either of the servers.

VI. Latency Tests of Kinetic Drives and LevelDB Servers

Latency is a measure of the delay between when data is requested and when it actually arrives. This is critical for Quality of Service (QoS) of time sensitive applications. Moreover, latency is an indicator of the response time of a server. Users often dislike long wait times for their applications.

Read Latency of Kinetic Drives Following Random and Sequential Order Writes: Here we obtain and compare latency results for random key order reads following random key order writes or sequential key order writes (described in Section IV-A) on the Kinetic drives, Server-SATA, and Server-SAS. There are three different sets of tests conducted:

- Random Read Latency for Kinetic Drives after 100 GB or 3.7 TB of Random Order or Sequential Order Writes
- Random Read Latency for Server-SATA after 100 GB of Random Order or Sequential Order Writes
- Random Read Latency for Server-SAS after 100 GB of Random Order or Sequential Order Writes

Figure 13 presents a histogram of our Kinetic results. We first perform the 100 GB Kinetic sequential key order write operation as described in Section IV-A and then record latency

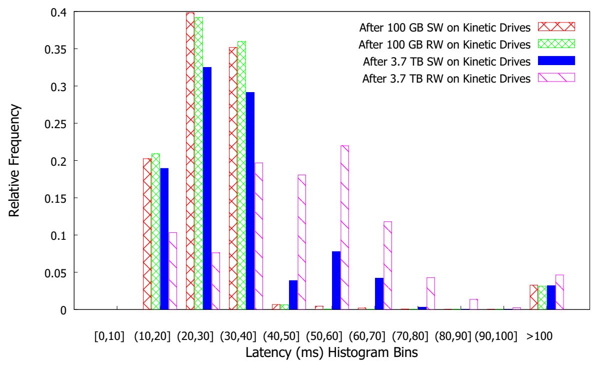

Fig. 13. 100 GB Random Read after Sequential Order Writes (SW) or Random Order Writes (RW) on Kinetic Drives

as we read 100 GB in random key order by requesting 100,000 key-value pairs (writes and reads use value size of 1 MB and key size of 16 bytes). Having recorded the time it takes to successfully complete each random read request, we put these latencies in different histogram bins to determine a count, or frequency of accesses, in that latency range. We obtain relative frequencies by dividing the number in each bin by the number of key-value pairs (100,000 in this case). We repeat this process for various write patterns to complete Figure 13.

Looking at Figure 13, we can observe that the read latencies are higher when the Kinetic drive is full, and reads following a drive filled with random order writes show the highest latency. In the figure, we group all latencies above 100 ms, which account for approximately 4-5% of the reads, into the rightmost bin. For a read request, LevelDB may have to read files from multiple levels before it finds the correct KV pair. Reads satisfied by early levels like L0 or L1 will have lower latencies than requests that still have to access files in early levels but are not satisfied until L5, for example. The more data we write to the drive, the more data resides in higher/later levels of LevelDB, and reads from these higher levels cause the higher latencies.

Next, we run similar latency tests with the LevelDB benchmarking program, db_bench.cc, on Server-SATA and Server-SAS and plot our findings in Figure 14. From these results, we observe that a substantial number of read latencies for

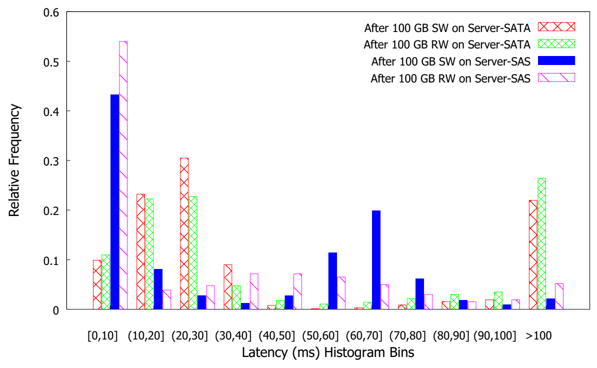

Fig. 14. 100 GB Random Read after Sequential Order Writes (SW) or Random Order Writes (RW) on Server-SATA and Server-SAS

Server-SAS fall in the "[0,10]" bin, which implies that Server-SAS tends to be faster for read operations than Server-SATA. These findings are again reflective of the larger memory size of Server-SAS (64 GB vs. 12 GB in Server-SATA), its faster memory (2100 MHz DDR4 vs. 1600MHz DDR3 in Server-SATA), and its faster HDD (7200 RPM vs. 5900 RPM in Server-SATA). Compared to Figure 13, we see that Server-SATA has many more >100 ms reads than the Kinetic drive.

This trend is due to the same optimizations we describe in the previous section that reduce read and write amplification in the Kinetic drive compared to the default LevelDB server. Seagate's Kinetic optimizations allow it to outperform Server-SATA, which has a similar hard drive, while the obvious hardware advantages of Server-SAS allow it to dominate over the other systems.

Overall, we can conclude that the amount of data written and the amount of randomness in that data affect the latencies for subsequent read operations on these Kinetic drives.

VII. Yahoo Cloud Serving Benchmark

In addition to the micro benchmarks discussed in the previous sections, we choose a popular macro benchmark, Yahoo Cloud Serving Benchmark (YCSB) [15], to simulate several more realistic application-level workload mixes. Because there is no direct LevelDB interface to YCSB, we initially tried to use Mapkeeper [16] as the key-value store with LevelDB as the storage backend. Unfortunately, Mapkeeper is no longer supported by YCSB. After much effort, we were only able to make it work with an older version of YCSB, but even then there were file locking issues after the YCSB load phase that prevented the run phase from working correctly.

Another key-value store, Riak KV [17], can use an optimized version of LevelDB as its storage backend. Riak is under active development and works with the newest version of YCSB. Our YCSB testing for a normal hard drive uses a single Riak node configured on Server-SATA to use LevelDB as the backend and the 4 TB SATA drive as the data storage location. Because we use only one node, we set the number of data copies from the default of three to only one. All other parameters are left as their defaults. We first run the Riak tests with the YCSB client running locally on the Riak node itself. Then, in order to add a network layer similar to that used by the Kinetic drives, we repeat the experiments with the YCSB client sending data over a Gigabit Ethernet network to the Riak node from an identical machine (i.e., the same specs as Server-SATA). These tests use YCSB version 0.12.0 and Riak version 2.2.0. For YCSB tests on the Kinetic drives, we use the version that comes with the Kinetic Java Tools software [18]. Both the normal HDD and the Kinetic YCSB testing use the same workload parameters for a fair comparison.

YCSB includes the following six workload patterns: Workload A is a heavily updated workload with a 50/50 read/write ratio. Workload B is a read-mostly workload with a 95/5 read/write ratio. Workload C is 100% reads. Workload D has 5% updates and 95% reads, but the recently updated records are the ones that are read the most; thus, it is a read-latest workload. Workload E queries short ranges of records, but the scan command is not yet implemented on the Kinetic drives, so we do not use this workload. Workload F is half reads and half read-modify-write commands where a record is read, updated, and written back.

The YCSB load phase of Workload A is 100% writes, and we follow that with a run of Workloads A, B, C, D, and then F. We insert a pause of five minutes between each workload to allow the system some idle time for LevelDB compaction or other internal processes.

Figure 15 shows the average throughput while loading and running these workloads with a value size of 1 MB. For each workload, 50 GB of data is transferred. In this case we see that a Kinetic drive is able to consistently outperform the Riak/LevelDB node for both the load and run of Workload A as well as for the run of Workload F. For runs of Workloads B, C, and D, the Kinetic drive shows throughput similar to or slightly worse than that of the Riak node when its YCSB test is run locally. When YCSB is run over the Gigabit network to the Riak node, its performance declines, and the Kinetic drive shows similar or slightly better throughput for those workloads. The Riak node shows poor performance in workloads with large amounts of updates/writes like Workloads A and F.

Figure 16 displays the results for a value size of 4 KB. In order for the experiment to complete in a reasonable amount of time, the data transferred per workload was reduced to 5 GB. Here we see that the Kinetic drive becomes I/O bound due to the large number of small accesses while the Riak server,

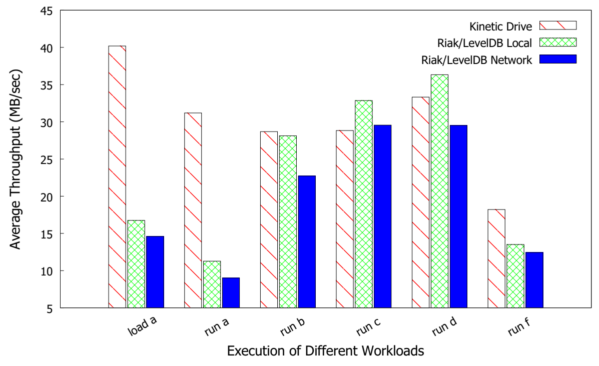

Fig. 15. Average throughput for loading phase and running phases of different YCSB workloads using 1 MB value size on Kinetic drive vs. Riak server with LevelDB backend.

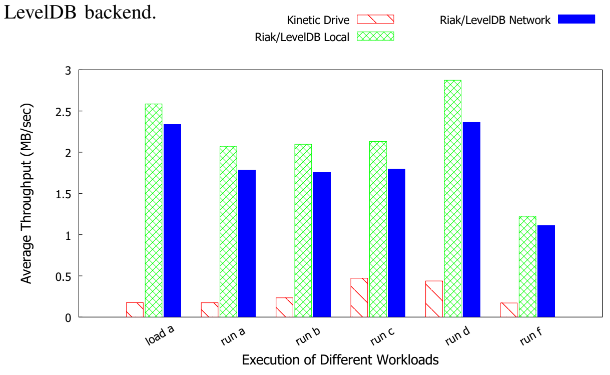

Fig. 16. Average throughput for loading phase and running phases of different YCSB workloads using 4 KB value size on Kinetic drive vs. Riak server with LevelDB backend.

with its larger amount of RAM, is able to better organize these small transfers for more efficient disk access. However, the performance of both the Kinetic drive and the Riak node is far lower with 4 KB values than with 1 MB values. Our findings with YCSB confirm our findings of the Section IV-C. This poor performance with small I/Os is typical of rotating mechanical media and is due to the disk seek time. Again, the Riak tests suffer a slight drop in performance when run over the network compared to when run locally.

In summary, the YCSB tests use a more realistic configuration to reaffirm the earlier micro benchmark results. For smaller value sizes, the full server is easily able to show higher performance; but, for larger value sizes, the Kinetic drive can perform surprisingly well, especially in write-intensive workloads where it outperforms the server for similar reasons as discussed in Section V.

VIII. Conclusion

Based on our study, we conclude that our sample Kinetic drives provide an average read throughput of 78 MB/sec and an average write throughput of 63 MB/sec with a large value size. These drives scale well in a data center environment, because each drive acts as a tiny independent key-value storage server. Moreover, new features such as P2P data transfer are unique to Kinetic drives. However, the tested first generation Kinetic drives do have significant bandwidth limitations due to inadequacies of the internal processor. This internal CPU shows nearly 100% utilization during many of our tests where the hardware should otherwise be able to produce higher throughput. Seagate is aware that the internal CPU of these sample drives is a performance bottleneck and newer versions of their Kinetic drives, should they become widely available, will have a faster processor. Despite this limitation, for some applications with large value sizes or many searches for keys that do not exist, the Kinetic drives can outperform a SATA-based mid-range server and provide better scalability.

Acknowledgments

This work was supported in part by the Center for Research in Intelligent Storage (CRIS), which is supported by National Science Foundation grant no. IIP-1439622 (July, 2014 - August, 2019) and member companies. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF. This work was partially supported by Hankuk University of Foreign Studies Research Fund. The authors would like to thank Seagate for their support of this project and, in particular, Bryan Wyatt for his technical assistance.

References

[1] M. Walker, "Structured vs. unstructured data: The rise of data anarchy," http://www.datasciencecentral.com/profiles/blogs/structured-vs-unstructured-data-the-rise-of-data-anarchy, 2012, accessed: 2017-6-27.

[2] J. Mallory, "Save your data deep storage considerations," http://web.stanford.edu/group/dlss/pasig/PASIG_March2015/20150313_Presentations/Mallory_Isilon.pdf, 2014, accessed: 2017-6-27.

[3] D. Gascon and G. White, "Managing unstructured data in object-based storage," http://www.dell.com/downloads/global/power/ps4q10-20100472-white.pdf, 2010, accessed: 2017-6-27.

[4] J. Arnold, "Kinetic motion with seagate and openstack swift," https://swiftstack.com/blog/2013/10/22/kinetic-for-openstack-swift-with-seagate/, 2013, accessed: 2017-6-27.

[5] "Kinetic open storage project," https://www.openkinetic.org/, 2016, accessed: 2017-6-27.

[6] 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. [Online]. Available: http://doi.acm.org/10.1145/1323293.1294281

[7] "How we made github fast," https://github.com/blog/530-how-we-made-github-fast, accessed: 2017-6-27.

[8] "Rocksdb," http://rocksdb.org/, accessed: 2017-6-27.

[9] "Seagate's kinetic will impact object storage," https://go.forrester.com/blogs/seagates-kinetic-will_impact-object-storage-and-data-driven-applications, 2013, accessed: 2017-9-27.

[10] Google, "Leveldb - lightweight database library by google," https://github.com/google/leveldb, 2016, accessed: 2017-6-27.

[11] M. Shetty, "Seagate kinetic open storage platform," http://www.snia.org/sites/default/files/MayurShelty_Seagate-Kinetic.pdf, 2014, accessed: 2017-6-27.

[12] Seagate, "Seagate kinetic hdd," http://www.seagate.com/www-content/product-content/hdd-fam/kinetic-hdd/en-us/docs/100764174b.pdf, 2014, accessed: 2017-6-27.

[13] S. Seagate, Supermicro, "Reference design: Scalable object storage with seagate kinetic, supermicro, and swiftstack," https://swiftstack.com/wp-content/uploads/2015/05/KineticReferenceDesign-SwiftStackSupermicroSeagate.pdf, 2015, accessed: 2017-6-27.

[14] L. Lu, T. S. Pillai, H. Gopalakrishnan, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "Wisckey: Separating keys from values in ssd-conscious storage," *Trans. Storage*, vol. 13, no. 1, pp. 5:1–5:28, Mar. 2017. [Online]. Available: http://doi.acm.org/10.1145/3033273

[15] K. Risden, "Yahoo cloud serving benchmark," https://github.com/brianfrankcooper/YCSB/wiki, 2014, accessed: 2017-6-27.

[16] "Mapkeeper," https://github.com/m1ch1/mapkeeper/wiki, 2014, accessed: 2017-6-27.

[17] "Riak," http://docs.basho.com/riak/kv/2.2.0/, 2014, accessed: 2017-6-27.

[18] "Kinetic java tools," https://github.com/Seagate/kinetic-java-tools, 2014, accessed: 2017-6-27.