# Host Side Caching: Solutions and Opportunities*

*Shahram Ghandeharizadeh, Jai Menon, Gary Kotzur, Sujoy Sen, Gaurav Chawla*

Database Laboratory Technical Report 2015-02

Computer Science Department, USC

Los Angeles, California 90089-0781

March 29, 2016

**Abstract**

Host side caches use a form of storage faster than disk and less expensive than DRAM to deliver the speed demanded by data intensive applications. Today, this form of storage is NAND Flash, complementing a disk-based solution. A host side cache may integrate into an existing application seamlessly. This may be realized by using an infrastructure component (such as a storage stack middleware or the operating system) to intercept the application read and write requests for disk pages, populate the flash cache with disk pages, and use the flash to service read and write requests intelligently. This study provides an overview of host side caches, an analysis of its overhead and costs to justify its use, alternative architectures including the use of the emerging Non Volatile Memory (NVM) for the host-side cache, and future research directions. We show results using Dell's host-side caching solution named Fluid Cache. It was built to improve the performance of OLTP workloads. However, our results using a social networking benchmark named BG shows that Fluid Cache also enhances the performance of social networking workloads anywhere from a factor of 3.6 to 18.

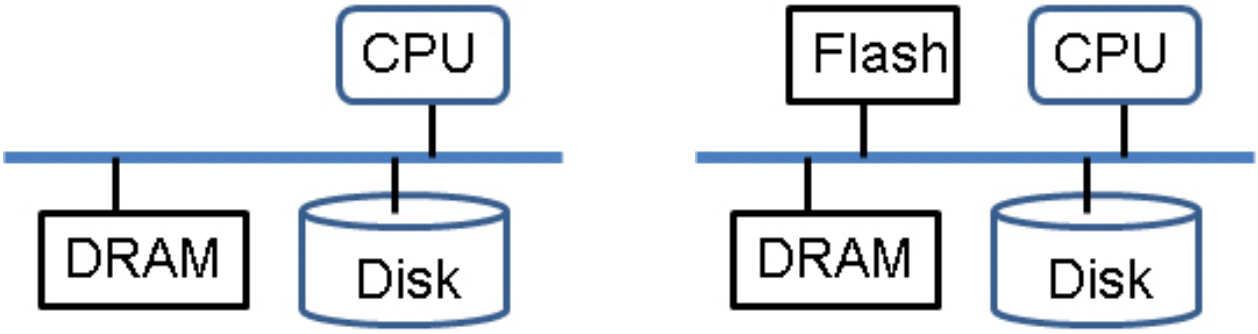
*This work was conducted at Dell Research, Santa Clara, California 54080. In Proceedings of the IEEE 12th International Baltic Conference on Databases and Information Systems (DB&IS), Riga, Latvia, July 2016.

# 1 Introduction

Enterprises deploy *host side caches* to enhance the performance of their data intensive applications. These caches utilize a form of storage faster than disk and cheaper than DRAM to stage disk pages referenced by an application, enhancing performance. Example storage used in host-side cache include today's NAND Flash and the emerging STT-RAM [32], Memristor [51] and PCM [5, 45, 14, 54, 30] as future candidates. See Table 1 for details. An example application may be the server component of a database management system (DBMS) such as Oracle and MongoDB that processes queries and updates. A host side cache might be either managed by the application [25, 41] or deployed seamlessly using a storage stack middleware or the operating system (termed the caching software) [47, 11, 48, 37, 49, 7, 27, 33, 29].

Figure 1.a shows a traditional architecture of an application consisting of a two level hierarchy, DRAM and disk. With this architecture, the application's read and write operations for disk pages might be serviced by the DRAM (a cache hit) or the disk (a cache miss). The application may manage the writing of a dirty block from DRAM to disk. For example, a DBMS server may write log records synchronously to implement the ACID property of transactions.

With Figure 1.b, flash memory is the host side cache and used as an intermediate staging area between DRAM and disk. It is faster than the disk and transparent to the application because the caching software intercepts the disk page read and write requests to service them using the flash. (The caching software may cache at the granularity of a file, however, we assume block based caches in this paper.) The DRAM continues to service the hottest data items. Those data items with a sparser reference pattern reside on the host side cache (flash). Since the host-side cache is realized using technology that is cheaper than DRAM, its size can be much larger than DRAM size. The caching software is aware of both the flash (host side cache) and disk (permanent storage) and may implement different policies to perform the application writes [31]. For example, with a *write-through* policy, the caching software executes writes by writing to both the host side cache and the permanent storage before confirming the write operation to the application. However, with a *write-back* policy, the caching software performs the same write by writing only to the host side cache and confirms




1.a. Traditional. 1.b. Flash as a cache.

Figure 1: A traditional architecture and its extension with Flash as a host side cache.

| | Memristor | FeRAM | PCM | STT-RAM | DRAM | NAND Flash | Disk |
|---|---|---|---|---|---|---|---|
| Read Time (ns) | < 10 | 20-40 | 20-70 | 10-30 | 10-50 | 25,000 | 2-8x$10^6$ |
| Write Time (ns) | 20-30 | 10-65 | 50-500 | 13-95 | 10-50 | 200,000 | 4-8x$10^6$ |
| Retention | > 10 years | ∼10 years | < 10 years | Weeks | < 100 msec | ∼10 years | ∼10 years |
| Energy/bit (pJ)[2] | 0.1-3 | 0.01-1 | 2-100 | 0.1-1 | 2-4 | 10-$10^4$ | $10^6$-$10^7$ |
| 3D capability | Yes | Yes | Yes | No | No | Yes | N/A |

Table 1: Alternative data storage technologies [14, 39].

the write immediately. Either a background process or a cache eviction policy performs the write to the permanent storage.

A host side cache is different than both a Network Attached Storage (NAS) cache and a Key-Value Store (KVS) cache. We describe each in turn and detail how they differ from a host side cache. A NAS cache is an appliance on the network that acts as an intermediary between a NAS and its applications [53, 13, 52]. This appliance is configured with NAND Flash and stages data from a disk based NAS intelligently to expedite application performance. Host side caches reside on the server that processes the application. They provide block level access to implement durable writes required to implement a transaction processing system.

KVS caches manage key-value pairs where a *key* is either a query or a function instance and a *value* is the result of the query or function instance [21, 40, 6, 36, 43, 26]. A popular in-memory KVS is memcached in use by social networking sites such as Facebook [40]. Similar to a host side cache, a KVS cache may be managed by either the application [6, 40] or deployed transparently [22, 20, 36, 21, 43, 26]. The key-value pairs managed by a KVS cache may have different sizes with varying costs [18]. This is in sharp contrast to the fixed size disk pages managed by a host side cache. In order to simplify discussion and without loss of generality, the rest of this paper focuses on host side caches only.

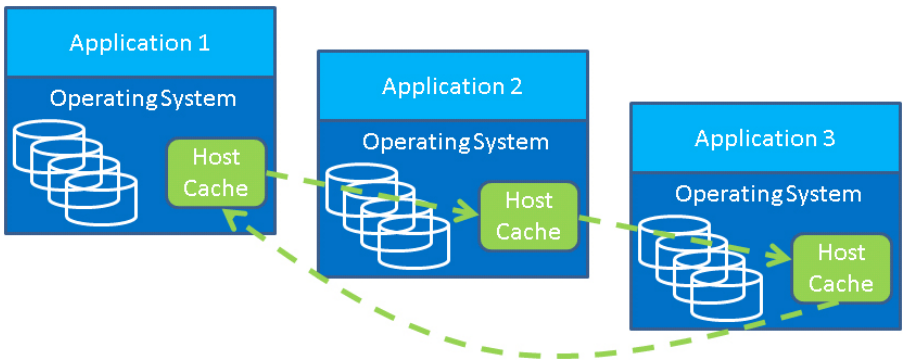
This paper identifies the alternative storage architectures to realize a host side cache, and discusses future research and development directions. The rest of this paper is organized as follows. Section 2 presents several architectures for host side caches. In Section 3, we present a case study of the Dell Fluid Cache solution. Section 4 presents analytical models that quantify the cost and benefits of host side caches, enabling one to explain why today's flash caches enhance system performance. Section 5 presents future extensions and opportunities, including the use of Non-Volatile Memory (NVM). Brief words of conclusion are presented in Section 6.

# 2 Multi-Node Architectures

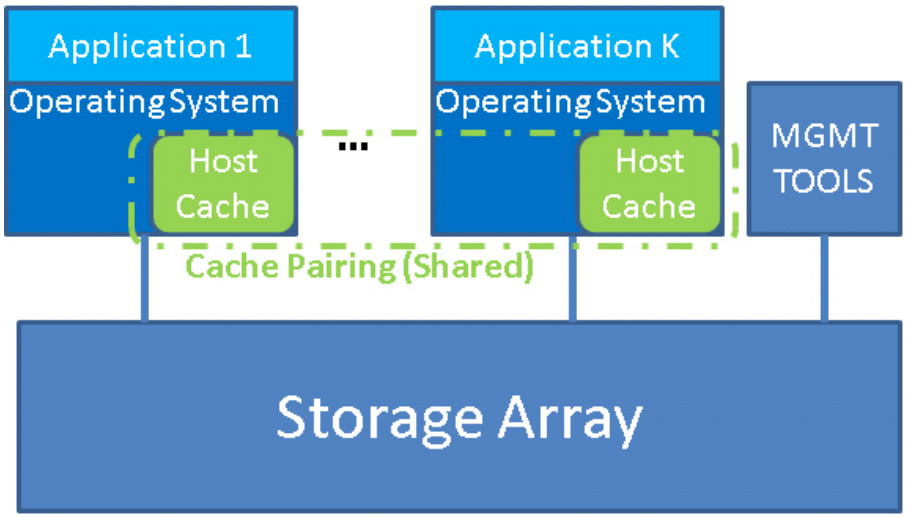
Several commercial architectures of a multi-node implementation of a host-side cache are shown in Figure 2. These are categorized into shared-nothing and shared-disk. In the presence of node failures and disk failures, an architecture must address (1) availability of data on the permanent store and (2) durability of writes with the write back-policy using the host-side cache. Consider each in turn.

With data on the permanent store, a shared-disk architecture may employ multiple host-bus adapters and switches in combination with RAID [42] to enhance availability of data. To realize the same, a shared-nothing architecture may use a variant of techniques used by file systems such as the Google File System [24], database management systems such as Gamma [12, 28, 16], and peer-to-peer data stores such as CAN [46] and Chord [50]. These techniques construct a partition of disk pages, assigning its primary and secondary copies to different nodes. In the presence of failures that render a primary partition unavailable, one of its secondary copies is promoted to be the primary.

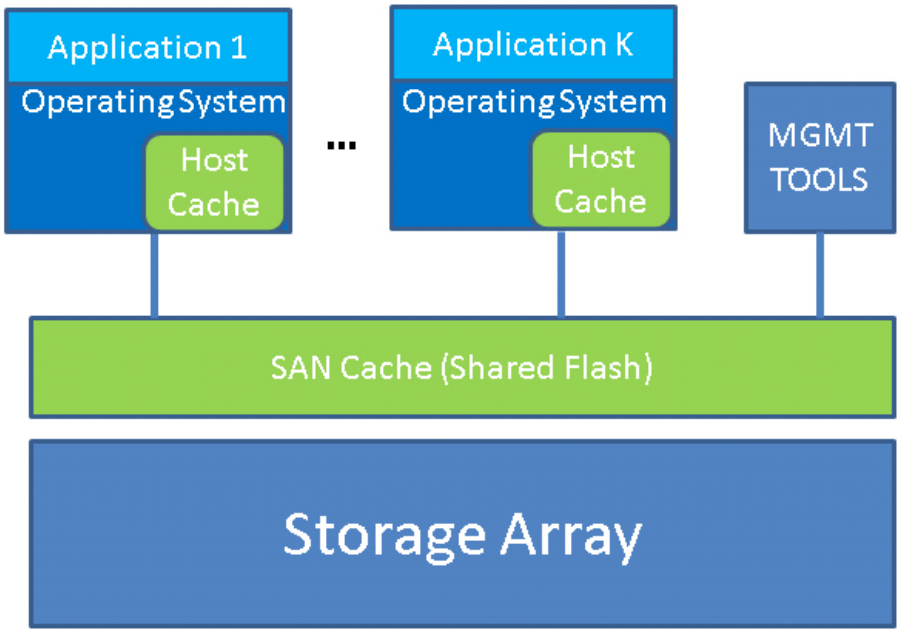
To ensure durability of writes with the write-back policy, architectures of Figure 2 and 2.b may pair the host-side caches of multiple nodes with one another [11, 7, 33, 29], designating the content of one as *primary* and the rest as *secondary*. A write must be performed to both its primary and secondary host-side cached copies synchronously. In the presence of a node failure that renders the dirty primary copy of a disk page on a host-side cache unavailable, the system writes one of its secondary copies to the SAN [11]. With N-1 secondary copies




2.a. A shared-nothing architecture.




2.b.Shared disk with distributed shared flash.




2.c. Shared disk and shared flash.

Figure 2: Alternative architectures.

for a disk page and a maximum of $\delta$ time units for a dirty disk page to be written to the SAN (in the presence of failures), all N replicas must fail during $\delta$ in order for a write to become non-durable. One may minimizie this possibility by either reducing the duration of $\delta$ or increasing the value of N.

The architecture of Figure 2.c assumes both a shared disk and a shared flash (termed SAN Cache). With this architecture, there is no pairing of the host-side caches. Instead, every write to the host-side cache is also written to the SAN Cache synchronously. NetApp presents this architecture in the context of a data center deployment with multiple virtual machines (hosting one or more DBMSs) accessing a hypervisor host that employs a host cache seamlessly [7, 33]. The use of SAN Cache is motivated by the observation that a write to the flash requires the same amount of time as a network hop [7].

# 3 A Case Study: Dell Fluid Cache

Dell Fluid Cache is a leading host-side caching solution for enterprise systems. It implements the architecture of Figure 2.b using a separate, low-latency private cache network dedicated for processing the asynchronous writes to a pairing of the host-side caches. To tolerate switch failures, the architecture may be extended with a second switch. Fluid Cache does not require every server using the SAN to be configured with a flash cache. As long as these servers are connected to the cache network, they may use the caches of the other contributing servers to process requests [15].

One may deploy Fluid Cache across multiple nodes in a variety of configurations. Figure 3 shows one deployment evaluated using a social networking benchmark named BG [1]. This deployment consists of three nodes, each with a disk (iSCSI LUN) and one or more SSDs. It shows MySQL version 5.5 server is deployed on one node. The BG Client that generates requests for this server is deployed on the same node.

Fluid Cache can be configured with either the write-back or the write-through policy [31]. In the write-back mode, writes from the application (i.e., MySQL) are written to the SSDs of two different nodes to enhance the durability of data in the presence of node failures. In the write-through mode, writes from the application are written to both the cache and the

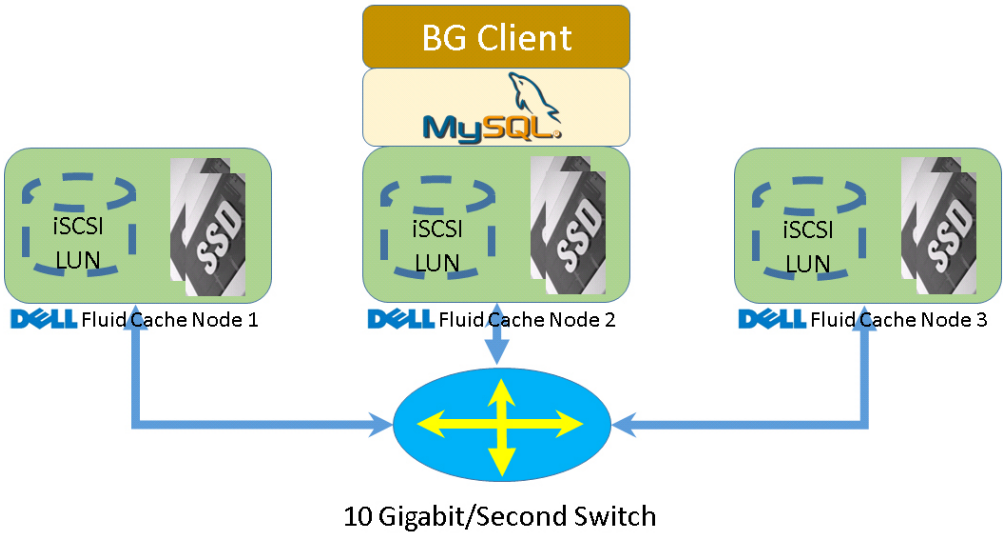



Figure 3: BG benchmark evaluating MySQL with Fluid Cache.

| BG Social Actions | Type | Read-Only | Mix of 90% Read and 10% Write |
|---|---|---|---|
| View Profile (VP) | Read | 100% | 50% |
| List Friends (LF) | Read | 0% | 20% |
| View Friends Requests (VFR) | Read | 0% | 20% |
| Invite Friend (IF) | Write | 0% | 4% |
| Accept Friend Request (AFR) | Write | 0% | 2% |
| Reject Friend Request (RFR) | Write | 0% | 2% |
| Thaw Friendship (TF) | Write | 0% | 2% |

Table 2: Two BG workloads considered in this study.

permanent store (a 145 GB iSCSI LUN in our tests).

When a node is configured with multiple SSD cards, Fluid Cache partitions the disk pages across the available SSDs, harnessing their aggregate bandwidth to service different requests. In our experiments, a node is configured with 2 SSDs each with 16 GB of memory.

With multiple nodes, Fluid Cache may distribute the disk pages across the SSDs of different nodes (termed Proportional) or assign the disk pages to the SSDs local to the node that references these disk pages (termed Client Affinity). In this study, we configure Fluid Cache with the Client Affinity policy.

The BG Client creates a social graph consisting of a fixed number of members. In our experiments, the social graph consists of 10 million members and is approximately 50 GB in

size[1]. A thread emulates a member of the social graph issuing one of the interactive social networking action shown in Table 2. Each emulated member is termed a *socialite*. A socialite issues an action shown in Table 2 with a fixed probability. These actions are categorized into either read or write actions. An example read action is for a socialite to view another member's profile. An example write action is for a socialite to invite another member to be friends. Table 2 shows two different workloads: 1) Read-only consisting of the View Profile action, and 2) A mixed workload with 90% reads and 10% writes. See [1] for the detail specification of each action.

BG quantifies both the service time and the overall processing capability of a solution for processing a workload. The service time is measured with 1 socialite (thread) issuing requests one at a time. It is quantified as the elapsed time from when BG issues an action (that may result in one or two SQL commands to MySQL) to the time the action is processed. We use BG with 16 concurrent socialites (threads) to quantify the number of actions performed by the Fluid Cache deployment. This is the throughput of the system.

A novel feature of BG is its ability to quantify the amount of stale, erroneous, and incorrect (termed unpredictable [2]) data produced by a solution. This is due to the NoSQL movement that may employ weaker forms of consistency to enhance system performance [8]. In all our experiments with the mixed workload, there were no unpredictable data as MySQL implements strong consistency guarantees and the Fluid Cache system preserves the correctness of the application.

For the read-only workload, the choice of write-back or write-through policies has no impact on the observed performance. Fluid Cache enhances the average service time by more than a factor of six with one socialite. It enhances the system throughput by more than a factor of 18 with 16 socialites.

For the mixed read/write workload, Fluid Cache configured with the write-back policy enhances service time by a factor of 3.8. It enhances the throughput of the system by a factor of 6.4 with 16 socialites. The write-back policy outperforms the write-through policy by 14% for service time and 27% for throughput.

Read-only workloads are improved more than mixed workloads with Fluid Cache. This is

[1]With Fluid Cache configured using the Client Affinity policy, the total size of host-side cache is 32 GB.

| Term | Definition |
|---|---|
| $r$ | DRAM hit rate with the host side cache. |
| $p$ | Hit rate of the host side cache. |
| $q$ | Hit rate for the memory footprint of the host side cache. |
| $o$ | Number of bytes per disk page frame to implement the host side cache. |
| $n$ | Number of disk page frames supported by the host side cache. |
| $L_{Device}$ | Latency to retrieve a page from a device such as DRAM, Flash, Disk. |
| $B_{Device}$ | Bandwidth to transfer a page from a device such as DRAM, Flash, Disk. |

Table 3: List of terms and their definitions.

primarily due to the higher SSD bandwidth for reads versus for writes. Using the FIO v2.1.7 microbenchmark, the read bandwidth observed with 1 thread is twice the write bandwidth. With 10 threads, the read bandwidth is approximately 10 times higher.

# 4 Memory Overhead

An implementation of a host side cache requires in-memory meta-data to identify which disk page reads should be serviced by the host side cache. The meta-data should provide sufficient information to process the disk page read in O(1) look up with no false positives, resulting in one flash I/O. This in-memory meta-data is the foot print of the host side cache. It is the overhead of using a host side cache because the system could have used this memory to process application requests instead of implementing the host side cache. For example, Mercury [7] requires 22 byte of in-memory meta-data for each 4 KB disk page frame in the flash cache. Hence, it consumes 2.75 GB of memory for a 512 GB flash cache formatted as 4 KB pages, a 0.5% overhead.

This section presents simple analytical models to quantify the cost of this overhead and the benefits of the host side cache. These models can be used to configure a NVM to maximize the impact of host side caches, see Section 5.

Let $p$ denote the hit rate observed with a host side cache, e.g., 512 GB of flash cache. And, let $q$ denote the hit rate observed with the memory overhead of a host side cache had the system not implemented the host side cache, e.g., 2.75 GB of DRAM. The following proposition relates these two metrics with the physical properties of DRAM, host side cache, and disk. These physical properties include latency, bandwidth, and service time.

**Proposition 1**

a. *For a read only workload, the ratio of p to q ($\frac{p}{q}$) must be greater than $\frac{L_{DRAM}-L_{Disk}}{L_{Flash}-L_{Disk}}$ in order for the flash host side cache to enhance the observed average latency; where L is the latency to read a disk page from DRAM ($L_{DRAM}$), flash ($L_{Flash}$), and disk ($L_{Disk}$).*

b. *The expression $\frac{L_{DRAM}-L_{Disk}}{L_{Flash}-L_{Disk}}$ results in a value slightly greater than one. This is because $L_{Disk}$ is significantly larger than the latency of DRAM and flash and the latency of DRAM is faster than that of flash, see Table 1. In the example below, $\frac{L_{DRAM}-L_{Disk}}{L_{Flash}-L_{Disk}}$ equals 1.00628.*

c. *In order for the overhead of the host side cache to outweigh its benefits, $\frac{p}{q}$ must be a value smaller than one, requiring q to exceed p.*

d. *q may not exceed p because the pages that fit in the memory foot print of the host side cache are most likely a subset of those that fit in the host side cache itself.*

**Proof:** To prove Proposition 1.a consider the latency with and without a flash cache. With no flash cache, the average latency is:

$$\overline{L_{NoCache}} = ((r+q) \times L_{DRAM}) + ((1-r-q)L_{Disk}) \tag{1}$$

where $r$ is the probability of a page reference observing a hit in DRAM with the host side cache. We add $q$ to $r$ because there is no flash cache and the memory overhead of the host side cache is not incurred.

The average latency with the flash cache is as follows:

$$\overline{L_{Cache}} = (r \times L_{DRAM}) + (p \times L_{Flash}) + ((1-r-p) \times L_{Disk}) \tag{2}$$

Note that the latency associated with flash is incurred for those requests that observe a hit with the host side cache.

One may solve for the break even point when Equation 1 equals Equation 2. This point is realized when $\frac{p}{q} = \frac{L_{DRAM}-L_{Disk}}{L_{Flash}-L_{Disk}}$. When $\frac{p}{q}$ is less than $\frac{L_{DRAM}-L_{Disk}}{L_{Flash}-L_{Disk}}$, the host side cache is not beneficial in reducing the average latency and should be abandoned in favor of using its memory foot print to service application disk pages. Otherwise, the host side cache is beneficial and enhances the application performance. ■

**Example:** As an example, assume $L_{DRAM}$ is 30 nanoseconds, $L_{Flash}$ is 25 microseconds (25 thousand nanoseconds), and $L_{Disk}$ is 4 milliseconds (4 million nanoseconds). The ratio $\frac{L_{DRAM}-L_{Disk}}{L_{Flash}-L_{Disk}}$ equals 1.00628, requiring $\frac{p}{q}$ to be greater than this value in order for the host side cache to enhance the average latency. This means if the DRAM hit rate is 60% ($r$=0.6) and the hit rate attributed to the memory foot print of the flash cache is 20% ($q$=0.2) then the hit rate of the flash cache must exceed 20.13% ($p$ >0.2013) to provide a superior latency. This is feasible because the flash cache is much larger (512 GB) than its memory foot print (2.75 GB) and contains a super set of the disk pages contained by its memory foot print. ■

One may conduct a similar analysis with either the bandwidth provided by the different devices or the page service time (latency plus transfer time of a page from a storage medium using its bandwidth). Note that the observation with one metric (say latency) may not apply to another (say bandwidth). To illustrate assume the bandwidth of disk and flash is 3 and 400 MB/sec, respectively ($B_{Disk}$=3, $B_{Flash}$=400). Moreover, assume the bandwidth of DRAM is limited by the system bus at 12 GB/sec. Based on proposition 1, the break even point is realized when $\frac{p}{q}=\frac{B_{DRAM}-B_{Disk}}{B_{Flash}-B_{Disk}}$. Note that this equation violates Proposition 1.b because the value of $\frac{B_{DRAM}-B_{Disk}}{B_{Flash}-B_{Disk}}$ might be many folds higher than one ($B_{Disk}$ is lower than $B_{Flash}$ and $B_{DRAM}$). Thus, the remaining propositions no longer apply. To demonstrate, consider the hit rates assumed in our example ($r$=0.6 and $p$=0.2), the maximum possible value of $q$ (40%) will not justify the use of a host side flash cache as it reduces the observed bandwidth by 30%. The value of $q$ must exceed 618.9% in order for a host side cache to break even. Cache hit rates higher than 100% are not feasible, causing the host side cache to be too costly.

The value of $\frac{B_{DRAM}-B_{Disk}}{B_{Flash}-B_{Disk}}$ becomes negative when $B_{Disk}$ is greater than $B_{Flash}$, rendering the host side cache undesirable from a bandwidth perspective. For example, with an enterprise class disk array such as a Dell Compellent SC2020 providing a maximum bandwidth of 6 GB/sec with 256 KByte disk pages and a flash cache providing a bandwidth in the order of a few hundred MB/sec, a host side cache is not beneficial to improve bandwidth.

With service time defined as device latency, i.e., $L_{Device}$, plus the transfer time of a page from that device, i.e., $\frac{Page\ Size}{B_{Device}}$, a host side cache is once again justified. This is because the

disk latency ($L_{Disk}$) is a significant contributing factor.

As described in Section 5, these analytical models can be used to configure a NVM to maximize the impact of a host side cache.

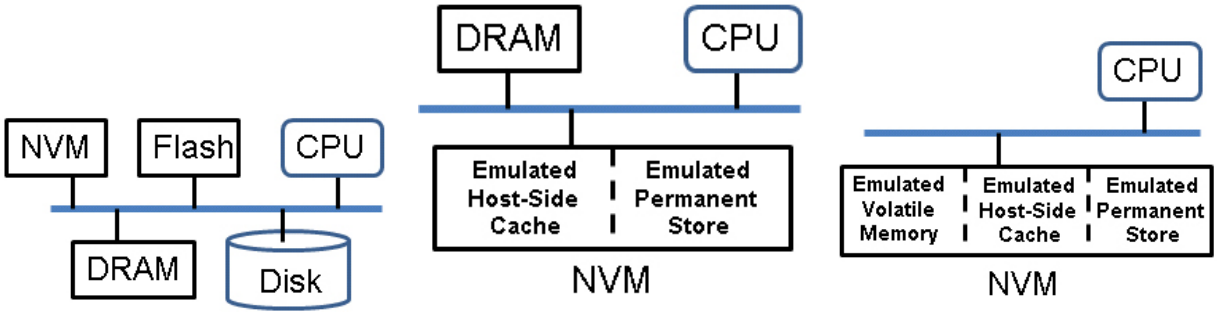
# 5 Opportunities

The host side cache offers several opportunities for additional research and development. We categorize these into effective monitoring and management tools, extended memory hierarchy, and novel software designs. Below, we describe these in turn.

**Effective Monitoring and Management Tools:** The host side cache requires effective management tools to enable an administrator to identify bottlenecks and resolve them effectively. Such tools may report on cache hit rates, amount of data served from local and remote caches, and the I/O rates for the cache and the disk (for reads and writes). The tools may enable the administrator to turn certain hints on and off to evaluate their impact on system performance.

It is paramount for the tools to enable an administrator to shut down a host cache instance gracefully. Ideally, the load of this host cache should be divided across the other host cache instances in a manner that avoids formation of hot spots and bottlenecks. Moreover, it might be more effective for the popular content of this host cache to be migrated to the other host caches to minimize the adverse impact on performance attributed to switching from a warm cache to a cold one.

This tool that shuts down a host-side cache works differently for the two architectural alternatives. For example, with the shared-nothing architecture of Figure 2.a, the tool must migrate permanent (disk) data from one node to the others. With a shared-disk architecture, the tool may migrate the popular data items from the host-side cache to warm-up the destination host caches prior to dismounting virtual disks (LUNs) from a source node. Subsequently, it mounts the LUNs of the source node on to one or more destination nodes [17, 3, 35].




1.a. NVM as a layer. 1.b. DRAM with NVM. 1.c. NVM only.

Figure 4: Alternative NVM architectures.

**Extended Memory Hierarchy:** Non-volatile memory (NVM) provides an exciting opportunity to further improve the performance uplift from host side caches. At the time of this writing, NVM is a broad class of technologies including STT-RAM [32], Memristors [51] and Phase-Change Memory [45] (PCM). They are faster than flash and some provide a byte addressable interface similar to DRAM, see Table 1. They are anticipated to provide latencies either comparable to DRAM or an order of magnitude slower depending on their configuration. There is general consensus that NVM will provide higher storage densities than today's DRAM and will be cheaper than DRAM, though more expensive than Flash initially.

Figure 4.a shows an architecture consisting of DRAM, NVM, flash, and Disk. Today's host side caches might be adapted to use this architecture with (a) flash as the host side cache and using NVM as an extension of volatile memory (DRAM), or (b) NVM as the host-side cache and flash as an extension of permanent storage (disk). Another possibility is to implement a host-side cache that manages both NVM and flash separately, caching data across these devices intelligently. A challenge is how to benefit from the byte-addressability features of NVM while the flash (and the application) interface is block-based. Another consideration is the DRAM requirement of such a design, see discussions of Section 4. A key research question is what are the tradeoffs associated with these alternative designs? In particular, the additional complexity of the last design must be justified by significant performance benefits when compared with the other two possibilities.

Figures 4.b and 4.c are specific to NVMs that are envisioned to be configurable into $N$ memory banks with varying performance and non-volatility characteristics. In Figure 4.b,

NVM is used as a host-side cache, and it is also separately used to emulate permanent store (disk). The motivation for slowing down a portion of NVM to emulate a memory bank with a higher retention that serves as the permanent store is to enhance the energy efficiency of a solution (battery life, see Table 1) or its physical size by eliminating interfaces and devices. Most likely, extremes such as non-volatility in the order of tens of years (disk) is an overkill for most applications. It might be more appropriate to emulate segments providing storage with latencies several folds slower than DRAM with retention in the order of a few years. An open research direction is the design and implementation of algorithms that decide the characteristics (size, volatility, performance) of different memory banks based on the requirements of an application.

In the absence of actual NVMs, one may investigate alternative designs using simlation studies. The architectures of Figures 4.b and 4.c may be simulated by coupling DRAM with Flash [38] (non-volatile DIMM) or by forcing a portion of DRAM to behave similar to NVM [44]. We plan to use these in a variety of studies to investigate designs that may enhance performance during normal mode of operation, increase availability of data in the presence of failures, expedite recovery after a failure with the content of the cache intact, and improve both vertical and horizontal scalability of a shared-nothing and a shared-disk deployment.

**Novel Software Designs:** Today's host side caches are either fully managed by the application or completely transparent. A hybrid may consistent of a transparent cache that accepts hints from an application to enable it to prioritize asynchronous writes to the persistent store. For example, it might be useful for a DBMS to identify its log file to enable the host cache to flush these to the persistent store more aggressively as the likelihood of their repeated reference is very low during the normal mode of operation. This improves the cache hit rate of the host cache to enhance performance. Such a hybrid requires an effective abstraction that is exposed by the caching layer (a storage stack middleware, OS, Hypervisor) and exercised by an application.

A challenge is how to recover from a short lived failure or an intermittent network connectivity event between a host cache instance and its peers without clearing the content of

the cache. The time required to warm up a flash cache might be in the order of hours due to its high capacity. As an example, a 300 Gigabyte flash cache required more than 10 hours to reach its maximum hit rate for an enterprise[2] workload [7]. Once warmed up, it is desirable to maintain the content of the cache intact. After a short failure, only a small subset of the cache content might be invalid. An opportunity is to identify the invalid content and discard them either lazily or eagerly. Once a new host cache instance is deployed, a deployment may warm up its cache aggressively using the caches of peer instances [40]. Alternatively, it may monitor the application workload, predict the popular disk pages, and cache them [29].

Data consistency is another challenge faced by host side caches. Consider a scenario where an application modifies a disk page in the host flash cache which is not yet written to the shared disk (a write-back cache policy). If the SAN controller attempts to snapshot or clone a file that contains this dirty disk page, the file will not contain a consistent set of disk pages since the modified disk page will not be visible to the storage controller. Similarly, if the storage controller either reverts to an older snapshot of a file or receives asynchronous mirror changes from a remote primary, it will not be visible to a host flash cache with conflicting data. One may adapt frameworks such as IQ [23] to address this consistency challenge.

In general, today's data stores are not positioned to take advantage of the byte addressability of NVMs [10, 9]. Most data stores manage blocks of data with either a row or a column organization of records/documents/key-values [8]. Currently, an object-relational mapping framework such as Hibernate [4, 34, 19] might be best suited for NVM as they enable a software designer to identify classes (data structures) that should persist. One may envision adapters for Hibernate to support these data structures effectively using the NVM byte addressability characteristics. However, the time for a complete re-design and re-write of complex applications such as a DBMS is waiting to be seized [10].

# 6 Conclusions

Host side caches employ a form of storage faster than disk and less expensive than DRAM, e.g., flash or Non-Volatile Memory, to enhance the performance of data intensive applications.

[2]A pair of 256 Gigabyte flash caches required about 3 hours to warmup with an OLTP workload [47].

They complement disk-based solutions and integrate into an application seamlessly. This paper surveys the different architectures to manage these caches, and opportunities for their future extensions. These extensions are in the form of monitoring tools, extended memory hierarchy, and novel software designs. The KVS caches [21] (see Section 1) are in synergy with host side caches and benefit from similar extensions. We believe both caches will co-exist in future offerings. This is demonstrated by social networking sites such as Facebook that employ both flashcache [37] and memcached [40] to service millions of requests per second.

# References


[1] Barahmand, S., and Ghandeharizadeh, S. BG: A Benchmark to Evaluate Interactive Social Networking Actions. *CIDR* (January 2013).

[2] Barahmand, S., and Ghandeharizadeh, S. Benchmarking Correctness of Operations in Big Data Applications. In *IEEE 22nd International Symposium on Modelling, Analysis & Simulation of Computer and Telecommunication Systems, MASCOTS* (2014).

[3] Barker, S., Chi, Y., Moon, H., Hacigümüs, H., and Shenoy, P. J. "Cut Me Some Slack": Latency-Aware Live Migration for Databases. In *EDBT* (2012), pp. 432–443.

[4] Biswas, R., and Ort, E. The Java Persistence API - A Simpler Programming Model for Entity Persistence, May 2006. http://java.sun.com/developer/technicalArticles/J2EE/jpa.

[5] Breitwisch, M. J. Phase change memory. In *Interconnect Technology Conference* (2008), pp. 219–221.

[6] Bronson, N., Amsden, Z., Cabrera, G., Chakka, P., Dimov, P., Ding, H., Ferris, J., Giardullo, A., Kulkarni, S., Li, H., Marchukov, M., Petrov, D., Puzar, L., Song, Y. J., and Venkataramani, V. TAO: Facebook's Distributed Data Store for the Social Graph. In *USENIX Annual Technical Conference (USENIX ATC 13)* (San Jose, CA, 2013), pp. 49–60.

[7] Byan, S., Lentini, J., Madan, A., Pabon, L., Condict, M., Kimmel, J., Kleiman, S., Small, C., and Storer, M. Mercury: Host-side Flash Caching for the Data Center. In *IEEE Symposium on Mass Storage Systems and Technologies (MSST)* (2012).

[8] Cattell, R. Scalable SQL and NoSQL Data Stores. *SIGMOD Rec. 39* (May 2011), 12–27.

[9] Chang, J., Ranganathan, P., Mudge, T., Roberts, D., Shah, M. A., and Lim, K. T. A Limits Study of Benefits from Nanostore-Based Future Data-Centric System Architectures. In *CF* (2012), p. 3342.

[10] Coburn, J., Caulfield, A. M., Akel, A., Grupp, L. M., Gupta, R. K., Jhala, R., and Swanson, S. NV-Heaps: Making Persistent Objects Fast and Safe with Next-Generation, Non-Volatile Memories. In *ASPLOS* (2011), pp. 105–118.

[11] Dell. Dell Fluid Cache for Storage Area Networks, http://www.dell.com/learn/us/en/04/solutions/fluid-cache-san, 2014.

[12] DeWitt, D., Ghandeharizadeh, S., Schneider, D., Bricker, A., Hsiao, H., and Rasmussen, R. The Gamma Database Machine Project. *IEEE Transactions on Knowledge and Data Engineering 1*, 2 (March 1990).

[13] EMC. Migrating Data From an EMC Celerra or VNS Array to a VNX2 Using VNX Replicator. EMC White Paper, 2014.

[14] Fink, M. Beyond DRAM and Flash, Part 2: New Memory Technology for the Data Deluge, HP Next, http://www8.hp.com/hpnext/posts/beyond-dram-and-flash-part-2-new-memory-technology-data-deluge.vcb6vrbcfe8, 2014.

[15] Gagrani, K., and Makransky, K. Turbocharging Application Response, Dell Power Solutions, Issue 2, 2014 http://www.dell.com/learn/us/en/555/power/ps2q14-20140344-makransky.

[16] Ghandeharizadeh, S., and DeWitt, D. J. Hybrid-Range Partitioning Strategy: A New Declustering Strategy for Multiprocessor Database Machines. In *VLDB* (1990).

[17] Ghandeharizadeh, S., Goodney, A., Sharma, C., Bissell, C., Carino, F., Nannapaneni, N., Wergeles, A., and Whitcomb, A. Taming the Storage Dragon: The Adventures of HoTMaN. In *SIGMOD* (2009), pp. 925–930.

[18] Ghandeharizadeh, S., Irani, S., Lam, J., and Yap, J. CAMP: A Cost Adaptive Multi-Queue Eviction Policy for Key-Value Stores. *Middleware* (2014).

[19] Ghandeharizadeh, S., and Mutha, A. An Evaluation of the Hibernate Object-Relational Mapping for Processing Interactive Social Networking Actions. In *The 16th International Conference on Information Integration and Web-based Applications and Services* (2014).

[20] Ghandeharizadeh, S., and Yap, J. Gumball: A Race Condition Prevention Technique for Cache Augmented SQL Database Management Systems. In *ACM SIGMOD DBSocial Workshop* (2012).

[21] Ghandeharizadeh, S., and Yap, J. Cache Augmented Database Management Systems. In *ACM SIGMOD DBSocial Workshop* (June 2013).

[22] Ghandeharizadeh, S., Yap, J., and Barahmand, S. COSAR-CQN: An Application Transparent Approach to Cache Consistency. In *International Conference On Software Engineering and Data Engineering* (2012).

[23] Ghandeharizadeh, S., Yap, J., and Nguyen, H. Strong Consistency in Cache Augmented SQL Systems. *Middleware* (2014).

[24] Ghemawat, S., Gobioff, H., and Leung, S. The Google File System. In *SOSP '03: Proceedings of nineteenth ACM SIGOPS symposium on Operating systems principles* (2003), ACM Press.

[25] Graefe, G. The Five-Minute Rule Twenty Years Later, and How Flash Memory Changes the Rules. In *DaMoN* (2007), p. 6.

[26] Gupta, P., Zeldovich, N., and Madden, S. A Trigger-Based Middleware Cache for ORMs. In *Middleware* (2011).

[27] Holland, D. A., Angelino, E., Wald, G., and Seltzer, M. I. Flash Caching on the Storage Client. In *USENIXATC* (2013).

[28] Hsiao, H., and DeWitt, D. J. A Performance Study of Three High Availability Data Replication Strategies. *Distrib. Parallel Databases 1*, 1 (January 1993).

[29] Kim, H., Koltsidas, I., Ioannou, N., Seshadri, S., Muench, P., Dickey, C., and Chiu, L. Flash-Conscious Cache Population for Enterprise Database Workloads. In *Fifth International Workshop on Accelerating Data Management Systems Using Mod-*

*ern Processor and Storage Architectures* (2014).

[30] Kim, H., Seshadri, S., Dickey, C. L., and Chiu, L. Evaluating Phase Change Memory for Enterprise Storage Systems: A Study of Caching and Tiering Approaches. In *Proceedings of the 12th USENIX Conference on File and Storage Technologies (FAST 14)* (2014).

[31] Koller, R., Marmol, L., Rangaswami, R., Sundararaman, S., Talagala, N., and Zhao, M. Write Policies for Host-side Flash Caches. In *Presented as part of the 11th USENIX Conference on File and Storage Technologies (FAST 13)* (2013).

[32] Kultursay, E., Kandemir, M. T., Sivasubramaniam, A., and Mutlu, O. Evaluating STT-RAM as an Energy-Efficient Main Memory Alternative. In *IEEE ISPASS* (2013), pp. 256–267.

[33] Liu, D., Tai, J., Lo, J., Mi, N., and Zhu, X. VFRM: Flash Resource Manager in VMware ESX Server. In *IEEE Network Operations and Management Symposium* (2014).

[34] Marquez, A., Zigman, J. N., and Blackburn, S. Fast portable orthogonally persistent java. *Softw., Pract. Exper. 30*, 4 (2000), 449–479.

[35] Michael, N., and Shen, Y. Downtime-Free Live Migration in a Multitenant Database. In *TPC Technical Conference* (2014).

[36] Mitra LLC. KOSAR, http://kosarsql.com 2014.

[37] Mituzas, D. Flashcache at Facebook: From 2010 to 2013 and Beyond, https://www.facebook.com/notes/facebook-engineering/flashcache-at-facebook-from-2010-to-2013-and-beyond/10151725297413920, 2010.

[38] Mogul, J. C., Argollo, E., Shah, M. A., and Faraboschi, P. Operating System Support for NVM+DRAM Hybrid Main Memory. In *HotOS* (2009).

[39] Muller, C., Courtade, L., Turquat, C., Goux, L., and Wouters, D. Reliability of Three-Dimensional Ferroelectric Capacitor Memory-Like Arrays Simultaneoulsy Submitted to X-Rays and Electrical Stresses. In *Non-Volatile Memory Technology Symposium* (2006).

[40] Nishtala, R., Fugal, H., Grimm, S., Kwiatkowski, M., Lee, H., Li, H. C., McElroy, R., Paleczny, M., Peek, D., Saab, P., Stafford, D., Tung, T.,

and Venkataramani, V. Scaling Memcache at Facebook. In *NSDI* (Berkeley, CA, 2013), USENIX, pp. 385–398.

[41] Oracle Inc. Oracle Database Smart Flash Cache, 2010.

[42] Patterson, D. A., Gibson, G., and Katz, R. H. A Case for Redundant Arrays of Inexpensive Disks (RAID). In *Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data* (1988), SIGMOD '88, pp. 109–116.

[43] Ports, D. R. K., Clements, A. T., Zhang, I., Madden, S., and Liskov, B. Transactional Consistency and Automatic Management in an Application Data Cache. In *OSDI* (October 2010), USENIX.

[44] Rao, D. S., Kumar, S., Keshavamurthy, A., Lantz, P., Reddy, D., Sankaran, R., and Jackson, J. System Software for Persistent Memory. In *Ninth Eurosys Conference* (2014), p. 15.

[45] Raoux, S., Burr, G., Breitwisch, M., Rettner, C., Chen, Y., Shelby, R., Salinga, M., Krebs, D., Chen, S.-H., Lung, H.-L., and Lam, C. Phase-Change Random Access Memory: A Scalable Technology. *IBM Journal of Research and Development 52*, 4.5 (2008), 465–479.

[46] Ratnasamy, S., Francis, P., Handley, M., Karp, R., and Schenker, S. A Scalable Content-Addressable Network. In *Proceedings of the ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications* (Aug. 2001), pp. 161–172.

[47] S. Daniel and S. Jafri. Using NetApp Flash Cache (PAM II) in Online Transaction Processing. NetApp White Paper, 2009.

[48] Stearns, W., and Overstreet, K. Bcache: Caching Beyond Just RAM. https://lwn.net/Articles/394672/,http://bcache.evilpiepirate.org/, 2010.

[49] STEC. EnhanceIO SSD Caching Software, https://github.com/stec-inc/EnhanceIO, 2012.

[50] Stoica, I., Morris, R., Karger, D., Kaashoek, M., and Balakrishnan, H. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In *ACM SIGCOMM* (San Diego, California, Aug. 2001), pp. 149–160.

[51] Strukov, D. B., Snider, G. S., Stewart, D. R., and Williams, R. S. The

Missing Memristor Found. *Nature 7191* (2008), 80–83.

[52] Tabor, J. Avere Architecture for Cloud NAS. Avere White Paper, 2014.

[53] Trammell, J. CacheIQ: Automatic Storage Tiering in the Age of Big Data. In *Flash Memory Summit 2012 Proceedings* (2012).

[54] Vučinić, D., Wang, Q., Guyot, C., Mateescu, R., Blagojević, F., Franca-Neto, L., Moal, D. L., Bunker, T., Xu, J., Swanson, S., and Bandić, Z. DC Express: Shortest Latency Protocol for Reading Phase Change Memory over PCI Express. In *Proceedings of the 12th USENIX Conference on File and Storage Technologies (FAST 14)* (2014).