BEST PRACTICES IN UNDERSTANDING AND USING FLASH

Paul Brant
Sr. Technical Education Consultant
EMC Corporation
Paul.Brant@emc.com
# Table of Contents

**Table of Contents**

Table of Contents ........................................................................................................... 2

**Table of Equations** ..................................................................................................... 5

**Introduction** ................................................................................................................ 6

Flash is not a “Flash in the pan” .................................................................................. 7

It’s all about the application but in the Data too ......................................................... 8

Flash Back to the Basics - The Flash Stack ............................................................... 10

Storage Classes ............................................................................................................. 12

The state of Persistent Storage .................................................................................... 12

  The spinning Disk – The Dilemma of the Trilemma ............................................... 12

  Here comes the Flash Mob ..................................................................................... 14

The economics of Flash .............................................................................................. 15

  The cloud is getting Flashy .................................................................................... 15

  A Dropbox in the Bucket ..................................................................................... 16

**Flash Dancing with the Server** ............................................................................... 17

Applications .................................................................................................................. 17

  Data Centricity ....................................................................................................... 17

High Performance Computing ..................................................................................... 19

  InfiniBand MPI Clusters .................................................................................... 19

  Storage and High Performance Computing ....................................................... 20

Flash Standards ........................................................................................................... 22

**Flash Dancing with Storage** ................................................................................ 23

Flash Personalities ....................................................................................................... 23

The Blender effect ......................................................................................................... 24

  Best Practice - Consider what the I/O Blender effect has in your environment .... 24

  Best Practice - Understand the Blender effect on trending apps .................. 26

  Best Practice - Utilization of storage offloading functions ............................. 26

  Best Practice - Be cognizant of LUN Alignment ............................................. 26

  Best Practice - Consider the workload that is being placed on the SDD storage system AND what will be placed on the system in the future ......................... 26
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Best Practice - Utilize expandable and extensible storage systems</td>
<td>26</td>
</tr>
<tr>
<td>Best Practice - Utilize SAN storage that is designed/optimized for virtualization</td>
<td>27</td>
</tr>
<tr>
<td>Best Practice - Utilize an up-to-date infrastructure</td>
<td>27</td>
</tr>
<tr>
<td><strong>Flash Dancing with the Network</strong></td>
<td>27</td>
</tr>
<tr>
<td>Data-Intensive applications</td>
<td>28</td>
</tr>
<tr>
<td>Transports</td>
<td>28</td>
</tr>
<tr>
<td>InfiniBand</td>
<td>29</td>
</tr>
<tr>
<td><strong>Best Practice</strong> - Consider utilizing InfiniBand’s inherent messaging service to enhance performance</td>
<td>29</td>
</tr>
<tr>
<td><strong>Best Practice</strong> - Consider InfiniBand solutions in the Enterprise</td>
<td>31</td>
</tr>
<tr>
<td><strong>Best Practice</strong> - Utilize InfiniBand to increase application performance processing</td>
<td>32</td>
</tr>
<tr>
<td><strong>Best Practice</strong> - Leverage InfiniBand Messaging Queue Architecture</td>
<td>33</td>
</tr>
<tr>
<td>Best Practice - Utilize an open API for Flash appliance related transports</td>
<td>35</td>
</tr>
<tr>
<td>FCoE</td>
<td>37</td>
</tr>
<tr>
<td>Network Accelerators (RDMA)</td>
<td>37</td>
</tr>
<tr>
<td>iWarp</td>
<td>38</td>
</tr>
<tr>
<td>RoCE</td>
<td>39</td>
</tr>
<tr>
<td><strong>Flash Technology Considerations</strong></td>
<td>40</td>
</tr>
<tr>
<td>Write Amplification</td>
<td>41</td>
</tr>
<tr>
<td><strong>Best Practice</strong> - Consider Flash’s limited write cycles</td>
<td>41</td>
</tr>
<tr>
<td><strong>Best Practice</strong> - Consider utilizing unmapping features in Flash drives</td>
<td>43</td>
</tr>
<tr>
<td>Flash Device I/O Management Overview</td>
<td>44</td>
</tr>
<tr>
<td>Flash Transaction Layer considerations</td>
<td>45</td>
</tr>
<tr>
<td><strong>Best Practice</strong> - Consider application characteristics in Flash Architectures</td>
<td>46</td>
</tr>
<tr>
<td><strong>Best Practice</strong> - Consider technologies changing how memory is committed</td>
<td>47</td>
</tr>
<tr>
<td>Hot Data vs. Cold Data management in Flash</td>
<td>48</td>
</tr>
<tr>
<td><strong>Best Practice</strong> - Consider Recency in FTL designs</td>
<td>49</td>
</tr>
<tr>
<td><strong>Best Practice</strong> - Consider utilizing hot data identification scheme based on multiple bloom filters</td>
<td>50</td>
</tr>
<tr>
<td>SSD Vector Interfaces</td>
<td>50</td>
</tr>
<tr>
<td><strong>Best Practice</strong> - Evaluate vector interfaces to reduce the software-I/O gap</td>
<td>50</td>
</tr>
</tbody>
</table>
Flash is changing RAID ................................................................. 51

Best Practice - Consider Forward Error Correction (FEC) codes when designing FLASH arrays ................................................................. 51
Storage Class Memories (SCM) .......................................................... 52

Flash in the Data Center – How does it all fit? .......................................... 54

The dark side of Flash ......................................................................... 58

Best Practice - Consider the various design principles previously discussed to address Flash warts ........................................................................ 59

Conclusion ......................................................................................... 61

Appendix A – How Flash fits cheat sheet .............................................. 62

Appendix B – References ..................................................................... 63

Author’s Biography ............................................................................. 65

Table of Figures

Figure 1: Changing Application workload vectors .................................. 6
Figure 2: Solid State Disks Taking the Lead in IT Mindshare .................. 7
Figure 3: IT transformation Enablers in the infrastructure stack ............. 7
Figure 4: The Server / Storage Performance Gap ................................... 8
Figure 5: Moving to a data-centric world ............................................. 9
Figure 6: Data Store proximity vs. Capacity ......................................... 10
Figure 7 - Shared File System ............................................................... 20
Figure 8 - Parallel System .................................................................... 22
Figure 9: Traditional Stack and Flash Translation Layer (FTL) Storage Stack ................................................................. 23
Figure 10 - InfiniBand Logical Channel Architecture ........................... 29
Figure 11: InfiniBand Software Stack .................................................. 31
Figure 12: Traditional I/O stack buffer transfer ..................................... 33
Figure 13: Open Fabrics Enterprise Distribution OFED Stack ............... 36
Figure 14: Basic FLASH gate Architecture .......................................... 40
Figure 15: Wear leveling and Garbage collection .................................. 41
Figure 16: Write Amplification Factor .................................................. 43
Figure 17 - Memory Commitment Architectures .................................. 47
Figure 18: Hot and Cold Identification ................................................ 49
Figure 19: SCM Design criteria triangle ............................................. 51
Table of Equations

Equation 1 – Write Amplification .................................................................43
Equation 2 – Write Amplification Factor ......................................................43

Disclaimer: The views, processes or methodologies published in this article are those of the author. They do not necessarily reflect EMC Corporation’s views, processes or methodologies.
Introduction

The amount of data being created is exploding, growing faster than could ever have been imagined. Data from richer sensors, digitization of online content, and new applications like Twitter, search, etc., will exacerbate data growth rates. Indeed, it is estimated that only 5% of the world’s online data has been made online so far. This growth in data is leading to a corresponding growth in data-centric applications that operate on data in diverse ways (capture, classify, analyze, process, archive, etc.). These workloads typically operate at larger scale (hundreds of thousands of servers) and on more diverse data (e.g. structured, unstructured, rich media) with I/O intensive, often random, data access patterns and limited locality. In addition, these workloads have demanded major changes in the software stack, targeted at increased scalability and commodity hardware.

Emerging data center workloads have diverse requirements in multiple dimensions or vectors, shown in Figure 1, and navigating this disk space/response time continuum is not easy.

For example, some applications impose a response time constraint, e.g. Web searches, while others may be background tasks, such as data compression and deduplication. Other vectors and dimensions include compute complexity using high performance computing with ultra-fast storage. In addition, I/O intensity is exploding and temporal data access locality is becoming more of an issue as data sets grow. With Big Data, the volume of information to be processed in a single operation, is ballooning. These application requirements significantly influence the best systems architectures out there for performance as outlined.

The advent of solid-state drives (SSD) based on NAND Flash memories is currently revolutionizing the primary storage computer architecture, ranging from notebooks to enterprise

---

storage systems and as previously mentioned, having an impact on ALL of the workload vectors mentioned above. These devices provide random I/O performance and access latency that are orders of magnitude better than that of rotating hard-disk drives (HDD). Moreover, SSDs significantly reduce power consumption, dramatically improve robustness and shock resistance thanks to the absence of moving parts. Energy efficiency will always be a driving factor addressing Sustainability issues.

**Flash is not a “Flash in the pan”**

It can be argued that the major trends happening today in the data center are Big Data, cloud computing, and high performance computing (HPC). However, as shown in Figure 2, SSDs are also here to stay! According to Google trend's database, SSDs are taking the lead in overall interest levels, an indication that Flash in the enterprise is not just a “Flash in the pan”.

![Figure 2: Solid State Disks Taking the Lead in IT Mindshare](image)

As shown in Figure 3, the infrastructure stack, as we know it today, is changing. Today, the focal points of applications (servers), network infrastructures, and storage are being transformed as they relate to the changing application workloads. In addition, the infrastructure stack needs to work together closer than it ever has before. Flash demands it! In order to efficiently use Flash technology, addressing the application, network, and storage in new ways is necessary! Servers that run the apps, the network, and persistent storage is converging, virtualizing, and becoming cloud-enabled as never before.

![Figure 3: IT transformation Enablers in the infrastructure stack](image)
So, one might ask, why is Flash important? Aren’t SSDs just a faster hard disk? **Consider that SSDs are 400 times faster than a hard disk! The speed of sound is “only” 250 times faster than walking!** Flash eliminates a key mechanical barrier to scaling computing systems. What applications would benefit drastically? There are many, but the top application use cases are, data-centric applications such as Database Consolidation, Business Analytics, Video Applications, I/O-intensive applications, Desktop Virtualization, Server Virtualization, Test, and Development.

**It’s all about the application but in the Data too**

It’s all about speeding up the app, getting data to the app as quickly as possible. One of the giants on the networking industry, Cisco, coined a phrase “Data Center 3.0”, whose strategy is aimed at supporting new business models such as desktop virtualization and public or private clouds. How do we get these applications to the user and give them the right tools to get the job done right? It can be argued that in order to meet the goals of achieving the next generation data center is through transforming the infrastructure to be more Flash aware. For example, Figure 4 depicts a continued widening gap between server or CPU performance and storage\(^2\). Over a fifteen year period, CPU performance has increased by two million times, while disks only achieved a ten times increase over the same period. Clearly, information data access is more than simply keeping up with CPU performance. Something must be done here.

There is also another trend. The infrastructure as we know it today, in conjunction with the application, is transforming into a “Data Services Engine” as shown in Figure 5. With virtualization of various applications.

---

components of the data center and the massive growth, velocity, and variety of information, not to mention the drive toward geographically distributed volumes of content; it seems that we are moving to a more data-centric world with data service engines emerging. Data service engines of various types including mobile and business focused varieties, will allow this trend will continue to grow and Flash technology will be a major force in that transformation, allowing the app satellite to get closer to the data planet as shown in Figure 5.

![Figure 5: Moving to a data-centric world](image)

Making all of those elements work together, ensuring a worthy user experience, and eventually delivering service level agreements (SLA) seems exceedingly complex, but if done right, the emergence of new ecosystems of data, built around cloud providers like Amazon or Google, or popular APIs such as Twitter will transform what we consider to be the infrastructure to a more service model engine. In effect, applications are transforming and becoming part of the infrastructure, but one thing that is not changing is the information needed to establish business value. Prior to virtualization, the application was a separate and distinct part of the infrastructure and the main or central focal point was the application with a specific vertical use case in mind. The information or data was allocated and used based on the specific needs of the application. That is no longer true. With virtualization, the data center is becoming an information blender and Flash must be an active ingredient.
Flash Back to the Basics - The Flash Stack

So, it's all in the application? Well, the application executes instructions on a server and this is where it all begins. From an application perspective, Figure 6 explains the data store stack in relation to its proximity to the program executing code and the ability to address growing requirements of data volume and application affinity mapping and maps it out for us. Let’s “Flash back” to the basics, for a moment. The CPU and registers are at the core of where work is done. This is where logic combines with the data. Since information can grow to enormous data sets, additional structures are needed to store it.

The next level is referred to as Cache (L1/L2/SRAM). Note that as one goes up this stack, the access time or latency increases noted by the red to blue shift indicating that it will take longer and longer to access the higher-level stack given the proximity—logically or physically—to the CPU. The CPU cache is more complicated than other kinds of caches and thus, CPU caches are divided into

![Figure 6: Data Store proximity vs. Capacity](image)

- **CPU/Registers**: Can’t get any faster than this (Ludicrous speed 😍), ultra expensive
- **L1/L2 SRAM Cache**: Very fast, complex, ~16 transistors per storage cell, High power usage/heat, Cost is expensive
- **DRAM/L3 Cache**: Fast, complexity is low, 1 transistor per cell, relatively low power/heat, Cost on planar card, moderate
- **Server Flash Appliance**: App usage: Virtual Desktop (VDI), Virtual Server, DB Test & Dev, Random I/O
- **All Flash Array**: App usage: HPC, High-Frequency Trading, Low-Latency Workloads
- **Hybrid Array**: App usage: OLTP, Email, Databases, Web 2.0, Skewed I/O Pattern
- **Archive**: App usage: Long term retention, cold data

Application Data Set Workload Index Mapping

Hot - Warm - Cold
two groups, Level 1 and Level 2. The L1 cache is memory built right next to the CPU and it is the CPU’s first access to data. The L2 cache is another memory, but instead of feeding the CPU, this one feeds the L1 cache. L2 caches may be built the same way as the L1 caches, into the CPU, but sometimes it can also be located in another chip or in a Multichip Package Module (MCP). It can also be a completely separate chip.

With some exceptions, L1 and L2 caches are usually static RAM (SRAM) while the memory of the computer is considered dynamic RAM (DRAM) or any kind of variation of DRAM. Some processors use another cache level named L3. The difference between L1, L2, and L3 in some cases is its size. L1 is smaller than L2 and L3. If the data is not found in L1, the data will be scanned in the L2’s bigger cache and if it is not there, access to memory then occurs, making the access much slower.

There are two architectural cache methods; inclusive and exclusive. In some processors, the data stored on the L1 cache will also be present in the L2. This is called inclusive or more technically “strictly inclusive”. The AMD Athlon, for example, uses an exclusive cache pattern so the data will either be available in the L1 or the L2 but will never be stored in both. Intel Pentium II, III, and IV use a mixed pattern where the data must not be in both of them but usually it is. This is called mainly inclusive policy.

Which method is better depends on the architecture. The exclusive cache method can store more data because the data is not repeated on both of them. The advantage is even greater depending on the size of both caches. The major advantage of inclusive policy is that when other devices or processors in a system with several processors want to delete some data, they only need to check the L2 cache because the data will be also stored in the L1 cache for sure, while the exclusive cache policy will have to check on both the L1 and the L2 cache, making the operation slower. As we continue towards higher capacity storage vectors shown, the higher the proximity vector, relative to the CPU, the longer the latency. Also, note the application affinity mapping to the various stack levels. Understanding the application use case, can determine the optimum storage tiers such as Flash cards, arrays, and hybrid solutions. As an application/infrastructure architect, it’s important to consider the appropriate Flash vector mapping moving forward. A detailed design decision tree is provided later in this article.
Storage Classes

Flash is a technology enabler. Therefore, what is the real story on how information is stored and what are the true differentiators between the different ways to store persistent data? Today, the concept of storage classes is taking hold. Storage classes, and specifically storage class memories, are discussed in more detail in the section titled “Storage Class Memories (SCM)”, but the goal is to understand how the various technologies, software abstractions, application considerations, protection mechanisms, and performance play in to the storage landscape. Table 1 shows how speed of information access and longevity requirements of the information relates to the various solutions available today and in the future.

For example, DRAM and other cache technologies are volatile and Flash and spinning disk have different non-volatile or persistent attributes. The matrix shows how storage classes will evolve mapping into a more concise class structure. The trend is different storage technologies are being developed and combined such as NOR Flash, DRAM plus NAND Flash, and PCH (Phase Change Memories) to name a few, to address the upper right hand quadrant.

The state of Persistent Storage

One question that needs to be asked is; what is the current state of technologies that address persistent storage technologies? This section will discuss where we are today.

The spinning Disk – The Dilemma of the Trilemma

Historically, spinning hard disk drives have been the major persistent storage device used in data centers. They are available in a wide range of capacities up to around 4TB with multiple rotational speeds to trade off latency, bandwidth, and cost, with a relatively high random seek
time of several milliseconds, which increases response time. A further drawback of hard disks is that even when idle, a large fraction of peak power is consumed by the spinning disk. Even though disks can be "spun down" during idle periods, long spin-up times compromise energy savings. In addition, there are limitations of magnetic storage in general. It is called the “trilemma”\(^3\). The trilemma states that there are three forces in magnetic storage: storage capacity, writability, and thermal stability. It states that in any design, one can have only two out of the three as shown in the diagram below.

![The trilemma:](image)

Problems can arise when increasing the amount of data in the same physical area. For instance, where the boundary between the '0' and '1' oriented areas of the media are well defined, the grain size of the medium must be reduced in order to retain sharp bit transitions. This then requires an increase in the magnetic anisotropy of the grains, which ensures that the magnetic state is stable for greater than 10 years. However, the magnetic field generated by the electromagnet (or write head) is limited by the material and is theoretically around 2 Tesla. The magnetic field required to reverse the grains depends on the magnetic anisotropy of the material. Thus, there are conflicting requirements for 'thermal stability' and 'writability'. This problem, known as the magnetic recording trilemma, summarizes the key principles of magnetic media design. Possible solutions to the trilemma involve changing the properties of the material so that the writability and thermal stability requirements are both satisfied (for example, using Exchange-Coupled-Composite Media), changing the material type to a single large grain (Bit Patterned Media), or changing the material properties during the write process. The latter example is where the heating part comes in. Since magnetic properties such as anisotropy and magnetization are temperature dependent, by applying heat it is possible to reduce the magnetic anisotropy during the write process, while at the storage temperature the data is stable for > 10 years. This is the principle behind heat-assisted magnetic recording (HAMR) another method that has promise.

\(^3\) http://www.fgarciasanchez.es/thesisfelipe/node5.html
Here comes the Flash Mob

More recently, NAND and NOR Flash-based SSDs have become available, offering random access times of tens of microseconds, with much lower idle power. The highest-performing SSDs are composed of single-level (SLC) Flash which stores one bit per cell. Cheaper, denser Flash chips using multi-level cells (MLC) can be used at the price of roughly double the read latency and triple the write latency for 2-bit-per-cell MLC. In many cases, NAND Flash is most suitable for data where high IOPS are required. Due to the price premium of an SSD, hybrid systems are likely to be more common in the near-term.

Current consumer-grade hard disks and solid-state disks employ serial ATA (SATA) interfaces, allowing throughputs of up to three Gbit/s or six Gbit/s. This narrow channel can often act as a bottleneck for SSDs, preventing the peak theoretical throughput of the storage medium from being reached. Therefore, multiple drives on separate channels can be used together for higher bandwidths. For more I/O-intensive workloads that can benefit from greater bandwidths, PCI-express based I/O cards based on NAND Flash memory provide bandwidths much higher. The bandwidth observed though is still low relative to a memory interface of a typical computer system, and present a bottleneck for workloads that spend most of their time moving data with little or no local manipulation. The usage models for NAND Flash integration can be categorized as:

1. Extended system memory usage model: NAND Flash memory modules are connected to the current system memory interface or to a dedicated Flash memory interface.
2. Storage accelerator usage model: NAND Flash PCI express cards or appliances are connected to the PCI express interface or network. Examples include EMC’s VFCache solution.
3. Alternative storage device usage model: A SSD replaces or augments the hard disk drive. It is connected to the disk interface. An example would be a SATA SSD or a hybrid storage array with Flash and disks.

---

So far, most proposed applications for Flash in the data center have fallen into two categories. The first is disk replacement. Quick access time and low power requirements make Flash a compelling replacement for conventional disks, but, today at a much lower density and higher cost. Accounting for servers and supporting infrastructure, SSDs consume roughly 10 times less energy when idle than disks. SSDs deliver 2.6 times more bandwidth per watt and 3.2 times more bandwidth per dollar, 25 times more I/O operations per second, per dollar, and 2,000 times more IOPS per watt. Flash sometimes also serves as a DRAM replacement. Density and again energy efficiency let Flash compete with DRAM in applications where latency and bandwidth are less important. Flash consumes one-fourth the power of DRAM per byte at one-fifth the price. Flash memory will remain a contender for both roles for the near future, but additional opportunities and challenges are on the horizon. Technology scaling will continue to increase bit density for another 1 to 2 process generations, which will drive down costs. However, smaller Flash cells tend to be less reliable and less durable. In the past two years, lifetime program/erase cycle ratings for high-density Flash devices have dropped from 10 thousand to five thousand cycles. Raw bit error rates have increased as well. This changing landscape of data center computing requires careful design. With all the performance and cost advantages, it is important to have a well thought-out approach to Flash or unsatisfactory performance outcomes are possible.

The economics of Flash
What if I told you Google, Amazon, Microsoft Azure, Dropbox, and others are using Flash?
Many believe that the Internet infrastructure is about commodity hardware and low cost is key. It turns out that not all cloud capacity is rotating rust or spinning disks. Cloud service providers are using Flash! However, some people still have a hard time grasping it, Flash drives can actually save a lot of money. They look more expensive, but when you need higher performance, you need much less and fewer of them.

The cloud is getting Flashy
Dropbox as an example is running servers equipped with SSDs. One SSD can handle about 40,000 reads or writes a second, whereas the average disk gives about 180. In addition, it spend far less on power, running at about one watt as opposed to 15 watts. Doing the math from a IOPS and power perspective, you can save a lot. Wikia, a collaborative wiki web site, first installed SSDs on the company’s caching servers used for providing quick access to data repeatedly accessed by web surfers. Then, they used Flash into the company’s database
servers, where data stored more permanently. Adding these drives provided so much additional speed, that the caching servers were no longer needed, thus increasing performance and reducing cost.

A Dropbox in the Bucket
Dropbox is another example of a web outfit that is embracing SSDs. When you upload files to Dropbox, it stores them in Amazon's S3. However, Dropbox, a cloud startup, runs its own servers that keep track of where all those files are stored. These cache machines are equipped with SSDs which deals with the file meta-data of ones content. Meta-data needs very high-performance databases. Response time or OLTP (On-Line Transaction Processing) is key in customer satisfaction, so performance overrides everything else.

With SSDs, developers can build services more quickly, optimizing time to market (writing code quicker) rather than focusing on efficient code. Historically, code development had to be efficient and account for spinning disk response time attributes, i.e. rotational and seek latency. Because SSDs are so fast, getting the same quick user experience does not require hyper-optimized code, so getting products to market is much quicker. Hard drives perform certain functions particularly well. As a result, one has to write code in such a way that it takes advantage of strengths of both persistent storage technologies (rotating media and Flash) and use them in the appropriate way.

SSDs are fundamentally a no-compromise persistent media solution. Flash frees you, the programmer, from having to think about this dimension on how your code works. This typically makes the biggest difference at the beginning of a project. Hard drives are good at providing access to sequentially stored data, for instance, but they’re not quite as good at accessing random data. “Typically, you would have to worry about how you put all your data very close together so that it could be accessed very quickly on a hard drive. But with SSDs you don’t have to deal with that, especially at the beginning.”

Facebook Gets Flashy
Like Dropbox, Facebook is using Flash storage in its database machines. The company is using hardware from Fusion I/O that adds Flash to servers using the PCI Express connector, long used to connect other peripherals on servers and other machines.
Facebook provides quick access to often used data using a data caching platform called Memcached, but when it pulls additional information from a traditional database, Facebook uses Flash SSDs. PCI-based Flash gives Facebook very high performance on requests, which provides a significant improvement on the overall round-trip time to the user.

Amazon, whose cloud services are now estimated as running one percent of the Internet, is using SSDs for its DynamoDB online database, and Microsoft using them to on the latest incarnation of Microsoft Azure.

**Flash Dancing with the Server**

As previously discussed, Flash design considerations touch the full data center stack. The server is where the application runs. Therefore, when considering how data is accessed from the storage perspective, be it local or distributed, it is very important to consider capacity and latency heuristics. Historically, achieving large capacities of working area (data) with minimal latency has always been a challenge. For applications being developed, today this performance/capacity tradeoff is of paramount importance.

**Applications**

As previously discussed, when one considers the various approaches to addressing specific application requirements and needs from the infrastructure, it is important to take into account the specific goals. Applications are computationally intensive or data intensive:

1. **Compute Intensive**: Maximizes system resources for processing large computations for simulation.
2. **Data Intensive**: Also known as Data Centricity, the application simplifies or addresses the various challenges of working with large datasets generated by sensors, simulation results, and other data sources. One approach is to move computation to the data.

**Data Centricity**

Large-scale data-intensive applications, such as high performance key-value storage systems, are growing in both size and importance. They now are critical parts of major Internet services

---

5 [http://www.wired.com/wiredenterprise/2012/01/amazon-dynamodb/](http://www.wired.com/wiredenterprise/2012/01/amazon-dynamodb/)
6 [http://www.wired.com/wiredenterprise/2012/06/azure-appfog/](http://www.wired.com/wiredenterprise/2012/06/azure-appfog/)
such as Amazon’s Dynamo, LinkedIn Voldemort, and Facebook Memcached architectures. The workloads these systems support share several characteristics. They are:

1. I/O, not computation, intensive, requiring random access over large datasets
2. Massively parallel architectures with thousands of concurrent, mostly-independent operations
3. High load requiring large clusters to support them
4. Size of objects stored is typically small, e.g. 1 KB values, for thumbnail images, a few hundred bytes for wall posts, Twitter messages, etc.

The clusters must provide both high performance and low-cost operation. Unfortunately, small-object random-access workloads are compromised by conventional disk-based or memory-based clusters. The pitiable seek time performance of disk make disk-based systems inefficient in terms of both system performance and performance per watt. High performance DRAM-based clusters, storing terabytes or petabytes of data, are both expensive and consume a substantial amount of power. For example, two 2 GB DIMMs consume as much energy as a one TB disk.

The power draw of these clusters is becoming an increasing percentage of their cost, up to 50% of the three-year total cost of owning a computer. The density of the data centers that house them is in turn limited by their ability to supply and cool 10–20 kW of power per rack and up to 10–20 MW per data center. These challenges necessitate asking: Can we build a cost-effective cluster for data-intensive workloads that uses less than a tenth of the power required by a conventional architecture, but still meets the same capacity, availability, throughput, and latency requirements? Currently, there are solutions being designed, for example, the FAWN (Fast Array of Wimpy Nodes) architecture addressing the challenges of massive scale, which includes the Data Centric characteristics as well as energy efficiency. FAWN couples low power, efficient, embedded CPUs with Flash storage to provide efficient, fast, and cost-effective access to large, random-access data. Flash is significantly faster than disk, much cheaper than the equivalent amount of DRAM, and consumes less power than both consume. FAWN creates a well-matched system architecture around Flash: each node can use the full capacity of the Flash without memory or bus bottlenecks, but does not waste power.

7 http://www.cs.cmu.edu/~fawnproj/
High Performance Computing

High performance computing (HPC) covers the data processing gamut, from the largest Top 500 class supercomputers down to small desktop clusters. HPC is generally categorized as being a class of systems where nearly all of the available compute capacity is devoted to solving a single large problem for substantial periods. HPC systems generally don’t run traditional enterprise applications such as mail, accounting, or productivity applications.

Examples of HPC applications include atmospheric modeling, genomics research, automotive crash test simulations, oil and gas extraction models, and fluid dynamics. HPC systems rely on a combination of high-performance storage and low latency Inter Process Communication (IPC) sessions to deliver performance and scalability to scientific applications. As will be discussed, InfiniBand’s low-latency/high-bandwidth characteristics and “channel” architecture in conjunction with Flash technology are crucially important. Ultra-low latency allows for:

- Scalability
- Cluster performance

Channel I/O delivers:

- Scalable storage bandwidth performance
- Support for shared disk cluster file systems and parallel file systems HPC clusters that are dependent to some extent on low latency IPC for scalability and application performance.

Since the processes comprising a cluster application are distributed across a number of cores, processors, and servers, it is clear that a low-latency IPC interconnect is an important determinant of cluster scalability and performance. In addition to demand for low-latency IPC, HPC systems frequently place enormous demands on storage bandwidth, performance (measured in I/Os per second), scalability, and capacity.

InfiniBand MPI Clusters

InfiniBand’s channel I/O architecture is well suited to high-performance storage and especially parallel file systems. InfiniBand support for HPC ranges from upper layer protocols tailored to support the Message Passing Interface (MPI) in HPC to support for large-scale high-performance parallel file systems such as Lustre. MPI is the predominant messaging middleware found in HPC systems.
MPI is the de facto standard and dominant model in use today for communication in a parallel system. Although it is possible to run MPI on a shared memory system, the more common deployment is as the communication layer connecting the nodes of a cluster. Cluster performance and scalability are closely related to the rate at which messages can be exchanged between cluster nodes; the classic low latency requirement. This is native InfiniBand territory and its clear stock in trade.

MPI is sometimes described as communication middleware, but perhaps a better, more functional description would be to say that it provides a communication service to the distributed processes comprising the HPC application. To be effective, the MPI communication service relies on an underlying messaging service to conduct the actual communication messages from one node to another. The InfiniBand Architecture, together with its accompanying application interfaces, provides a RDMA messaging service to the MPI layer. Applications use this service to transfer messages between them. In the case of HPC, one can think of the processes comprising the HPC application as being the clients of the MPI communication service, and MPI, in turn as the client of the underlying RDMA message transport service. InfiniBand’s copy avoidance architecture and stack bypass deliver ultra-low application-to-application latency and high bandwidth coupled with extremely low CPU utilization. MPI is easily implemented on InfiniBand by deploying one of a number of implementations, which take full advantage of the InfiniBand Architecture.

**Storage and High Performance Computing**

A HPC environment comprises multiple processes running in parallel. In a HPC cluster, processes are distributed. The distribution could be across the cores comprising a processor chip, across a number of processor chips comprising a server, or distributed across a number of physical servers. Depending on the use case of the cluster, it is often the case that each of the distributed processes comprising the application requires access to storage. As the cluster is scaled up in terms of the number of processes comprising the application, the demand for

---

**Figure 7 - Shared File System**
storage bandwidth increases. Two HPC storage architectures, shared disk cluster file systems, and parallel file systems are quite common and channel I/O either through InfiniBand or RoCE (RDMA over converged Ethernet), which plays an important role, especially when storage comprises Flash. Other storage architectures, such as network-attached storage (NAS), are also possible.

A shared disk cluster file system is exactly as its name implies, as shown in Figure 7. It permits distributed processes comprising the cluster to share access to a common pool of storage. Typically, in a shared disk file system, the file system is distributed among the servers hosting the distributed application with an instance of the file system running on each processor and a side band communication channel between file system instances used for distributed lock management. The storage is typically block. By creating a series of independent channels, each instance of the distributed file system is given direct access to the shared disk storage system. The file system places a certain load on the storage system, which is equal to the sum of the loads placed by each instance of the file system. By distributing this load across a series of unique channels, the channel I/O architecture provides benefits in terms of parallel accesses and bandwidth allocation. The performance of a shared disk file system and hence the overall system is dependent on the speed with which locks can be exchanged between file system instances. InfiniBand’s ultra-low latency characteristic allows the cluster to extract maximum performance from the storage system.
A parallel file system, as shown in Figure 8, such as the Lustre file system is similar in some ways to a shared disk file system. It is designed to satisfy storage demand from a number of clients in parallel, but with several distinct differences. The file system however, is sited together with the storage devices rather than being distributed. A relatively thin file system client serves each process comprising the distributed application. Within the storage system, the control plane (file system metadata), is often separated from the data plane (object servers) containing user data. This promotes parallelism by reducing bottlenecks associated with accessing file system metadata, and it permits user data to distribute across a number of storage servers, which are devoted to strictly serving user data. Each file system client creates a unique channel to the file system metadata. This accelerates the metadata look-up process and allows the metadata server to service a substantial number of accesses in parallel. In addition, each file system client creates unique channels to the object storage servers where user data is actually stored. By creating unique connections between each file system client and the storage subsystem, a parallel file system can reap maximum advantage from InfiniBand’s channel I/O architecture. The unique and persistent low latency connections to the metadata server(s) allow a high degree of scalability by eliminating the serialization of accesses sometimes experienced in a distributed lock system. By distributing the data across an array of object servers and by providing each file system client with a unique connection to the object servers, a high degree of parallelism occurs as multiple file system clients attempt to access objects. By creating unique and persistent low-latency connections between each file system client and the metadata servers, and between each file system client and the object storage servers, the “Lustre” file system delivers scalable performance.

Flash Standards
Currently, enterprise Flash solutions utilize quite often proprietary and inefficient software to communicate with Flash. However, with the increased adoption of SSDs in data centers, there
has emerged a need to design a storage interface suitable for high performance and lower latency. The SCSI protocol is the logical foundation for all modern storage systems, and SCSI Express provides a way of transporting SCSI over PCIe in a standardized fashion.

**Flash Dancing with Storage**

As companies focus on the Next-Generation Data Center and alignment to the cloud, it is very important that they understand the storage ecosystem and how all of the pieces fit together to help companies achieve business agility and efficiency. The ecosystem includes how storage arrays connect to their respective counterparts and how virtualization is causing massive consolidation of system resources. This section will also discuss how Flash technology will play in this new environment.

**Flash Personalities**

Most Flash drives have an abstraction interface to the physical storage. As shown in Figure 9, the kernel writes directly to the hardware disk through LBA’s (Logical Block Addresses). However, for Flash, a translation layer is required to convert LBA’s to PBA (Physical Block Addresses) to address the many limitations of this type of media. This introduces additional complexities and as such, additional performance obstacles exist.

---

**Figure 9: Traditional Stack and Flash Translation Layer (FTL) Storage Stack**

One solution considered is to expose a richer interface to the system (Kernel and even the application). This might include providing multiple storage media “personalities” to the system. This could include a “disk personality” (the current standard) as well as others that might expose Flash’s block and page structures while providing transparency, for example, block-level wear-leveling. Another personality would expose the Flash directly and require the system to manage
wear leveling and remapping as needed by that application. For such a storage device, customization could occur in the kernel driver or even at the application level, letting applications map their storage needs directly and efficiently onto the Flash.

Aggregating management across multiple Flash devices would enable a large server or server farm to make global optimizations based on error rate and performance variations, allowing it to extract the maximum performance from the Flash array while meeting application-specific reliability targets. In the second approach, the FTL might provide an object store abstraction that lets applications indicate use patterns on a per object basis. An application might designate one object for streaming, sequential writes and another for fast, random reads. We will discuss additional details and examples in the section titled Flash Transaction Layer considerations later in this article.

**The Blender effect**

Prior to the widespread adoption of virtualization in enterprise environments, storage was “predictable.” System admins and storage admins could easily describe what the I/O patterns looked like at the server, fabric, and storage levels. For example, the enterprise BI (business intelligence) environment may have had high read requests during the day, and significant write operations at night as the processing, analysis, and reporting functions were executed. For the most part, storage was isolated to a specific server. Run-of-the-mill file systems did not handle shared/concurrent access efficiently, which resulted in isolated local storage or LUN-masked SAN storage. Storage and compute became a 1:1 mapping, which was predictable. However, once virtualization hit the scene, I/O was sliced and diced into what is known as the I/O Blender effect. As a result, I/O is no longer predictable. Rather, it looks like it has gone through an I/O Blender.

**Best Practice - Consider what the I/O Blender effect has in your environment**

The I/O Blender effect can influence a number of infrastructure components and can have a negative impact on system performance:

1. Storage controllers are configured with specific amounts of cache that buffer and some level of workspace for incoming/outgoing data. The unpredictable insufficient controller cache nature of I/O Blender patterns and concurrent access to the same
resources can overwhelm controller caches and result in degraded performance with added read/write latencies.

2. **Insufficient SSD cache** – SSD devices are hitting new levels of IOPS and throughput for persistent storage and their impact is indeed significant. However, so is the cost per Gigabyte. Modern storage systems can utilize SSDs as a caching mechanism. Due to cost, though, many environments can only afford the bare minimum for caching functionality. The I/O Blender effect can exhaust the added buffer that SSDs provide in SAN caching. Due to the random I/O patterns and concurrent access, a SAN cache algorithm may have a difficult time determining the host’s hot and cold data for migration, writing cached data to the spinning disks in the array, or only be able to keep a small amount of a number of systems in cache. Essentially, the functionality and benefit of the purchase has a questionable return for the environment.

3. **Read/write scheduling in arrays** - Storage controllers are responsible for coordinating read and write operations to the arrays. Depending on the disk and array configuration, scheduling and performing these operations can have a significant impact on performance, considering that traditional storage arrays are typically designed for handling sequential reads and writes. This design decision made sense at the time because it could be expected that inbound/outbound requests were for larger contiguous amounts of data. The I/O Blender effect negates the semblance of sequentially.

4. **VMDK file locking in block storage** – In VMware-based environments that utilize block storage, a mechanism is in place to lock VMDK files prior to accessing them for I/O operations. In a highly active environment, such as those impacted by the I/O Blender effect, the hypervisor spends much time and effort locking and unlocking files, negating any SDD performance enhancement opportunities. In addition, all of the locking and unlocking can take a heavy toll on SAN and environmental performance.

5. **LUN Alignment** – Operating system disk access relies upon placing data in blocks/clusters on disk. SAN storage relies upon placing data on blocks in the array of disks. A condition exists wherein the block sizes of the OS do not match (by some factor) with the block sizes on SAN storage. This LUN misalignment can have serious impact on performance.\(^8\)

---
\(^8\) [http://www.boche.net/blog/index.php/2009/03/20/storage-block-size-and-alignment/]
**Best Practice - Understand the Blender effect on trending apps**

The I/O Blender may not go away anytime soon. Virtualization has taken an almost monopoly position in enterprise environments and the benefits of virtualization have been realized and highly adopted. There is no turning back. With upcoming trends such as VDI, Big Data, Multicast rich media, etc., the effect will only worsen, unless one plans ahead. One approach is to adjust the organization of VMs for like workloads. This may involve different SAN and SDD configurations to ensure like-workloads are organized rather than providing a single array of disks. Similar workloads are more predictable and can be planned appropriately. Consider that in many instances, this may result in a larger number of storage pools in your environment to manage.

**Best Practice - Utilization of storage offloading functions**

Hypervisor vendors have been working hard with storage vendors to offload many storage functions from the hypervisor to the storage controllers. This allows the storage system to more efficiently handle the operation, reduces pressure on the hypervisor, and helps reduce congestion on any storage connections (local, fiber, or network) and reap what SSDs can offer.

**Best Practice - Be cognizant of LUN Alignment**

This is mostly an issue with older Operating System versions. Newer versions will auto-align on installation and should not be an issue. However, spending the time to ensure LUN alignment is correct will reduce the I/O overhead and relieve pressure on the storage controller and subsystems that perform the operations. LUN alignment has the ability to have a significant and positive impact on systemic health!

**Best Practice - Consider the workload that is being placed on the SDD storage system AND what will be placed on the system in the future**

Understanding the workload will help system admins and storage admins ensure proper grouping of like-workload types. Being able to proactively plan where workloads are going to be placed will ensure optimal performance will be maintained and identification of capacity constraints BEFORE they occur.

**Best Practice - Utilize expandable and extensible storage systems**

Storage systems are exceptionally difficult to replace. High cost and the role in the compute environment are difficult to overcome. While storage migration functions in the hypervisors help ease the transition, being able to expand a storage system in some fashion ensures the decisions you make now can be adapted, adjusted, and enhanced to meet the needs of today.
and tomorrow. Adjustments and enhancements may include: addition of trays of disks, support for multiple disk types (SAS/FC, SATA, SSD), SSD-based caching, upgradable firmware to add support for future features, additional controllers, swappable HBAs/NICs, etc.

**Best Practice - Utilize SAN storage that is designed/optimized for virtualization**

A small number of vendors are bringing virtualization-focused storage solutions to the market. EMC is one of them. Large storage vendors are re-tooling their solutions to address issues introduced by virtualization too. Make a conscious effort to scrutinize and evaluate storage solutions outside of the box. Storage systems with a fondness for virtualization have immediate value outside of the ability to store VM data. Additional functions, like performance metrics, provisioning, management, functional offloading, migration of VM workloads in tiers of storage, etc. all have value beyond storage connectivity and RAID configurations.

**Best Practice - Utilize an up-to-date infrastructure**

The I/O Blender effect has a way of highlighting the bottlenecks in your virtualization environment. The network is the backbone of any data center. Reducing or eliminating bottlenecks can be reduced or eliminated. For example;

**10Gb networking**: for IP-based storage solutions, 10Gb networking can ensure storage requests are not being held up at the NIC level. 10Gb can provide more than ample bandwidth between the compute and storage tiers.

**Fiber channel HBAs**: Similar to 10Gb networking, utilizing faster/more efficient HBAs ensure that data requests are not clogged up in the fabric.

**Servers**: CPUs, memory, PCI expansion cards, etc. all have a role in ensuring peak performance. CPU performance is increasing in some fashion all the time. Memory is getting denser and faster (which reduces the need to go to storage in many cases). PCI buses are speeding up and the cards that are used are faster and more efficient.

**SSDs, Tiering, and offload functions**: SSDs provide a significant increase in performance, if utilized properly. SSDs, in conjunction with automated tiering functions, provide amazing value to all workloads. Offloading hypervisor functions to the storage systems allow the storage system to more efficiently handle the request without sustained impact to the compute and network/fabric.

**Flash Dancing with the Network**

Networks are often thought of as the set of routers, switches, and cables that are plugged in to servers, storage devices and clients.
When asked, most people would probably say that a network is used to connect servers to other servers, storage, and a network backbone. Historically, networking has been a “bottom up” approach, focused on the “traffic”, but not necessarily what the application wants or needs. A server provides a selection of communication services to the applications it supports; one for storage, one for networking, and frequently a third for specialized IPC (inter process communication) traffic. This complement of communications stacks (storage, networking, etc.) and the device adapters accompanying each one comprise a shared resource. This means that the OS “owns” these resources and makes them available to applications on an as-needed basis. The application usually relies on the operating system to provide the communication services it needs.

We will find out that this model is changing and Flash storage is the accelerator. The network aspects of a Flash design cannot be ignored. It’s not just access time to the Flash, it also means how do we get the data from the storage array to the compute platform with the highest bandwidth with the least amount of latency? Therefore, one needs to consider both the transport and compute platform stack to achieve the highest performance possible.

**Data-Intensive applications**
Providing high-speed data transfer is vital to various data-intensive applications. There have been remarkable technology advances to provide ultra-high-speed network bandwidth. Existing protocols and applications may not be able to fully utilize this bare-metal bandwidth due to their inefficient design, specifically in the server OS. A new methodology is required when access time and response time of the underlying data becomes a less obtrusive performance factor. As a result, two areas need to be addressed to achieve this network speed up; it is in the network transport (FCoE or InfiniBand) and server network accelerators such as RDMA (Remote Direct Memory Access) technologies, which will be discussed in more detail in the following sections.

**Transports**
The first Flash dance to learn in the networking world is focused on the transport. How do we get the data from and to external Flash arrays and other Flash appliances within the data center to the application as quickly as possible? The two major players are InfiniBand and FCoE (Fiber Channel over Ethernet) with a little help from InfiniBand’s secret sauce called RDMA, also known as RoCE (RDMA over Converged Ethernet).
**InfiniBand**

InfiniBand is distinctly an application-centric network solution with the primary goal of making application accesses to other applications and to storage, specifically Flash, as simple, efficient and direct as possible. Note that when the term application is used, it does not necessarily mean end-user apps only. Embedded apps such as storage backbone networking is also a use-case.

The idea behind InfiniBand is providing applications with an easy-to-use “messaging service”. This service can be used to communicate with other applications or processes or to access storage. Instead of making a request to the operating system for access to one of the server’s communication resources, an application accesses the InfiniBand messaging service directly. Since the service provided is a straightforward messaging service, the complex dance between an application and a traditional network is eliminated. Since storage traffic can be viewed as consisting of control messages and data messages, the same messaging service, as previously discussed, is equally at home for use in storage applications. Many storage array vendors, such as EMC, are using this as there embedded messaging backbone of choice.

**Best Practice - Consider utilizing InfiniBand’s inherent messaging service to enhance performance**

InfiniBand’s architecture gives every application direct access to the messaging service. Direct access means that an application need not rely on the operating system to transfer messages. The application still has isolation and protection that an operating system would provide through this messaging service by creating a “channel” or queue pair (QP), connecting an application to any other application or service with which the application needs to communicate as shown in Figure 10. This application-centric approach is the key differentiator between InfiniBand and traditional networks. The true power of InfiniBand’s design is that even two disparate virtual spaces potentially located in entirely disjointed physical address spaces, hosted by multiple servers, even over a physical distance, can communicate.

![Figure 10 - InfiniBand Logical Channel Architecture](image)
seamlessly and never will a piece of data be copied twice. This is huge!

In addition to the “channel” and “direct buffer copy” attributes of InfiniBand, the third architectural strength of InfiniBand provides channel transfer semantics. There are two transfer semantics; a channel semantic sometimes called SEND/RECEIVE and a pair of memory semantics called RDMA READ and RDMA WRITE. When using the channel semantic, the message is received in a data structure provided by the application on the receiving side. This data structure is pre-posted on its receive queue. Thus, the sending side does not have visibility into the buffers or data structures on the receiving side; instead, it simply SENDS the message and the receiving application RECEIVES the message.

The memory semantic is a little different. In this case, the receiving side application registers a buffer in its virtual memory space. It passes control of that buffer to the sending side which then uses RDMA READ or RDMA WRITE operations to either read or write the data in that buffer. A typical storage operation may illustrate the difference. The initiator wishes to store a block of data. The initiator places the block of data in a buffer in its virtual address space and uses a SEND operation to send a storage request to the target (EX Storage). The target then uses RDMA READ operations to fetch the block of data from the initiator virtual buffer. Once the operation is done, the target uses a SEND operation to return ending status to the initiator. Note that the initiator does not wait for completion; this architecture is designed for multiprocess operations. The data transfer phase and the storage operation comprise the bulk of the operation. Included as part of the InfiniBand Architecture and located right above the transport layer is the software transport interface. The software transport interface contains the queue pairs (QPs); keep in mind that the QPs are the structure by which the RDMA message transport service is accessed.

The software transport interface also defines all the methods and mechanisms that an application needs to take full advantage of the RDMA message transport service. For example, the software transport interface describes the methods that applications use to establish a channel between them. An implementation of the software transport interface includes the APIs and libraries needed by an application to create and control the channel and to use the QPs in order to transfer messages.
As shown in Figure 11, this messaging service requires a complete network stack just as one would find in any traditional network. It includes the InfiniBand transport layer to provide reliability and delivery guarantees, a network layer, and link and physical layers similar to Ethernet. However, it’s a special kind of network stack because it has features that make it simple to transport messages directly between an applications’ virtual memory, even if the applications are remote with respect to each other. So, with the transport layer and the API, this architecture is better thought of as a Remote Direct Memory Access (RDMA) message transport service.

How, exactly, does a sending application, for example, send a message? The InfiniBand software transport interface specification defines the concept of a verb. The word “verb” was chosen since a verb describes action. The set of verbs, taken together, are simply a semantic description of the methods the application uses to request service from the RDMA message transport service. For example, “Post Send Request” is a commonly used verb to request transmission of a message to another application. The verbs are fully defined by the specification and are the basis for specifying the APIs that an application uses. The InfiniBand Architecture specification doesn’t define the actual APIs; that is left to other organizations such as the OpenFabrics Alliance, which provides a complete set of open source APIs and software, which implements the verbs and works seamlessly with the InfiniBand hardware.

**Best Practice - Consider InfiniBand solutions in the Enterprise**

Many consider InfiniBand’s performance (latency, bandwidth, CPU utilization) to be of extreme importance. For example, certain applications in the financial services industry (such as arbitrage trading) demand the lowest possible application-to-application latencies at nearly any cost. Reducing expenses associated with purchasing and operating a data center means being able to do more with less—fewer servers, fewer switches, fewer routers, and fewer cables. As a
result, enterprise data center management focuses on the efficient use of IT resources. Server virtualization is a strategy successfully deployed now in an attempt to reduce the cost of server hardware in the data center. This means that fewer physical servers are needed to support approximately the same application workload. Even though server virtualization is an effective strategy for driving up consumption of previously unused CPU cycles, it does little to improve the effective utilization (efficiency) of the existing processor, memory, and persistent storage architectures. CPU cycles and memory bandwidth are still devoted to executing the network stack at the expense of application performance. Effective utilization, or efficiency, is the ratio of a server resource, such as CPU cycles or memory bandwidth that is devoted directly to executing an application program compared to the use of server resources devoted to housekeeping tasks. The trick is to apply technologies that can fundamentally improve the servers overall effective utilization. If successful, the data center owner will achieve much higher usage of the data center server’s, switches, and storage that were purchased.

**Best Practice - Utilize InfiniBand to increase application performance processing**

As previously discussed, the RDMA messaging service avoids the use of the operating system (using operating system bypass). Operating system bypass produces another effect of much greater value to the enterprise data center. Server architecture and design is an optimization process. At a given server price point, the server architect can afford to provide only a certain amount of main memory; the important corollary is that there is a finite amount of memory bandwidth available to support application processing at that price. As it turns out, main memory bandwidth is a key determinant of server performance. The memory bandwidth provided on a given server must be allocated between application processing and I/O processing. This is true because traditional I/O, such as TCP/IP networking, relies on main memory to create an anonymous buffer pool, which is integral to the operation of the network stack. Each inbound packet must be placed into an anonymous buffer and then copied out of the anonymous buffer and delivered to a user buffer in the application’s virtual memory space. Each inbound packet is copied at least twice.

As shown in Figure 12, each inbound network packet consumes from two to three times the memory bandwidth as the inbound packet is placed into an anonymous buffer by the NIC and then copied from the anonymous buffer and into the application’s virtual buffer space.
The load offered to the memory by network traffic is largely dependent on how much aggregate network traffic is created by the applications hosted on the server. As the number of processor sockets on the motherboard increases, and as the number of cores per socket increases, and as the number of execution threads per core increases, the offered network load increases proportionally. In addition, each increment of offered network load consumes two to three times as much memory bandwidth as the underlying signaling rate of the network. For example, if an application consumes 10Gb/s of inbound network traffic (which translates into 20Gb/s of memory bandwidth consumed), multiple instances of an application running on a multicore CPU will consume proportionally more network bandwidth and twice the memory bandwidth. The TCP/IP network protocol stack also consumes CPU processor cycles as well. The cycles consumed in executing the network protocol and copying data out of the anonymous buffer pool and into the applications’ virtual buffer space are cycles that cannot be devoted to application processing. These cycles are purely overhead. Therefore, significant savings could be realized if you could reduce the number of CPUs required or the amount of memory bandwidth consumed without reducing the ability to execute user applications. The benefit and savings of both capital and operating expense would be maximized if one used InfiniBand. The InfiniBand Architecture with Channel I/O dramatically improves the server’s effective utilization (its efficiency) by reducing CPU, memory, network, and I/O demand. Directing precious CPU and memory resources to useful application processing is the goal.

**Best Practice - Leverage InfiniBand Messaging Queue Architecture**

InfiniBand’s architecture provides a messaging service. This message service can be used to transport IPC messages, transport control and data messages for storage, or transport messages associated with any other of a range of usages. This differentiates InfiniBand from a traditional TCP/IP/Ethernet, which provides a byte stream transport service, or Fibre Channel, which provides a transport service specific to the Fibre Channel wire protocol. The key feature that sets InfiniBand apart from other network technologies is that InfiniBand is designed to
provide message transport services directly to the application layer, whether it is a user application or a kernel application.

Traditional networks require the application to solicit assistance from the operating system to transfer a stream of bytes. The top layer of the InfiniBand Architecture defines a *software transport interface*, which defines the methods that an application uses to access the complete set of services provided by InfiniBand as shown in Table 2. An application accesses InfiniBand’s message transport service by posting a Work Request (WR) to a work queue. As previously discussed, the work queue is part of a structure, called a Queue Pair (QP) representing the endpoint of a channel connecting the application with another application. A QP contains two work queues; a Send Queue and a Receive Queue, hence the expression QP. A WR, once placed on the work queue, is in effect an instruction to the InfiniBand RDMA message transport to transmit a message on the channel, or perform some sort of control or housekeeping function on the channel. An application uses a POST SEND verb, for example, to request that the InfiniBand transport send a message on the given channel. This is what is meant when one says that the InfiniBand network stack goes all the way up to the application layer. InfiniBand provides methods that are used directly by an application to request service from the InfiniBand transport. This is very different from a traditional network, which requires the application to solicit assistance from the operating system in order to access the server’s network services.

<table>
<thead>
<tr>
<th>Stack</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Application</strong></td>
<td>Consumer of InfiniBand message services</td>
</tr>
<tr>
<td></td>
<td>– posts ‘work requests’ to a queue</td>
</tr>
<tr>
<td></td>
<td>– each work request represents a message…a unit of work</td>
</tr>
<tr>
<td><strong>App Interface (channel</strong></td>
<td><strong>Channel interface (verbs):</strong></td>
</tr>
<tr>
<td><strong>Interface</strong></td>
<td>– allows the consumer to request services</td>
</tr>
<tr>
<td><strong>Software Transport</strong></td>
<td><strong>InfiniBand Channel interface provider:</strong></td>
</tr>
<tr>
<td><strong>Interface</strong></td>
<td>– Maintains work queues</td>
</tr>
<tr>
<td></td>
<td>– Manages address translation</td>
</tr>
<tr>
<td></td>
<td>– Provides completion and event mechanisms</td>
</tr>
<tr>
<td><strong>InfiniBand Transport</strong></td>
<td><strong>IB Transport:</strong></td>
</tr>
<tr>
<td><strong>Layer</strong></td>
<td>– Packetizes messages</td>
</tr>
<tr>
<td></td>
<td>– Provides transport service</td>
</tr>
<tr>
<td></td>
<td>- reliable/unreliable, connected/unconnected, datagram</td>
</tr>
</tbody>
</table>
- Implements RDMA protocol
- send/receive, RDMA r/w, Atomics
- Implements end-to-end reliability
- Assures reliable delivery

Table 2: InfiniBand Stack

**Best Practice - Design Flash networking stacks built on top of standard Verbs**

The verbs, as defined, are sufficient to enable an application developer to design and code the application for use directly on top of an InfiniBand network. However, if the application developer were to do so, one would need to create a set of customized application or OS interface code. In a world of specialized applications, such as high performance computing, or embedded applications or as a storage network backbone, where the last ounce of performance is required, such an approach makes a great deal of sense. However, in a world of more broadly distributed applications and operating systems it may be prohibitive for the application or hardware developer to adopt a given application or technology. In more broadly deployed systems, application accesses system services via a set of APIs often supplied as part of the operating system. InfiniBand verbs fully specify the required behaviors of the API’s. In effect, the verbs represent a semantic definition of the application interface by specifying the methods and mechanisms.

**Best Practice - Utilize an open API for Flash appliance related transports**

OpenFabrics Alliance (OFA) and other commercial providers supply a complete set of software stacks, which are both necessary and sufficient to deploy an InfiniBand-based system for high performance Flash-based systems. The software stack available from OFA, as the name suggests, is open source and freely downloadable under both BSD and GNU licenses. The OFA is broadly supported by providers of InfiniBand hardware, by major operating system suppliers and suppliers of middleware, and by the Linux open source community. However, even supplying APIs is not, by itself, sufficient to deploy InfiniBand. The software elements required to implement InfiniBand fall into three major areas.

The first is the set of Upper Layer Protocols (ULPs) and associated libraries. There is a set of mid-layer functions helpful in configuring and managing the underlying InfiniBand fabric and that provide needed services to the upper layer protocols, and there is a set of hardware specific device drivers.
The complete set of software components provided by the OpenFabrics Alliance is formally known as the Open Fabrics Enterprise Distribution (OFED). A rich set of Upper Layer Protocols (ULPs) are provided by the OFA, as shown in Figure 13, including:

1. **SDP – Sockets Direct Protocol.** This ULP allows a sockets application to take advantage of an InfiniBand network with no change to the application.

2. **SRP – SCSI RDMA Protocol.** This allows a SCSI file system to directly connect to a remote block storage chassis using RDMA semantics. Again, there is no impact to the file system itself.

3. **iSER – iSCSI Extensions for RDMA.** iSCSI is a protocol allowing a block storage file system to access a block storage device over a generic network. iSER allows the user to operate the iSCSI protocol over an RDMA capable network.

4. **IPoIB – IP over InfiniBand.** This important part of the suite of ULPs allows an application hosted in, for example, an InfiniBand-based network to communicate with other sources outside the InfiniBand network using standard IP semantics. Although often used to transport TCP/IP over an InfiniBand network, the IPoIB ULP can be used to transport any of the suites of IP protocols including UDP, SCTP, and others.

5. **NFS-RDMA – Network File System over RDMA.** NFS is a well-known and widely deployed file system providing file level I/O (as opposed to block level I/O) over a conventional TCP/IP network. This enables easy file sharing. NFS-RDMA extends the protocol and enables it to take full advantage of the high bandwidth and parallelism provided naturally by InfiniBand.

6. **Lustre support** – Lustre is a parallel file system enabling, for example, a set of clients hosted on a number of servers to access the data store in parallel. It does this by taking advantage of the InfiniBand Channel I/O architecture, allowing each client to establish an independent protected channel between itself and the Lustre Metadata Servers (MDS) and associated Object Storage Servers and Targets (OSS, OST).
7. **RDS – Reliable Datagram Sockets** offers a Berkeley sockets API allowing messages to be sent to multiple destinations from a single socket. This ULP, originally developed by Oracle, is ideally designed to allow database systems to take full advantage of the parallelism and low latency characteristics of InfiniBand.

8. **MPI** – The MPI ULP for HPC clusters provides full support for MPI function calls.

A traditional API provides access to a particular kind of service, such as a network service or a storage service. These services are usually provided over a specific type of network or interconnect. The socket API, for example, gives an application access to a network service and is closely coupled with an underlying TCP/IP network. Similarly, a block-level file system is closely coupled with a purpose-built storage interconnect such as Fiber Channel. This is why data center networks today are commonly thought of as consisting of as many as three separate interconnects; one for networking, one for storage and possibly, a third for IPC. Each type of interconnect consists of its own infrastructure and application level APIs. This is where the InfiniBand Architecture is different: The set of ULPs provide a series of interfaces that an application can use to access a particular kind of service (storage service, networking service, IPC service, or others). However, each of those services is provided using a *single underlying network*. At their lower interfaces, the ULPs are bound to a single InfiniBand network. There is no need for multiple underlying networks coupled to each type of application interface. In effect, the set of ULPs gives an application access to a single unified fabric, completely hiding the details associated with a unified fabric, while avoiding the need for multiple data center networks.

**FCoE**

Fiber Channel over Ethernet (FCoE) took the physical transport layers of traditional Fiber Channel and replaced them with Ethernet, and the IEEE created the DCB efforts to make sure that the underlying Ethernet transport was reliable and lossless so that it could support FCoE. Because it still “looked” like Fiber Channel at the upper layers, there is a great deal of interoperability between Fiber Channel and FCoE. The standards bodies took this ubiquitous standard and enhanced it to support RDMA—the session part of InfiniBand—and created RoCE (RDMA over Converged Ethernet). More on this to follow.

**Network Accelerators (RDMA)**

Internet Wide Area RDMA Protocol (iWARP) and RDMA over Converged Ethernet (RoCE) are the two commonly known network accelerator protocols or Remote Direct Memory Access.
(RDMA) technologies over Ethernet and/or InfiniBand transports. Introduced in 2002, iWARP solutions over Ethernet have seen limited success due to implementation and deployment challenges. Benchmarks results published by lead iWARP technology providers such as Mellanox have shown 6usec latency numbers. Recent enhancements to the Ethernet data link layer under the umbrella of IEEE Data Center Bridging (DCB) enabled the application of advanced RDMA transport services over DCB.

In early 2010, this technology, now known as RDMA over Converged Ethernet (RoCE) was standardized by the IBTA. RDMA communications using Send/Receive semantics and kernel bypass technologies as discussed previously, in server and storage interconnect products, permit high throughput and low-latency networking. For example, the benefits of high throughput and low-latency networking became predominant today in EDC (Electronic Data Capture) applications. Many believe that efficiency is synonymous with return on investment (ROI), which is an ever-critical goal of the EDC market especially with the scaling needs of Web 2.0 and cloud computing applications. As such, the importance of low latency technologies such as RDMA has grown, and the need for viable RDMA products that is broadly deployable across market and application segments has become critical.

iWarp

iWARP uses IETF-defined Remote Direct Data Placement (RDDP) to deliver RDMA services over standard, unmodified IP network and standard TCP/IP Ethernet data link services and the RDDP is layered over existing IP transports. While this helps support a broad range of network characteristics, from short to long-range networks, RDDP brings some inherent disadvantages:

1. RDDP traffic cannot be easily managed and optimized in the fabric itself, leading to inefficiency in deployments. It does not provide a way to detect RDMA traffic at or below the transport layer, e.g. within the fabric itself. Sharing of TCP’s port space by RDDP makes using flow management impossible, since the port alone cannot identify whether the message carries RDMA or traditional TCP.

2. Application integration is harder. Because RDDP is layered over existing transports, it shares their port space, and the necessary RDMA capability detection and service resolution or negotiation is an additional, RDMA-specific step, which applications must make.

3. Hardware offload implementation is harder. RDDP is designed to work over the existing TCP transport and Ethernet data link layers. The Ethernet data link delivers best effort
service and is prone to packet loss, relying on the TCP layer to deliver reliable services. The need to support existing IP networks, including wide area networks, require coverage of a larger set of boundary conditions with respect to congestion handling, scaling, and error handling, causing hardware offload of the RDMA and associated transport operations significantly harder.

4. WARP is a relatively old technology that was introduced in 2002. Most advanced 10Gig Ethernet that support iWARP shows 6usec latency which can be achieved by a variety of today’s 10GigE NICS over common TCP/IP.

The above challenges have significantly delayed the availability of cost-effective and easily deployable iWARP products. As such, iWARP’s usage in the industry is very limited, and has not been able to leverage the ubiquity of Ethernet to proliferate its use.

RoCE

RoCE (RDMA over Converged Ethernet) and even more affectionately pronounced “Rocky”, is a recent enhancement to the Ethernet data link layer protocol and opens significant opportunities to proliferate the use of RDMA into mainstream data center applications by taking a fresh and evolutionary look at how such services can be more easily and efficiently delivered over Ethernet. It is based on the following standards:

a. IEEE 802.1Qbb Priority flow control (PFC) standardizes a link level flow control that recognizes eight traffic classes per port. While traditional Ethernet pause is flow controlled at the granularity of physical ports, with priority flow control (PFC), pause is at the granularity of a traffic class. PFC is the first per priority flow control mechanism for Ethernet.

b. IEEE 802.1Qau standardizes congestion notification through an admission control algorithm called Quantized Congestion Notification (QCN). The QCN algorithm is inspired by congestion control in TCP, but implemented at layer 2.

With these new enhancements, the market took a fresh look at iWARP as discussed in the previous section. As noted, the wider scope of iWARP to deliver RDMA services over existing IP networks resulted in unnecessary burdens and product and deployment failures and delays. Many of the services in the iWARP stack become redundant when applied over DCB. Since DCB provides congestion control, the congestion functions provided in TCP, SCTP, and RDDP become redundant. Over DCB, TCP slow-start behavior can be relaxed or removed. The question is whether the iWARP stack be redesigned with a new approach to make it more
suitable to operate over DCB and reduce the implement and deployment complexities it continues to suffer. From both the technology and business perspectives, the natural choice for building RDMA services over DCB is to apply more efficient RDMA transport services over Ethernet. RDMA over Converged Ethernet or RoCE has quickly become the sought-after solution.

**Flash Technology Considerations**

Some computer chip background is appropriate at this point. What does a device or gate look like? Well, most of us are not semiconductor engineers, but Figure 14 provides a high-level view of Flash. Basically, there is a floating gate, that is either charged or not charged so that the connection between the source and drain are either connected or not. The important thing to know is this floating gate keeps its state until the system changes it. The problem is that this floating gate wears out and, just as your plumber must do when the faucet leaks, you have to change the fixture and replace it.

Over the past few years, computer systems of all types have started integrating Flash memory. Initially, Flash’s small size, low power consumption, and physical durability made it a natural fit for media players and embedded devices.

Lately, Flash’s rising density has won it a place in laptops and some desktop machines. Flash is now poised to make deep inroads into the data center. There, Flash memory’s high density, low power, and low-cost I/O’s per second will drive its adoption and enable its application far beyond simple hard drive replacements. To date, however, many uses of Flash have been hamstrung by a fundamental challenge of the technology: Flash is neither magnetic disk nor DRAM. It has its own performance advantages and quirks that system designers must address at several levels to best exploit it. In the following sections, we will discuss specific details to consider when understanding and designing with Flash technology.
Write Amplification

Write amplification (WA) is an undesirable phenomenon associated with Flash memory and SSDs where the actual amount of physical information written is a multiple of the logical or intended amount.

Best Practice - Consider Flash’s limited write cycles

Because Flash memory must be erased before it can be rewritten, the process to perform these operations results in moving (or rewriting) user data and metadata more than once. This multiplying effect increases the number of writes required over the life of the SSD, which shortens the time it can reliably operate. The increased writes also consume bandwidth to the Flash memory, which mainly reduces random write performance to the SSD. Many factors will affect the write amplification of an SSD; some can be controlled by the user and some are a direct result of the data written to and usage of the SSD. Write amplification is typically measured by the ratio of writes coming from the host system and the writes going to the Flash memory. Without compression, write amplification cannot drop below 1. Using compression, vendors have claimed to achieve a typical write amplification of 0.5, with best case values as low as 0.14. The two major contributors to write amplification is wear leveling and garbage collection as shown in Figure 15.

Due to the nature of Flash memory’s operation, data cannot be directly overwritten as it can in a hard disk drive. When data is first written to an SSD, the cells all start in an erased state so data can be written directly using pages at a time (often 4–8 kilobytes in size). The SSD controller on the SSD, which manages the Flash memory and interfaces with the host system, uses a logical to physical mapping system known as logical block addressing (LBA) and that is part of the Flash translation layer (FTL). When new data comes in replacing older data already written, the SSD controller will write the new data in a new location and update the logical mapping to point to the new physical location. The old location is no longer holding valid data, but it will eventually need to be erased before it can be written again. Flash memory can only be programmed and erased a limited number of times. This is often referred to as the maximum number of program/erase cycles (P/E cycles) it can sustain over the life of the Flash memory. Single-level
cell (SLC) Flash, designed for higher performance and longer endurance, can typically operate between 50,000 and 100,000 cycles. Multi-level cell (MLC) Flash is designed for lower cost applications and has a greatly reduced cycle count of typically between 3,000 and 5,000. Lower write amplification is more desirable, as it corresponds to a reduced number of P/E cycles on the Flash memory and thereby to an increased SSD life.

The factors affecting the WA value are shown in Table 3.

<table>
<thead>
<tr>
<th>Write Amplification Factor (Helps/Hurts)</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Wear leveling (Hurts)</strong></td>
<td>The efficiency of the algorithm that ensures every block is written an equal number of times to all other blocks as evenly as possible</td>
</tr>
<tr>
<td><strong>Separating static and dynamic data (Helps)</strong></td>
<td>Grouping data based on how often it tends to change</td>
</tr>
<tr>
<td><strong>Sequential writes (Helps)</strong></td>
<td>In theory, sequential writes have a write amplification of 1, but other factors will still affect the value</td>
</tr>
<tr>
<td><strong>Random writes (Hurts)</strong></td>
<td>Writing to non-sequential LBAs will have the greatest impact on write amplification</td>
</tr>
<tr>
<td><strong>Data compression and data deduplication (Helps)</strong></td>
<td>The amount of redundant data that is eliminated before the resulting data is written to the Flash memory</td>
</tr>
<tr>
<td><strong>Garbage collection (Helps)</strong></td>
<td>The efficiency of the algorithm used to pick the next best block to erase and rewrite</td>
</tr>
<tr>
<td><strong>Over-provisioning (Helps)</strong></td>
<td>The percentage of physical capacity which is allocated to the SSD controller (and not given to the user)</td>
</tr>
<tr>
<td><strong>TRIM (Helps)</strong></td>
<td>A SATA command sent by the operating system which tells the SSD what data can be ignored during garbage collection</td>
</tr>
<tr>
<td><strong>Free user space (Helps)</strong></td>
<td>The percentage of the user capacity free of actual user data; requires TRIM, otherwise the SSD gains no benefit from any free user capacity</td>
</tr>
<tr>
<td><strong>Secure erase (Helps)</strong></td>
<td>Erases all user data and related metadata which resets the SSD to the initial out-of-box performance (until garbage collection resumes)</td>
</tr>
</tbody>
</table>

Table 3: Write Amplification Factors
Write amplification, \( A \), is defined as the average of actual write requests.

Figure 16, this is a two-step process. Suppose that \( I \) pages have been rewritten and hence have been invalidated in a block before this block is selected for garbage collection. The block still has \( V \) valid pages, where \( V+I \) are the number of pages that have to be relocated to another block before the block can be erased and reclaimed. In other words, in order to rewrite \( I \) user pages, the number of physical pages that have to be written is \( V+I \). Therefore the write amplification is:

**Equation 1 – Write Amplification**

\[
A = \frac{V + I}{I} = 1 + \frac{V}{I}
\]

In the equation above, the term \( V/I \) is the extra write requests due to relocation of valid pages which we define as follows. The write amplification factor, \( A_f \), is defined as the ratio of the average number of writes used to relocate pages to the average number of free pages gained through the garbage collection procedure. From this definition, the write amplification factor can be written as:

**Equation 2 – Write Amplification Factor**

\[
A_f = \frac{V}{I}
\]

A non-zero write amplification factor means that each user page write causes extra writes to relocate pages, leading on average to a total of \( (1 + A_f) \) page writes. Write amplification deteriorates not only user random write performance, but also endurance. For a strictly sequential write workload, the write amplification factor is zero, i.e., there is no write amplification at all.

**Best Practice - Consider utilizing unmapping features in Flash drives**

One process that differs significantly between Flash and disks is the way that data is deleted. Traditional hard drives delete a file simply by erasing the pointer to that file in the file system. The data still remains stored on the disk, but the operating system knows that chunk of disk space is free to be written to. Eventually, the data will be overwritten. This is not the case with
SSDs. Flash-memory devices require a cell to be empty before data can be written to it. When data is left in cells after being deleted, the program/erase process has to run before new data can be written. This slows down write time. However, there are options available today to address this SSD write penalty. For ATA or SATA interface devices, there is TRIM. When a file is deleted, the operating system initiates the garbage collection process immediately, so those cells will be clean when it is time to write to them. The operating systems that take advantage of TRIM today are Windows 7 and Windows Server 2008 R2 by default and Red Hat 6 as long as you have the Ext4 file system and it is not enabled by default. For SAS and SCSI vendors a similar function called UNMAP has been defined, but limited support as of this writing. In addition many Flash vendors offer a utility that will kick off garbage collection in advance.

**Flash Device I/O Management Overview**

Flash devices are in need of I/O management. NAND-Flash memories have unique characteristics that pose challenges to the SSD system design, especially the aspects of random write performance and write endurance. They are organized in terms of blocks, each block consisting of a fixed number of pages, typically 64 pages of 4 KB each. A block is the elementary unit for erase operations, whereas reads and writes are processed in terms of pages. Before data can be written to a page (i.e. the page is programmed with that data), the page must have been erased. Moreover, NAND-Flash memories have a limited program/erase cycle count.

Flash memory uses relocate-on-write, also called out-of-place write, mainly for performance reasons. If write-in-place is used instead, Flash will exhibit high latency due to the necessary reading, erasing, and reprogramming (writing) of the entire block in which data is being updated. However, relocate-on-write necessitates garbage-collection processes, which results in additional read and write operations. The efficiency of garbage collection could for instance, be improved by delaying those blocks holding data being actively invalidated. The number of read and write operations resulting from garbage collection depends on the number of valid pages in the block.

In contrast to disks, Flash memory blocks eventually wear out with progressing number of program-erase cycles until they can no longer be written. Wear-leveling techniques are therefore used to exhaust the program-erase cycles available (i.e. the cycle budget) of as many blocks as possible, in order to serve the largest number of user writes (or host writes), thereby maximizing endurance. Their performance is measured by the total unconsumed cycle budget.
left when garbage collection can no longer return a free block. Note that retention is another issue that can be addressed by wear leveling as well. Assuming independent and uniformly distributed random short writes, the optimal wear-leveling technique consists of wearing out all blocks over time as uniformly as possible. This can be achieved, for instance, by minimizing the delta between maximum wear and average wear over all blocks. Host writes typically are not uniformly distributed. If a distinction can be made between blocks with static data (i.e. addresses to which the host only infrequently rewrites data) and blocks with dynamic data (with frequent rewrites), wear leveling can benefit from treating these two types of blocks differently instead of wearing them out uniformly. In this case, wear-leveling performance not only depends on the unconsumed cycle budget, but also on the number of cycles wasted by repeatedly moving unmodified static data. In all cases, wear leveling causes additional read and write operations. Therefore, in Flash, write amplification corresponds to the additional writes caused by garbage collection and by wear leveling. Hence, the total number of user writes that can be served depends on the total cycle budget available, write amplification, and the eventual unconsumed cycle budget due to wear-leveling inadequacy. Finally, the management of out-of-place updates involves a mapping between logical block addresses (LBA), i.e. the user (or host) address space, and physical block addresses (PBA). This mapping may be used to distinguish dynamic from static data.

**Flash Transaction Layer considerations**

The Flash translation layer (FTL) hides many of Flash’s remaining challenges. It provides reliability through an abstraction of a uniform block address space. This is necessary since Flash cannot do in-place updates, Flash cells wear out which typically occur between five thousand and one million program/erase cycles depending on the Flash technology. Even read operations can potentially corrupt data!

Through heroic engineering and daunting complexity, the FTL masks these problems, but its performance impact can be significant. Intel’s Extreme SSDs have a read latency of 85 ms, but the Flash chips the drive uses internally have a read latency of just 25 to 35 ms. Flash clearly needs some kind of management and control layer to manage media errors and provide wear-leveling to maximize device lifetime. What interface should that layer present to the rest of the system, and where should it be implemented? Should the SSD give the application control over Flash management, or should applications express their requirements to the SSD?
Best Practice - Consider application characteristics in Flash Architectures

I guess the question is: What applications could exploit a lower-level interface to Flash? One example is “Bloom filters”. Bloom filters store set membership information, with a small chance of false positives in a large bit array (for example, a Flash page). To store an item, the filter computes different hash values, and marks the bits addressed by those values. To test for membership, the filter applies the same hash functions and checks the resulting locations. If all locations are marked, the item might be in the set. Flash device data sheets recommend programming each page of Flash only once between erase operations, to manage the program disturb phenomenon (a program to one page could corrupt data on another page, more on this to follow).

In practice, Flash chips allow multiple programs to a page with the caveat that each program can only turn 1’s (the erased state) into 0’s (the programmed state). Typically, single-level cell Flash devices found in high-end SSDs can tolerate several hundred repeated programs before significant corruption occurs. This combination of characteristics makes these devices a perfect fit for implementing Bloom filters. To insert an element into a filter, the application programs the bit vector for the element onto the page. The program operation performs an effective logical OR with bits already programmed. To maintain data integrity, the application copies the Bloom filter to the next page every 1,000 insertions or so.

Other tricks are also possible. Write-once data encodings can allow applications to write multiple sets of arbitrary logical bits to a single page by encoding each pair of logical bits as three physical bits and ensuring that the second write only changes 1s to 0s. This technique can significantly improve Flash lifetime and energy efficiency.

Several important applications for Flash do not need wear leveling because they are already log-structured as is the case in Google’s log-structured key-value store implementation. In this implementation, using a first-in, first-out replacement policy leads to an elegant log-structured management scheme that is a perfect match for Flash memory. In this case, a FTL would only get in the way. Conventional Flash interfaces and FTLs are innocuous for consumer devices, where performance is less critical. In the data center, however, the story is different. The value of replacing a one-size-fits-all FTL with a refined, mission-specific Flash management system is potentially enormous. The challenge is to define interfaces and abstractions that make Flash easy to manage while exposing Flash’s capabilities and performance potential.
**Best Practice - Consider technologies changing how memory is committed**

Some technologies are being developed, such as ACM (Auto Commit Memory)\(^9\) that can be used to reduce latency by reducing or eliminating block I/O. ACM isn’t just about making NAND Flash storage devices go faster, although it does that too. It is about introducing a much simpler and faster way for an application to guarantee data persistence. For decades, the industry norm for persisting data has been the same. An application manipulates data in memory, and when ready to persist the data, packages the data (sometimes called a transaction) for storage. At this point, the application asks the operating system to route the transaction through the kernel block I/O layer.

The kernel block layer was built to work with traditional disks. In order to minimize the effect of slow rotational-disk seek times, application I/O is packaged into blocks with sizes matching hard disk sector sizes and sequenced for delivery to the disk drive. Most real-world I/O patterns are dominated by small, random I/O requests, but can be a force-fit into 4k block I/O’s sequenced by the block layer to match the characteristics of rotating disks. As shown in Figure 17, note the number of steps in this pathway. This new approach is to designate a certain area within its memory space as persistent, and know that data in this area would maintain its state across system reboots.

This application would no longer have the burden of following the multi-step block I/O pathway to persist that data. It would no longer need smaller transactions to be packaged into 4k blocks for storage. It would just place selected data meant for persistence in this designated memory area, and then continue using it through normal memory access semantics. If the application or system experienced a failure, the restarted application would find its data persisted in non-volatile memory exactly where it was left.

To illustrate, how much faster could real-world databases go if the in-memory tail of their transaction logs had guaranteed persistence without waiting on synchronous block

\(^9\)http://www.t10.org/
I/O? How much faster could real-world key-value stores (typical in NoSQL databases) go if their indices could be updated in non-volatile memory and not block while waiting on kernel I/O?

For example, the traditional approach is as shown in Figure 17:

1. Data is written to DRAM
2. Application calls the OS to write the persisted data
3. OS calls storage driver
4. Storage driver transmits command to IO device
5. DMA transfers data from memory to IO device
6. IO device sends completion
7. Driver sends completion
8. OS send completion

Whereas, utilizing auto commit, the events are:

1. Data written to DRAM area
2. Data Transparently written and persisted to Flash

That is the simplicity of Auto Commit Memory. It reduces latency by eliminating entire steps in the persistence pathway. By reducing the work that the application and kernel I/O subsystems must do to package and route data for storage, applications can be accelerated. The approach is to let the application spend more time processing data in memory, and less time packaging and waiting for that data to arrive at a block storage destination.

**Hot Data vs. Cold Data management in Flash**

For Flash memory, hot data identification has a critical impact on its performance due to garbage collection as well as its lifespan, due to a wear leveling. Although it is an issue of paramount importance in Flash memory, little investigation has been done. Moreover, all existing FTL schemes focus almost exclusively on a frequency viewpoint.
Best Practice - Consider Recency in FTL designs

As it relates to FTL designs, recency—when the data was last used—also must be considered equally with the frequency for effective hot data identification. This can be done with a hot data identification scheme adopting multiple bloom filters to efficiently capture finer-grained recency as well as frequency. In addition to this scheme, a Window-based Direct Address Counting (WDAC)\(^ {10}\) algorithm to approximate ideal hot data identification as a baseline by using a sliding window concept is another consideration.

Recent technological breakthroughs in Flash memory and dramatic reduction of its price enable Flash-based storage systems to hold sway in the storage world. Flash memory does not allow in-place updates (i.e. overwriting). To resolve this issue, as previously discussed, a Flash Translation Layer (FTL) has been developed and deployed to the Flash. The FTL consists largely of an address allocator, garbage collector, and wear leveler all of which are fundamentally based on hot data identification. Therefore, effective hot data identification has a critical impact on performance as well as reliability of Flash-based storage systems. One can classify the frequently accessed data as hot data. Otherwise, they are regarded as cold data. This implementation is still in development stages. Most implementations take only frequency (i.e. the number of appearances) into account. However, there is another important factor, recency (i.e. closeness to the present) to identify hot data.

In general, many access patterns in workloads exhibit high temporal localities; as a result, recently accessed data are more likely to be accessed again in the near future. This is the rationale for including the recency factor in hot data classification as shown in Figure 18. The definition of hot data can be different for each application and can be applied to a variety of fields such as data caching, B-tree indexing in sensor networks, a garbage collection and a wear leveling in

---

\(^{10}\) http://www-users.cs.umn.edu/~park/research.php
Flash memory, and SLC-MLC hybrid SSDs. In addition to these, hot data identification has big potential to be exploited by many other applications. Although this hot data identification is an important issue in Flash memory, it has been least investigated. Existing schemes either suffer from large memory space requirements or incur huge computational overhead. Considering these observations, an efficient hot data identification scheme has to meet the following requirements: 1) effective capture of recency as well as frequency, 2) small memory consumption, and 3) low computational overhead.

**Best Practice - Consider utilizing hot data identification scheme based on multiple bloom filters**

Bloom Filters (BFs) is the key idea of this scheme in that each BF has a different recency weight and coverage so that it can capture finer-grained recency. The advantage of this methodology is:

a. **An Efficient Hot Data Identification Scheme**: A BF can provide computational and space efficiency. Multiple BFs enable finer-grained recency information so that one can achieve more accurate hot data classification. Not only does this require less memory space, it also lowers computational overhead.

b. **A More Reasonable Baseline Algorithm**: An approximation algorithm named Window-based Direct Address Counting (WDAC) adopts a sliding window. Whenever a write request is issued, the LBA is stored in the head of the window and the oldest one is evicted like in a FIFO (First In First Out) queue. WDAC assigns different recency weights to all LBAs in the window according to the closeness to the present. Thus, when a new request arrives, all LBAs are shifted toward the tail of the window and all their recency values are reevaluated. Consequently, WDAC can catch precise recency as well as frequency.

**SSD Vector Interfaces**

The performance of non-volatile memories (NVM) has grown substantially over the last several years: Flash devices today are capable of over 1 million I/O's per second. Unfortunately, this incredible growth has put strain on software storage systems looking to extract their full potential.

**Best Practice - Evaluate vector interfaces to reduce the software-I/O gap**

Vector interfaces organize requests and computation in a distributed system into collections of similar but independent units of work, thereby providing opportunities to amortize and eliminate
the redundant work common in many high-performance systems. By integrating vector interfaces into storage and RPC (Remote Procedure Call) components, a single key-value storage server can provide substantially more requests per second with a median latency below one millisecond, and ensure that vectors of work propagate throughout a distributed system.

**Flash is changing RAID**

It is important to consider the changing times of error correction and drive rebuild methodologies. Flash memory and RAID, at times, do not mix. Given the physical nature of Flash memory with its limited write cycles and certain RAID types that continually write to all drives of a RAID set (RAID 5 or 6 as an example) for protection, it is a best practice to understand the evolving approaches and technologies to address this issue.

**Best Practice - Consider Forward Error Correction (FEC) codes when designing FLASH arrays**

Flash memory is a non-volatile computer storage device which consists of blocks of cells. While increasing the voltage level of a single cell is fast and simple, reducing the level of a cell requires erasing the entire block containing the cell. Since block-erasures are costly, traditional Flash coding schemes have been developed to maximize the number of writes before a block-erasure is needed. A novel coding scheme based on error-correcting codes allows the cell levels to increase as evenly as possible and as a result, increases the number of writes before a block-erasure. The scheme is based on the premise that cells whose levels are higher than others need not be increased. This introduces errors in the recorded data, which can be corrected by the error-correcting code provided that the number of erroneous cells are within the error-correcting capability of the code. The scheme is also capable of combating noise, causing additional errors and erases, in Flash memories in order to enhance data reliability. For added flexibility, the scheme can be combined with other Flash codes to yield concatenated schemes of high rates.
Storage Class Memories (SCM)
What is next for Flash? The answer is Storage class memories. The server uses memory to store information that it will need immediate access to. It uses storage, on the other hand, to store information that it is currently acting on, but does not need at that specific moment. Memory, typically thought of as DRAM, is known for high performance and low capacity and for being expensive. Storage, typically thought of as hard disk drives (HDD), is known for high capacity, affordability, but also for much lower performance than DRAM. “Storage Class Memory” is a form of memory that has capacity and economics that are similar to storage but with performance that is similar to memory. As described above, memory is the place where the processor stores information that it will need immediately. Ideally, there would be enough memory to store all the data that will be needed, but that is not always the case. Again, often thought of as DRAM, memory has challenges other than just cost. In the typical server there is only space to hold enough memory to store a fraction of the total data set the CPU uses. In addition to being expensive, DRAM memory is also volatile, meaning that if a server crashes or loses power, information stored in memory is lost. Because of these factors, most information is kept on storage for the majority of its lifespan.

As a user of Flash storage, there are specific design criteria that should be considered. As it relates to storage memory, which is a class of persistent media, one needs to understand the design triangle as shown in Figure 19. Storage is often thought of as a mechanical HDD that offers near limitless capacity, when compared to DRAM. It is also persistent, meaning that information is not lost if the server crashes or loses power. The problem with hard drives is that in many cases they are unable to provide information to the application quickly enough. Storage Class Memory is a new form of storage created out of Flash-based NAND that can provide an intermediate step between high-performance DRAM and cost-effective HDDs. It can provide read performance similar to DRAM and write performance that is significantly faster than HDD technology. While many Flash-based storage devices can offer compelling economics versus DRAM, to be considered an acceptable alternative to DRAM memory it must provide near-RAM performance. Simply putting any SSD into a server may not accomplish this goal. For most servers, an upgrade to SSD in almost any form, will provide some incremental performance improvement. Again this is when compared to HDD performance, not RAM performance.

Most Flash-based devices will connect to a server via a traditional storage protocol (SATA or SAS for example). Unfortunately, while this connection provides for easy installation, it can also become the biggest bottleneck to maximizing performance. When looking for a Flash-based
device that will provide memory-class performance, in other words Storage Class Memory, the design triangle shows us that speed, write endurance, and cost are the three trade-offs when weighting factors of storage vs. memory. A Storage Class Memory device made out of Flash will need to perform at the full capabilities of the Flash module in both multi-threaded application environments and single threaded application environments. Companies such as Fusion-Io and Verident are designing PCIe-based solid-state storage that is memory class. By leveraging a single controller that has access to all the available Flash modules it can provide high-performance in all situations.

Storage Class Memory fills an important gap between DRAM memory and HDD storage. It provides near-DRAM performance at significantly lower costs and significantly higher capacities. However, when Flash memory is positioned as a viable alternative to DRAM, the expectations of that device are higher. It cannot be bottlenecked by legacy connectivity or require unusually high levels of application threading. Finally, it should be built around a flexible Flash management architecture that allows for rapid design and the ability to leverage the investment that vendors like Intel have made in the advancement of CPUs.

SCM is blurring the line between RAM and persistent large capacity media. NAND Flash is getting pretty close to SCM, but not there yet. The many competing technologies for SCM are:

1. **Phase Change RAM** This technology is the most promising now, since it can scale.
2. **Magnetic RAM** This technology is used today, but poor scaling and not very space efficient.
3. **Magnetic Racetrack** This is in the basic research stage, but very promising long term.
4. **Ferroelectric RAM** This technology is used today, but poor scalability is limiting its viability.
5. **Solid Electrolyte and resistive RAM (Memristor)** This technology is in early development, but could be promising.
6. **Organic, Nano particle and polymeric RAM** Many different devices in this class, unlikely a near term viable solution.
7. **Improved FLASH** It seems the design triangle outlined above continues to hold true. Flash continues to be relatively slow on writes and continues to have poor write endurance.
Flash in the Data Center – How does it all fit?

Now, let us try to sum it up in terms of how Flash technology, from an architectural perspective, addresses all that has been outlined so far. Flash technology offers new ways to bring performance to the application. It is important to understand the decision points along the way, and as such, utilizing a decision tree to understand when it is appropriate to include Flash technology in the server, network, or storage makes a lot of sense.

Figure 20 depicts a flow chart of the questions that need to be answered while navigating the data center Flash world.

**Decision point 1:** First, Where do you put Flash? Is it in the server, network, or storage? For the server, the storage model is DAS (Direct Attached Storage) which typically is not shared between servers. Considerations on how you handle clusters needs to be addressed. It can be done, but it is complicated. This configuration has the lowest latency, (read latency 20--200us) and the network roundtrip latency is approximately 10--100us. Relevant apps are real-time trading and data analytics, but for most apps, network latency is masked by queuing. It has scalable bandwidth and you do not need an expensive network or scale-out storage. For most apps though, throughput is not bound by network bandwidth, but by the queue depth as shown in Figure 21 (latency and throughput vs. queue depth). As the queue depth increases, DAS and SAN latency are equivalent and throughput remains constant.

![Flash in the Data Center Architectural Decision Tree](image)

**Figure 20 - Decision Tree**

![Figure 21 - QD Dependency](image)
For the Server area network, a Flash appliance used in the infrastructure offers the best shared storage with a low latency transport method and utilizes RDMA technology, which bypasses the OS and TCP/IP stack. The options on the transport-side are InfiniBand and RoCE. As discussed in previous sections, InfiniBand has the inherent messaging architecture and is the standard in the HPC world. RoCE has the advantage of ubiquitous Ethernet connectivity leveraging the existing IP infrastructure.

SAN storage, that we all know so well, is the other shared solution offering all the advantages associated such as high availability, replication, and DR, but in addition, it supports multi-host distributed applications, file service, and VM migration from one host to another. Note that this can be done with DAS with host-to-host access, but again, it is complicated and one loses the proximity advantage of DAS such as network partitions. As discussed, it offers high availability, which remains available when a server host goes down. Only Flash in storage can accelerate the storage block map at a minimum and quite often, a small amount of Flash in storage has a big impact (i.e. Block map. as mentioned). Also, note that in most cases, Flash in the host does not preclude Flash in storage.

**Decision point 2:** In terms of evaluating a Flash only vs. a Hybrid Flash and disk solution as shown in Figure 22, this type of trade off decision needs to be done. In the case of Flash only, it guarantees high performance and lowers the worst-case latency, not just expected latency. One exception is possibly the period during a garbage collection process. Note that garbage collection can be mitigated through software and higher performance SSD drives. Relevant apps include stock trading, medical, and HPC. This solution is simpler. Flash and disk have different performance and failure traits. There are no moving parts, so it tolerates vibrations. Relevant apps include military and aerospace to name a few. However, with advances in global and in-line DE duplication, using Flash is now an option without loss of performance given Moore’s Law of CPU performance doing the DE duplication.

As a result, total cost of ownership for this solution can be very attractive. In terms of a hybrid solution, Flash is expensive, enterprise high-density HDD is ~$0.10/GB (SATA, low cost) and commodity MLC SSD are ~$1.50/GB (15x). Enterprise MLC SSDs are ~$3/GB or 30x in price. It
requires sophisticated controllers and firmware and over provisioning to reduce write amplification and it is not clear how fast the Flash to disk price ratio will fall. Also note that with increasing density, poorer performance and reliability is incurred, as previously discussed. Data reduction helps, but compression works on both disk and Flash (1.5 to 3x). Deduplication is easier on Flash given its fast random I/O characteristics, but less dependable. In many applications, much of the data is cold, for example, old email, stopped VMs, and inactive test/dev databases. Data protection is based on snapshots and replication. In addition, there is uncertainty of Flash reliability. Peculiarities with write-endurance, retention, and read-disturb continue. This can be mitigated, but at a further expense. The real question long term is, will Flash be the right solid-state replacement for disk? More on that later.

Decision point 3: In a hybrid solution, Flash can be used as the endpoint (resting point, tiering or a peer to disk) or as the accelerator (a cache or buffer to disk) as shown in Figure 25. First, it’s simpler to use Flash as a peer to conventional disk. If you mask the technical differences and just use access and response time differences, it is a simple integration process for storage vendors. However, automatic tiering is complex, since you have to be constantly computing when to promote and demote as shown in Figure 23. Another advantage is it avoids duplication of data on Flash and disk. One disadvantage of an endpoint resting architecture is migrating data between endpoints is slow, which causes the following: Changing home location is expensive. Promoting to Flash requires demoting to disk. As a result, bigger cost demands a bigger benefit TCO. It can take hours or days to determine if data as hot or cold. One requires migration of data in large chunks. This methodology encourages “pinning” the entire data set in Flash as well as sizing Flash tier conservatively given its limited capacity. Copying data and evicting for acceleration is fast, as shown in Figure 24, and the data locality in Flash is temporary. Copying data to Flash does not require additional disk access.
**Decision point 4:** In terms of using Flash as an accelerator, either as a cache or buffer for disk, the next option to consider is if the Flash is a write buffer utilizing “write back caching” or read cache utilizing a “write through or write invalidate”. When Flash is used as a write buffer, one achieves low-latency writes. Disk latency is 5ms, while Flash latency is from 200 to 600us (SLC or MLC). Note that NVRAM (DRAM + power protection) is faster with 50us to 50ns access times. Also note that NVRAM does not burn out (battery powered). Write caching can engross large bursts of writes, while Flash supports larger buffers than NVRAM. Also, note that sustainable throughput is limited by the size of the drain to disk as shown in Figure 26. This methodology also supports a greater opportunity to absorb overwrites as well as opportunity to re-sort writes. It improves sequentiality of drained data, preserving locality, but limited by consistent checkpoints on disk. Flash as a read cache allows control over Flash wear by inserting only cache-worthy data in Flash, which results in the ability to throttle cache insertions based on remaining life heuristics. This method also tolerates unreliability of Flash gracefully. Data in cache is a subset of data on disk (all “clean”) and can checksum, verify, and discard on failure so one does not need high endurance Flash, and as such, no need for redundancy (parity or mirroring). Note that dual parity incurs approximately a 20% overhead.

**Decision point 5:** When write through/write invalidate is considered the best option, the next question is whether one wants an optimized read (write in place) or write (write coalescing, no locality writes as one chunk) layout. A read-optimized layout preserves sequential locality on disk and as such, allows fast sequential reads, but does not help with random reads. A smaller map of logical addresses to physical locations is required, saving space and complexity as well as supporting a common unit of mapping (RAID stripe). One should consider a write-optimized disk layout since, for many applications, the I/O trend to storage targets is increasingly dominated by writes, since more read hits occur down the Flash proximity stack. Large caches in host/storage reduce reads greatly. As noted in a paper by Rosenblum in 199111, increasing memory

11 http://www.cs.berkeley.edu/~brewer/cs262/LFS.pdf
sizes will make the caches more and more effective at satisfying read requests. As a result, disk traffic will become dominated by writes and as a result, buffers will not reduce writes much as shown in Figure 27. Coalescing random writes leverages sequential throughput of disk increasing performance. Redirect-on-write snapshots produces a no-copy-on-write overhead since Flash does not do write DIP (Data in Place) and as such, Snaps are relatively easy. Writes into free space (writing in the holes) become fragmented over time, reducing performance even if opportunistic coalescing based on hole size is implemented. NetApp’s WAFL and IBM’s ZFS are examples. Writing in large stripes (i.e. log-structured writes) and GC (Garbage Collection), sweeps holes into full stripes. It also supports variable block size: compression as well as preserving write-order locality for reads since it if a block is written together, it will be most likely be written together. Therefore, to summarize, quite often the most efficient solutions are a combination of various technologies that when designed properly, will address most application requirements utilizing Flash.

The dark side of Flash

Many believe that the core Flash technologies SLC and MLC will continue to get ever faster, cheaper, denser, reliable, and more cost effective. Flash is a great semiconductor technology. However, in contrast to other semiconductor technologies such as Complementary Metal–Oxide–Semiconductor (CMOS), for constructing integrated circuits, microprocessors, micro controllers, static RAM, and other digital logic circuits, moving to smaller geometries for NAND Flash has its down side as shown in Figure 28. Normalized to semiconductor technologies advancing over time, write and read latency increases while write and read throughput decreases. We are heading in the wrong direction! In addition, it turns out that Flash NAND devices has detrimental attributes in areas defined as NAND Error Modes. They are:

---

1. [cseweb.ucsd.edu/users/swanson/papers/FAST2012BleakFlash.pdf](cseweb.ucsd.edu/users/swanson/papers/FAST2012BleakFlash.pdf)
1. **Program Disturb** – When cells not being programmed receive elevated voltage stress when data is not changing but is in a block being programmed. Disturbed bits can be managed with error correction codes (ECC).

2. **Read Disturb** – When cells not being read receive elevated voltage stress when always in the block being read. Charge collects on the floating gate causing the cell to appear to be weakly programmed. Again, disturbed bits can be managed with ECC. A rule of thumb for excessive reads per block between ERASE operations is SLC – 1,000,000 READ cycles and MLC – 100,000 READ cycles before possible error.

3. **Data Retention** – When a charge loss/gain occurs on the floating gate over time, the device threshold voltage trends to a quiescent level. However, the cell is undamaged and the block can be reliably erased and reprogrammed as shown in Figure 29, describing retention required vs. block cycles.

4. **Endurance** – When issuing program/erase cycles, charge is trapped in the dielectric, causing a permanent shift in cell characteristics and not recovered by erase. It is observed as a failed program or erase status. Blocks that fail should be retired (marked as bad and no longer used).

**Best Practice - Consider the various design principles previously discussed to address Flash warts**

It is important to consider the target data-error rate for your particular system. Understand the use model that you intend for your system. Design the ECC circuit to improve the raw-bit error rate (BER) of the NAND Flash, under your use conditions, to meet the system's target BER. Also consider utilizing Flash drives that implement "managed NAND" methodologies such as Micron’s eMMC (enhanced multimedia card), as shown in Figure 30. The “Direct NAND Interface” is the current methodology as shown in Figure 31, but a more efficient and better performing solution is with eMMC. eMMC is backward compatible with
previous MMC (multimedia card) systems. As previously mentioned, there are ways to get around this such as wear leveling, ECC, and other block management methodologies as discussed in previous sections, but the bottom line is, it is going to get worse, not better.

The technology trends described put SSDs in an unusual position for a cutting-edge technology. SSDs will continue to improve by some metrics (notably density and cost per bit), but everything else about them is poised to get worse. This makes the future of SSDs precarious. While the growing capacity of SSDs and high IOP rates will make them attractive in many applications, the reduction in performance that is necessary to increase capacity while keeping costs down will possibly make it difficult for SSDs to scale as a viable sustainable technology for some applications.

![Figure 31 - Direct NAND](image)
Conclusion

Spinning disk, with the dilemma of the Trilemma wreaking havoc across the data center universe. There must be another way. The way is Flash.

With what was discussed, we are now using tools and methodologies based on Flash technology and have now realized how this new technology is pivotal to changing data center platforms for the better.

We found that when addressing the data center, one needs to address the server, network, and storage. Best practices, methodologies, and use cases for this new Flash technology have been discussed. Also discussed is how Flash can be used in various ways and how Flash fits in various storage, network, and application architectures.

Flash does have a dark side, but the intent to move from a spinning mechanical disk to a semiconductor solution, ultimately, will be the correct decision. The current incarnation of persistent semiconductor technology—NAND Flash—may not be the ultimate destination, but for today, it’s the best technology we have.
Appendix B – References

[3] The Bleak Future of NAND Flash Memory, Laura M. Grupp†, John D. Davis‡, Steven Swanson†, Department of Computer Science and Engineering, University of California, San Diego, ‡Microsoft Research, Mountain View
[14] CAFTL: A Content-Aware Flash Translation Layer Enhancing the Lifespan of Flash Memory based Solid State Drives, Feng Chen Tian Luo Xiaodong Zhang, Dept. of Computer Science & Engineering, The Ohio State University Columbus, OH 43210, USA


Author’s Biography

Paul Brant is a Senior Education Technology Consultant at EMC in the New Product Readiness Group based in Franklin MA. He has over 29 years’ experience in semiconductor design, board level hardware and software design, as well as IT technical pre-sales solutions selling as well as marketing, and educational development and delivery. He also holds a number of patents in the data communication and semiconductor fields. Paul has a Bachelor (BSEE) and Master’s Degree (MSEE) in Electrical Engineering from New York University (NYU), located in downtown Manhattan as well as a Master’s in Business Administration (MBA), from Dowling College, located in Suffolk County, Long Island, NY. In his spare time, he enjoys his family of five, bicycling, and other various endurance sports. Certifications include EMC Proven Professional Cloud Architect Expert, Technology Architect, NAS Specialist, VMware VCP5, Cisco CCDA, and Certified Technical Trainer CTT+, CompTIA Security +.