STORAGE

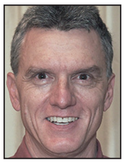
Eusocial Storage Devices

Offloading Data Management to Storage Devices that Can Act Collectively

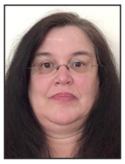
PHILIP KUFELDT, CARLOS MALTZAHN, TIM FELDMAN, CHRISTINE GREEN, GRANT MACKEY, AND SHINGO TANAKA

Philip Kufeldt is a Director of Storage Standards at Huawei Technologies, focusing on NVMe, new key value, and smart media-based storage standards. He has over 20 years of experience in storage, software, and systems. Before Huawei, Philip was at Toshiba, creating new smart media devices; Marvell, Parascale, VERITAS Software, Sun Microsystems, and IBM; and he has founded several storage-oriented startups. pak@protium.com

Carlos Maltzahn is an Adjunct Professor of Computer Science at UC Santa Cruz and the Director of the Center for Research in Open Source Software (CROSS) and the Systems Research Lab (SRL). Carlos graduated with a PhD in computer science from the University of Colorado at Boulder. carlosm@ucsc.edu



Tim Feldman works on drive design at Seagate Technology's Longmont, Colorado, Design Center. His current work focuses on object storage. He also spends time randonneuring, Nordic skiing, and logging. timothy.r.feldman@seagate.com



Christine Green led the Kinetic Open Source Project development at Seagate and continues work on Seagate's ActiveDrive™ technology. Her HDD background includes experience with recording heads and media as well as VLSI and signal processing. She has a BS in electrical engineering from Stanford University. christine.green@seagate.com

As storage devices get faster, data management tasks rob the host of CPU cycles and DDR bandwidth. In this article, we examine a new interface to storage devices that can leverage existing and new CPU and DRAM resources to take over data management tasks like availability, recovery, and migrations. This new interface provides a roadmap for device-to-device interactions and more powerful storage devices capable of providing in-store compute services that can dramatically improve performance. We call such storage devices "eusocial" because we are inspired by eusocial insects like ants, termites, and bees, which as individuals are primitive but collectively accomplish amazing things.

The Evolution of the Problem

Why Try Smart Storage Again, and Why Now?

Offloading storage processing has been around since the earliest days of computing. The idea of having a dedicated and cheaper I/O processor offloading the main processor complex made sense at a time when processor cycles were incredibly scarce and costly. However, over the years, processor cycle availability has geometrically increased and costs have plummeted making the utilitarian I/O processor costlier in terms of complexity in both hardware architecture and software. These fast and large CPU complexes permit general-purpose execution and I/O management, including data management. Data management tasks are beyond the basic tasks of storing and retrieving data, including services such as translation, mapping, deduplication, compaction, sorting, scrubbing, data movement, data redundancy, and recovery.

Including I/O management created a tight coupling of storage with the server system architecture. With such compute resources available, storage devices need only do the media management and map logical placement information to physical placement information, leaving essentially all data management relegated to the general-purpose processor. Furthermore, the simplistic API required to accomplish these goals treats every device as completely independent even though data management necessarily creates device relationships and dependencies, all of which have to be managed by the general-purpose processor.

This has driven the evolution of the storage component towards a highly cost-efficient model that has resisted most attempts to offload tasks to the component. Attempts to push some of data management back into the device, such as SCSI OSD or Kinetic, have all failed due to the need for additional compute and memory in the device pushing up per-GiB costs.

NAS Succeeds in Offloading

The one place where data management offloading was successful was Network Attached Storage (NAS). NAS environments offload all the data management to centralized servers on the network (Figure 1).

Client servers then use a network-based access protocol (NFS, CIFS) to store and retrieve data. The reason for the offloading was two-fold:

Grant Mackey works as a researcher in the office of the CTO at Western Digital Corporation in Irvine, California. His current work focuses on modeling and simulation of data-centric computing. His off time is spent hiking, cooking, and fiddling with IoT devices at home. grant.mackey@wdc.com

Shingo Tanaka is working on new types of SSD projects in the Flash Storage Department, SSD Division, Toshiba Memory Corporation, Tokyo, Japan. His current work focuses on higher functioning SSD, which can offload host tasks into SSD, effectively integrating them into existing SSD architecture. shingo3.tanaka@toshiba.co.jp

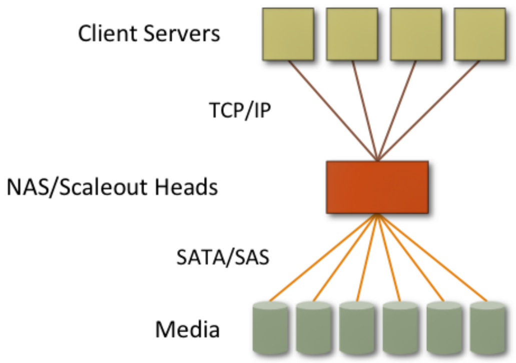


Figure 1: NAS provides centralized storage management while disaggregating storage from the server.

1. Centralizing storage management
2. Disaggregating the storage from the server

The former is a straightforward centralization gain; by consolidating all of the server management touch points, management man hours were reduced. Disaggregation was driven by the need for sharing and availability. By separating data resources from the local server and its processor complex, these resources could be placed on a general-purposes network. This provided two big wins: the data could be shared by many servers, and data resources could remain available even with the loss of the local server. This did introduce additional costs into the data layer. It was made cost effective by scaling up the number of storage devices and, hence, the number of GiB available attached to a NAS server, driving down the per-GiB costs. Also, being a central resource, this cost was further diluted by the increased number of servers being served. Even with the gains, the NAS servers themselves were tightly coupled to the storage, which created scaling limits and vulnerable islands of storage.

So, what has changed?

1. The commodity smartphone market over the last 10 years has driven the cost of embedded multi-core 64-bit processors, such as ARM, way below the cost of server processors.
2. The smartphone market also drove the power consumption of these embedded processors way down.
3. The densities of storage devices continue to skyrocket, making it easier to hide additional computing resources in the per-GiB cost.
4. New denser flash media is permitting the bandwidth aggregation of many discrete flash chips, making a single device capable of streaming GiB/s of throughput, and they will continue to get faster. This new performance level demands greater processor capabilities, and, as such, the processor costs are already partially priced into the per-GiB cost of flash devices.

But there is more. High-speed flash devices must be connected to the processor complex. Luckily, PCIe bandwidth has kept pace, growing rapidly with PCIe v3/v4, and PCIe v5 is on the horizon. However, the same cannot be said about the ultimate destination of data-system memory (Figure 2). System memory bandwidth is growing at a much slower pace than the flash devices and their interconnects. Since all I/O requests must ultimately be transferred into system memory, the system memory is already becoming the next real bottleneck.

As the storage devices deliver higher and higher throughput, the system memory bottleneck will reduce the number of storage devices that a server can effectively utilize. This problem requires that the data transferred by the server to the storage be classified and prioritized.

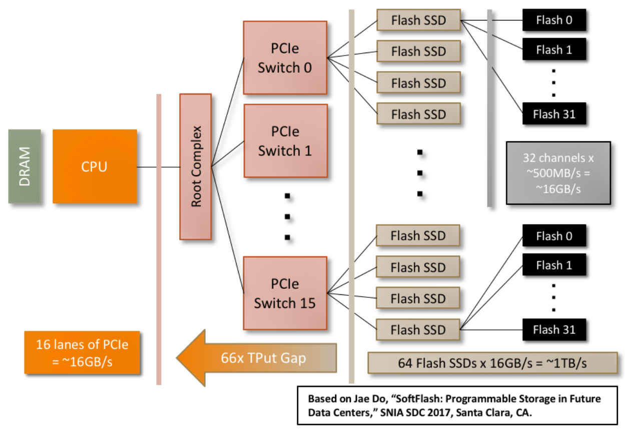

DRAM
CPU
Root Complex
PCIe Switch 0
PCIe Switch 1
PCIe Switch 15
Flash SSD
Flash 0
Flash 1
Flash 31
32 channels x ~500MB/s = ~16GB/s
16 lanes of PCIe = ~16GB/s
66x TPut Gap
64 Flash SSDs x 16GB/s = ~1TB/s
Based on Jae Do, "SoftFlash: Programmable Storage in Future Data Centers," SNIA SDC 2017, Santa Clara, CA.

Figure 2: Throughput mismatch, based on Jae Do, "SoftFlash: Programmable Storage in Future Data Centers," SNIA SDC 2017, Santa Clara, CA

Data transfers strictly for data management (mapping/placement, scrubbing, redundancy, recovery, and accessibility) is of less benefit than actual I/O transfers for real work and should be offload targets.

RocksDB as an Example

Take the example of a key-value store like RocksDB, a library that allows an application to maintain a key-value database. The real work by the application is not likely to be the storing and retrieving of data; rather, the application is dependent on being able to persistently put and get data to/from the RocksDB store. This means any I/O transfers done to map the data and ensure its durability, availability, or accessibility is work done outside of the knowledge of the application.

RocksDB maps the key-value data through a data structure called a log-structured merge-tree (LSM). This LSM tree is implemented atop a file system, which in turn uses a block device to persistently store and retrieve data (Figure 3). The LSM tree itself constantly sorts the data as it comes into the store. This requires that large sections of data be read into the server memory via the file system and block device, manipulated and then stored into new file system structures. In addition, RocksDB ensures durability by checksumming the data as it is added to the store. It then periodically scrubs the data by transferring it to server memory and validating the checksums. All of this I/O can be considered north-south data transfers, moving data in and out of the storage device. RocksDB is not only consuming processor cycles in the sorting and scrubbing of data but, more importantly, is consuming memory bandwidth for its own data management outside of the application's knowledge.

This example gets worse when considering availability, recovery, and accessibility. In this example, RocksDB is using local storage resources. To guarantee availability of the store even if the

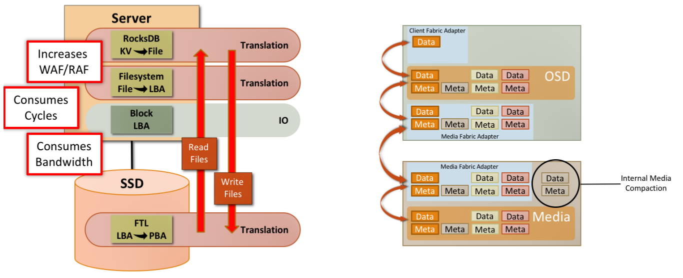


Figure 3: Translations for data management and DMA use for data management. WAF/RAF stands for the write/read amplification in Flash devices.

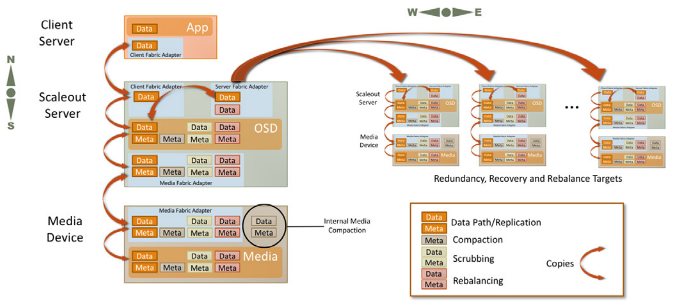


Figure 4: When scaling out a key-value database, data moves in two different dimensions: within the scale-out server (north-south) and between servers (east-west).

server fails, the data would need to be replicated to other servers with storage resources. This means data has to be transferred not only to local storage controllers but also to network controllers, greatly increasing the memory bandwidth usage. These transfers can be considered east-west transfers because the data moves laterally from one server to another (Figure 4).

Recovery again potentially moves data east and west when there are failures. Data accessibility incurs east and west data movement for the purposes of tiering, caching, or load balancing across a set of servers.

All of these activities increase the usage of the server memory bus for data management, putting the data management directly in contention with the application. With the advent of the cheap, low-power embedded processors, high-density storage devices, and high-speed devices, it is now time again to look at offloading data management to the devices themselves.

Goals of the Solution

The current standard storage API characterizes a storage device's available space as a static, linear address space of contiguous fixed-size data blocks. These address spaces can be segmented (partitioned) but have the same properties as the parent address space. Data within these address spaces is accessed in block granularities by giving a location address (logical block address, LBA) within the address space and the number of sequential blocks to be transferred. This interface requires applications to locate their data by remembering the device, the partition, the LBA, and the length.

This location-centric model provides no other alternatives for abstract data location, data layout, redundancy, deduplication, sorting, scrubbing, data movement, data recovery, QoS, etc. Therefore, offloading data management to a storage device is more than expansion of the current storage device API—it is indeed a complete sea change.

It is important to understand the scope and high-level goals of a new storage API. The goals are:

1. Data placement within a device should be abstracted from the I/O path. Consequently, data layout should be opaque.
2. Data location should be abstracted from the I/O path.
3. Data movement from one device to another should be abstracted from the I/O path.
4. Data availability should be configurable and abstracted from the I/O path.
5. Data recovery and repair should be abstracted from the I/O path.
6. Data attributes should be supported.
7. Data access at scale should be supported.
8. Design should be mechanism-based, leaving policy to be defined by the user.

Introducing Eusocial Storage

Eusocial storage is a new API definition that drives data management activities into the device and sets a course towards in-store compute functionality. It takes into account today's scale requirements and builds on top of them.

Software

Eusocial storage is a mechanism-based software abstraction that standardizes a network/fabric-based object protocol that supports variable-sized keys and objects (Figure 5). Eusocial

- inherently disaggregates, permitting composable systems;
- inherently abstracts data location, permitting dynamic systems like scale-out storage with data availability, scaled access, and dynamic balancing;
- inherently reduces failure domains;
- supports peer-to-peer interactions, permitting autonomous data availability and data migrations between devices;
- supports device class organization, permitting scaling on a class-of-service basis;
- supports user-defined but autonomous data migrations between classes, providing for user-defined tiering and caching;
- supports in-store computing.

Hardware

Eusocial places no hard requirements on the hardware other than it must support a bidirectional network or fabric to satisfy the disaggregation, peer to peer, cluster, and control requirements. Other than this network requirement, the hardware can be defined in any fashion and take any form. There are no restrictions on media type, form factor, capacity, components, and fabric type. For example, a eusocial storage device could be a small Ethernet-enabled SSD, a small sled of HDDs, an optical jukebox, or even a media-less gateway to S3. It is anticipated that manufacturers will compete on designing hardware that is highly optimized for the media type or the targeted class of service.

Organization

Eusocial is organized into levels: storage devices, castes, and the cluster (Figure 6). The storage devices represent highly optimized, autonomous units of object-based storage. These devices define the lines of service they provide, such as throughput, latency, media type, and in-store compute availability. Devices that have similar lines of service can be organized into castes. Within a caste, devices scale out a line of service providing for data availability, data accessibility, and potentially in-store

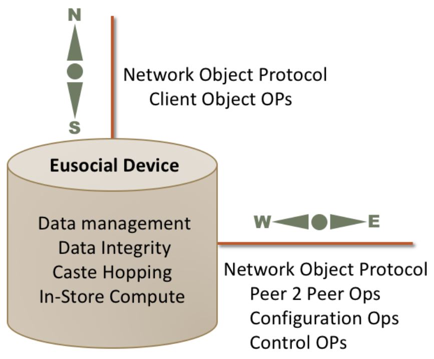


Figure 5: Eusocial device

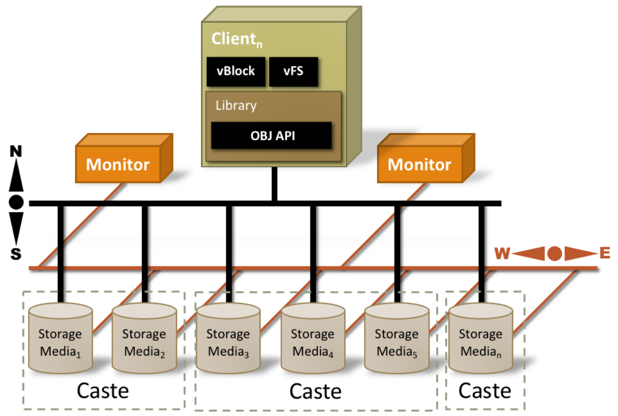


Figure 6: Eusocial hierarchy

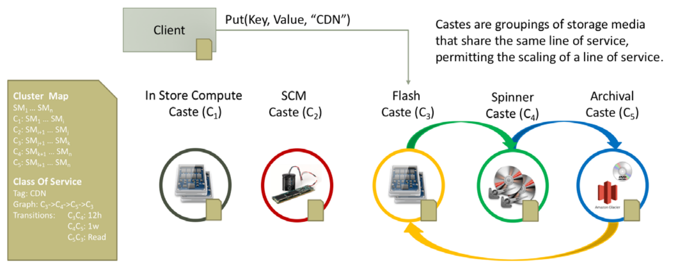

Client
Put(Key, Value, "CDN")
Castes are groupings of storage media that share the same line of service, permitting the scaling of a line of service.
Cluster Map
SM_1 … SM_n
C_1: SM_1 … SM_i
C_2: SM_{i+1} … SM_j
C_3: SM_{j+1} … SM_k
C_4: SM_{k+1} … SM_n
C_5: SM_{l+1} … SM_n
Class Of Service
Tag: CDN
Graph: C_3->C_4->C_5->C_3
Transitions: C_3C_4: 12h
C_4C_5: 1w
C_5C_3: Read
In Store Compute Caste (C_1)
SCM Caste (C_2)
Flash Caste (C_3)
Spinner Caste (C_4)
Archival Caste (C_5)
Amazon Glacier

Figure 7: Eusocial caste hopping

compute. As an example, one can imagine having the following storage-media groupings:

- High-speed: a caste consisting of scaled out replicated enterprise eusocial SSDs
- Warm: a caste consisting of scaled out replicated eusocial HDDs
- Cold: a caste consisting of scaled out erasure-encoded and spin controlled eusocial HDDs
- S3: a caste consisting of a gateway to S3
- Compute: a caste consisting of scaled out in-store computing-enabled eusocial devices

Once the devices are organized into castes, users can define the lifecycle of an object by defining caste relationships, called caste maps (Figure 7). These maps plot the trajectory of an object through a set of castes and what events trigger objects to move. Once a map is defined by a user, applications can tag objects with the appropriate map. Eusocial devices themselves are responsible for following these maps and moving the data as dictated by the map.

Ultimately, the eusocial devices and castes exist inside a cluster that shares the whole configuration with all members. The cluster is also responsible for managing events and event notification. Clients also receive the configuration so that they can get and put data within the system.

Because eusocial storage is a scale-out object protocol, traditional access methods such as block and file access would be implemented atop the eusocial protocol.

The Evolution of the Solution

Changing the API is a significant issue since the block storage APIs have been ingrained in our programming model for decades. Consequently, all server software has been written to the block interface. Changing this interface will require time and some strategy to occur.

The good news is that there are significant numbers of applications that use placement abstraction storage APIs such as key-value or object. Today, these applications require a layer(s) of software like a file system to map the abstracted data to the block interface. Removing these mapping or translation layers can provide not only performance enhancements to the app but also can return processor cycles and DMA bandwidth back to the server. Paying close attention to these applications' requirements creates a ready-made set of consumers for the new API.

In addition to picking the right first-use cases, care needs to be taken on how to roll out the API's inherent complexity. An API roadmap can be broken down into several discrete steps:

1. North-south data management offloading
2. Disaggregation
3. East-west data management offloading
4. In-store computing

North-South Data Management Offloading

A natural starting point for data management offloading is happening in the industry today. There are standards bodies already working on creating a simple key-value command set that will provide an alternative to the standard block command set. This work introduces the notion of an API that has abstracted placement information behind a key-value interface and places the work of maintaining the key value store inside the device. Although initially targeted for direct connect devices, this work is being done so that it can be easily used with disaggregated protocols as well. This step begins moving applications away from the block interface programming model and onto a key-value-based interface.

The initial industry-based work will target those ready-made consumers who already use key-value APIs but have to depend on server software to implement key-value store. This effort will provide a proof point and beachhead for the eusocial API work.

Eusocial will build on this by completely providing a full object API between a eusocial device and a host. Initially this can be done on directly connected devices.

Disaggregated Storage

Many and diverse solutions prove that disaggregation is beneficial to storage workloads: NAS, Ceph, Swift, Gluster, etc. Hence the eusocial approach is revolutionary not because it is arguing for disaggregation. The novelty is the granularity of that disaggregation is now properly attributed to singularly capable devices rather than a singular server (potentially far over-scoped) responsible for a collection of dependents. So instead of having to fan-in clients only to fan-out to media (Figure 1), the eusocial approach constructs a virtual crossbar allowing clients to talk directly to all the storage devices (Figure 8).

The resulting shift increases the number of devices that need to be managed by some type of service, and that can be seen as a detriment to eusocial storage. However, we argue that this added

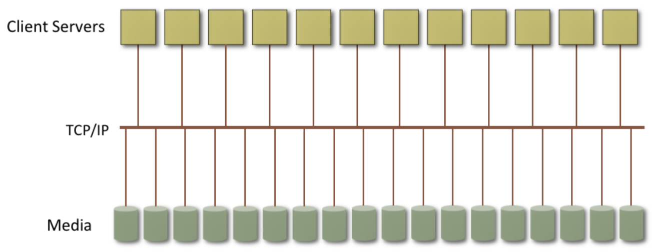


Figure 8: Full crossbar disaggregation

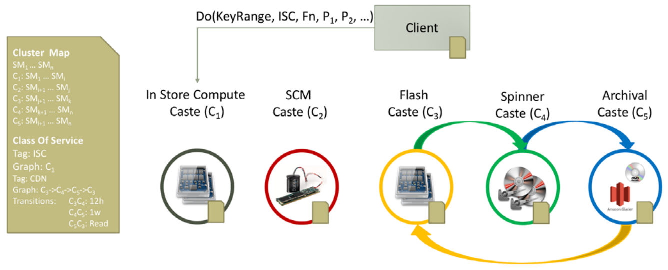

Do(KeyRange, ISC, Fn, P_1, P_2, ...)
Client
Cluster Map
SM_1 ... SM_n
C_1: SM_1 ... SM_i
C_2: SM_{i+1} ... SM_j
C_3: SM_{j+1} ... SM_k
C_4: SM_{k+1} ... SM_n
C_5: SM_{l+1} ... SM_n
Class Of Service
Tag: ISC
Graph: C_1
Tag: CDN
Graph: C_3->C_4->C_5->C_3
Transitions: C_3C_4: 12h
C_4C_5: 1w
C_5C_3: Read
In Store Compute Caste (C_1)
SCM Caste (C_2)
Flash Caste (C_3)
Spinner Caste (C_4)
Archival Caste (C_5)

Figure 9: In-store compute caste

complexity is just added flexibility. A software-defined storage layer with many options can make better choices for applications that need a certain class of storage with a certain level of quality of service.

Eusocial storage can take advantage of existing tools such as software-defined networking to centralize management of data path and management into simple interfaces, while maintaining storage disaggregation. As an example, we have a set of eusocial castes which are connected via a software-defined, fully connected crossbar, meaning that the latency of requests from any device in any caste to any other device is similar. The actual physical architecture connecting all of these devices may differ, but the way they are advertised to client applications is this simple crossbar.

This means two things to two different groups of individuals. First, the application team enjoys a remarkable degree of freedom in choosing the type of eusocial caste their data should live and act in, without having to consider the underlying system architecture. Second, because the underlying architecture is obfuscated from the application by this layer of software-defined networking, the system infrastructure group that maintains the various devices participating in the eusocial castes has freedom in architecting the disaggregation of devices so long as they do not violate the higher level QoS guarantees being advertised by a particular eusocial caste to applications. Combined, these two groups of people are happy, and the underlying infrastructure is more efficiently consumed.

East-West Data Management Offloading

Once eusocial devices are disaggregated, the balance of the data management features of eusocial can be delivered. This includes scale access, data movement, data redundancy, data recovery, and caste hopping. These features all require peer-to-peer operations, or east-west data movement.

In-Store Computing

The design of eusocial storage naturally progresses towards "in-store computing," that is, performing computing such as data filtering, transformation, and even more compute-intensive analytics within a storage device (Figure 9). The evolution observed so far shows a slow increase of data management offloading onto storage devices, slowly increasing the requirement for processing in the device (e.g., translation layers in flash and shingled magnetic recording disks). East-west communication among in-store computing devices will enable functionalities such as deduplication, secret sharing, and divergent replication (i.e., each replica has a different layout). We anticipate that due to the increase in cost, in-store computing devices will be separate from the main storage and reside in their own caste.

Because scale out is done within the caste, implementers are free to determine the number of in-store computing eusocial devices, how they are replicated, and, through the use of caste maps, when old data should be moved out. Datacenter uses will focus around big-data processing and search. A caste of IoT eusocial devices deployed at the edge store acquired data and then do first-pass processing before shuttling the data back to the home office castes.

The benefits from in-store computing have been studied by Seagate. While many details could not be made available for this article, some of the results are promising: using benchmarks that include MapReduce, search, and data maintenance tasks, Seagate was able to double storage throughput and reduce host CPU utilization by 15–20% by offloading these tasks to devices capable of in-store computing. Seagate also found evidence that in-store computing can increase uploading speeds. In one case, uploading speeds increased by a factor of 10 over a Hadoop installation with traditional devices. All these results are mainly due to scale-out effects of offloading data-intensive operations to many devices where the data already resides and where data transfers to hosts become unnecessary.

Conclusion

This paper has examined the effects of rapidly growing storage throughput on our computing environments as well as the need for scale environments. Both of these issues are forcing a dramatic change in the roles and responsibilities of components in our systems. Previously, it was desirable to have dumb and cheap devices that a system could program and manage. But as their capacities and speeds grow, it is becoming clear that their management and some data processing belong to the devices themselves. The eusocial concept is a design space that is opening the door to just such a future.

Acknowledgments
This work has been made possible by the Center for Research in Open Source Software at UC Santa Cruz (cross.ucsc.edu), which is funded by a donation from Sage Weil and industry memberships. Industry members include Toshiba Memory America, Inc., Micron Technology, Inc., Seagate Technology LLC, Western Digital Corporation, and Huawei Technologies Co. Ltd. CROSS, founded in September 2015 as a forum for high-impact research with strong industry participation, leverages technology transfer and standardization effects of open-source software communities and educates the next-generation of open-source software leadership among UC Santa Cruz doctoral students.

The setup for the in-store active-disk results are Linux; PLX PCIe switch, PCIe Gen-1, x4; 2 SSD drives Attribution: Nitin Kabra, Rajesh Bhagwat, Sneha Wagh; Seagate.

References

A. Acharya, M. Uysal, and J. Saltz, "Active Disks: Programming Model, Algorithms and Evaluation," *ACM SIGPLAN Notices*, vol. 33, no. 11 (1998), pp. 81–91: http://pages.cs.wisc.edu/~remzi/BrainClass/Wiki/Readings/Systems/p81-acharya.pdf.

E. Riedel, G. A. Gibson, and C. Faloutsos, "Active Storage for Large-Scale Data Mining and Multimedia," in *Proceedings of the 24th International Conference on Very Large Data Bases*, 1998, pp. 62–73: http://www.vldb.org/conf/1998/p062.pdf.

K. Keeton, D. A. Patterson, and J. M. Hellerstein, "A Case for Intelligent Disks (IDISKs)," *SIGMOD Record*, vol. 27, no. 3 (1998), pp. 42–52: http://db.cs.berkeley.edu/papers/sigmodr98-idisk.pdf.

H. Lim, V. Kapoor, C. Wighe, and D. H. Du, "Active Disk File System: A Distributed, Scalable File System," in *Proceedings of the 18th IEEE Symposium on Mass Storage Systems and Technologies*, 2001: https://pdfs.semanticscholar.org/b290/7b9e230a68250781ff4779585b2dc2a144d0.pdf.

N. Golpayegani, S. Prathapan, M. Halem, R. Warmka, B. Wyatt, J. Trantham, and C. Markey, "Bringing MapReduce Closer to Data with Active Drives," Abstract IN21D-0059 presented at 2017 Fall Meeting, AGU.