Modeling and Simulating Flash based Solid-State Disks for Operating Systems

Kaoutar El Maghraoui, Gokul Kandiraju, Joefon Jann, and Pratap Pattnaik
IBM T.J Watson Research Center
P.O.Box 218, Route 134
Yorktown Heights, NY 10598
{kelmaghr,gokul,joefon,pratap}@us.ibm.com

ABSTRACT

Solid-State Disks (SSDs) made out of Flash devices have gained a lot of prominence in recent years due to their increasing performance and endurance. A number of mechanisms are being proposed to improve the performance and reliability of these devices from technological and operating system perspectives, to integrate them into personal computers and enterprise systems. Most of such proposals are being implemented and evaluated directly on top of these SSDs and require sophisticated framework and infrastructure for thorough performance evaluation. On the other hand, to our knowledge, very little has been done on modeling Flash devices and building efficient Flash simulators that can be used to simulate SSDs. Such models and simulators can give insights to make design decisions, save a lot of cumbersome work for setup and implementation, save hardware costs and allow researchers to focus on the real methods that are being proposed.

This paper presents a linear model for NAND-based Flash devices based on the internal architecture of these devices. Parameters of the model are presented along with micro-benchmarks that can be used to extract these parameters. The model is validated on the STEC Zeus Flash SSD and extracted parameters are used to build a Flash simulator as a kernel extension in the AIX operating system. A key feature of the simulator is that it simulates I/O requests by maintaining minimal state information and is independent of the internal organization of a Flash SSD. The simulator is validated using commercial and raw-IO applications through experimentation on the simulator and real Flash disks.

Categories and Subject Descriptors

D.4.8 [**Software**]: Operating Systems—*Performance*; D.4.8 [**Software**]: Operating Systems—*Storage Management*

General Terms

Performance Measurement

Keywords

Solid State Disks (SSD), NAND Flash Memory, Simulator, Modeling

1. INTRODUCTION

Flash memory is rapidly becoming an important and promising technology for the next-generation storage due to a number of reasons including its (i) low access latency (ii) low power consumption (iii) higher resistance to shocks (iv) light weight and (v) increasing endurance. A lot of research done in the past decade has focused on improving the performance and reliability of Flash devices and software [7, 5, 9, 12, 19, 18, 10]. Flash devices can be made out of NAND or NOR technologies. NAND-based Flash devices have emerged as a more acceptable candidate in the storage market. Research has been conducted on improving NAND Flash technology [29, 27], Flash organization [15, 10], increasing endurance [7], optimizations to access data at finer granularity [26] and software optimizations and improvements [19, 6].

Today, SSDs built on NAND Flash are not only being shipped as part of embedded systems but also as part of personal computers [25] and enterprise systems [30]. This recent widened and increased usage of Flash has strongly driven research to design reliable systems using Flash. Typical approach has been to use an array of Flash devices for higher performance and better endurance. While a lot of work for SSDs has focused on building SSDs using Flash devices and increasing the lifetime of SSDs, little has been done on operating system design for Flash. Recent work that focuses on the OS aspects of SSDs includes design of Flash file-systems [8, 12] and algorithms for disk scheduling [19]. Typical investigation of OS issues for Flash involves designing, implementing and evaluating new algorithms and methods on top of SSDs. In particular, this involves detailed perusal of specifications of an SSD, writing device drivers for it (or studying a shipped device driver in detail), changing the device driver and OS to incorporate newly designed algorithms, and a detailed performance evaluation using enterprise workloads. This not only requires expertise in OS and device-driver development but also a detailed knowledge of the specification of Flash devices. Therefore, designing, implementing, testing and evaluating new OS algorithms and techniques on top of real SSDs is a cumbersome task.

A promising alternative to this is the usage of modeling and simulation. Building models for system components and using simulators for system design in not new. One can quote numerous examples where modeling techniques and simulators have been extensively used for designing systems and system components such as processors, memory, hard disks, network interfaces, network topologies, etc. Simulators not only speed up the process of design and development but also provide insights to make decisions that can later be implemented and evaluated in real environments. Models give insights and intuition into behavior of system components and can save a lot of design and implementation efforts. Building a simulator for a Flash device also greatly reduces the wear and tear of it, since Flash has limited endurance for writes. During the design process, one could undertake a complete state space simulation of methods in the simulator and then evaluate only a few chosen ones on the real Flash disk. This paper presents a model for a Flash device and a novel way to build a Flash simulator using the model. The simulator can be used to test new techniques and algorithms - *both for OS and for Flash itself.*

Simulating Flash memories is a non-trivial task. A typical Flash memory is made up of pages (typical page size is 2KB or 4KB). A set of pages (usually 64 pages) forms a block and a set of blocks forms a plane. Several planes (power of 2) form a die, and several dies form a chip. An array of such Flash chips with a controller forms a Solid State Disk. For accurate simulation, one not only needs to know the internal organization of an SSD, but also the details of the associated adapter such as caching/pre-fetching effects. It is also to be noted that some of the Flash operations can happen in parallel (when there are multiple planes), depending on the size of the incoming requests. In addition to all of the above, Flash devices map reads/writes from the OS to read-/erase/program operations that are supported by the Flash chip. Therefore, the timing of OS read or write operations varies significantly from request to request depending on the size and the address of the request. While reads can be done at page granularity (typically in tens of microseconds), writes may involve both page-level programming (couple of hundreds of micro-seconds) and block-level erasing (usually in the order of milliseconds). This is because a write may involve erasing a block and then programming the block page by page. Thus, simulating Flash devices accurately requires maintaining state information for each page and logical-to-physical mappings for all the blocks in the device. Flash devices also maintain additional 'overhead' area for each page that keeps track of the erase counts and CRC (for error correction). Simulating erase counts is necessary if one is interested in investigating endurance optimizations for the Flash devices. On top of this, recent Flash vendors have also provided ways to do sub-page programs [26] and such optimizations result in an additional level of complexity for effective Flash simulation. Also, as Flash technology evolves rapidly, a simulator that bases itself on the internal organization of the Flash chip could get outdated quickly.

We present a model and a novel way of simulating a Flash device. The model is constructed using internal operating details of a Flash device but the simulation itself is independent of the Flash device and the adapter characteristics. Since Flash devices are randomly accessible, and do not have seek and rotational delays, the I/O time for a request is fundamentally dependent on the size of the request (in addition to other queuing delays in the drivers and controller). Also, since Flash devices might have to erase on a write, I/O time also depends on the operation of the request. We develop a model for the timing of a request and present a method to efficiently simulate delay for an incoming I/O request based on its (i) size (ii) direction of I/O and (iii) sequentiality. We maintain minimal state information about the requests themselves in doing so (the only information we maintain is the previous request block number, its size and direction of I/O). We validate the model on STEC Zeus Flash drives and extract its model parameters using micro-benchmarks that we designed. We then build a Flash simulator using the extracted parameters and finally validate the simulator using raw and commercial workloads.

The rest of the paper is organized as follows: Section 2 gives the background and basics of Flash-based SSDs. Section 3 discusses related work. Section 4 presents the model and parameters' extraction process. Section 5 presents the design of the Flash simulator and its implementation using the model parameters. Validation results for the model, and experimental evaluation of the simulator for raw-IO and commercial workloads are presented in Section 6. Finally, concluding remarks, future work and key contributions of this paper are discussed in Sections 7 and 8.

2. BACKGROUND

Flash memory has been traditionally used in portable and mobile devices (USB storage devices, hand held devices, etc.) because of its unique and attractive features: shock resistance, small size, low power consumption, and low latency [11]. More recently, this technology has made huge strides into the server and personal computer storage space in the form of SSD with the intention of replacing traditional hard disk drives (HDD). SSD devices are non-volatile storage devices based on the NAND Flash memory type. NAND-based memory is a solid-state memory that allows the storage of persistent data. A key feature of SSD devices is the lack of any mechanical moving parts compared to HDD. SSD have no seek time, which are inherent in conventional disks. Therefore, they can provide a much faster, and a more uniform random access speed compared to HDD.

2.1 Organizational Layout of a Flash chip

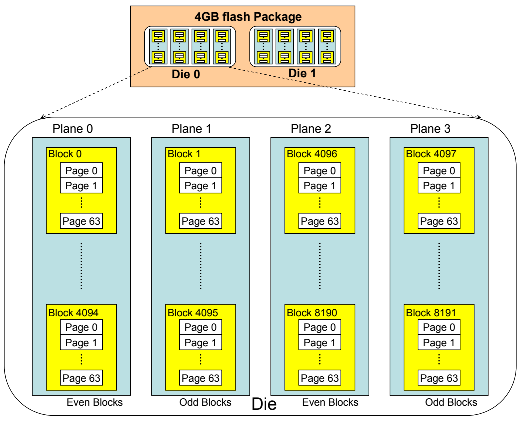

Figure 1: Internal Organization of the Samsung Flash Package.

A Flash chip is an electronic device that can be erased and reprogrammed electronically. It is of type EEPROM (Electrically Erasable Programmable Read-only Memory). Data is stored in Flash memory in an array of memory cells. There are two types of cells: single-level cells (SLC) and multi-level cells (MLC). In SLC, each cell can store only one bit of information. While in MLC, multiple bits can be stored in one cell location using multiple levels of electric charge. Each bit in SLC technology has a default value of 1. It must be programmed to take the value of 0. MLC chips are less expensive than SLC chips. However they have a shorter life span and are slower than SLC chips (an MLC can fail ten times more often than an SLC [16]). Secondary storage Flash devices are all based on NAND Flash. As explained earlier, a typical NAND-Flash package is composed of a number of dies, each containing an even number of planes which itself comprises of several blocks that are typically made up of 64 pages. Each page reserves some byte region to store metadata (error detection and correction checksum). A plane contains either odd or even blocks. Figure 1 shows the organizational layout of a Samsung 4GB Flash package. Other Flash vendors like Mircon and Hynix have similar layout.

2.2 Behavior of Flash SSD

Because of the absence of mechanical parts in Flash chips, their random read performance is almost as good as the sequential read performance. This is one of the major benefits that Flash SSD has over HDD. Reading is done at the granularity of a page. A Program operation is also done at a page granularity. Within the same blocks, page programs are performed sequentially. Erase operations on the other hand can only be done at the block level (typically 64 pages). A block erase sets all the bits in the block to 1. So any time a bit has been set to 0, changing this bit back to 1 requires erasing the entire block that it belongs to. An erase operation is also expensive (typically 1.5-2 ms to erase a block). Due to this major limitation, the cost of write performance can vary depending on whether the operation requires an erase operation or not. Another important aspect of Flash SSDs is the wearing behavior. Flash memory has a finite number of erase-write cycles. Most commercially available SLC-based products guarantee a life-span of about 10^5 write-erase cycles. Wear-leveling algorithms are used inside Flash-controllers to spread the erase operations across the Flash device in an attempt to increase its lifespan.

To hide the expensive erase operations and create abstractions for an in-place write, SSDs have an integrated controller that implements address translation, garbage collection and and wear-leveling algorithms in a software layer called the Flash translation layer (FTL) [19]. The FTL emulates a block device with the standard 512-byte sectors so that unmodified files systems can run on top of the Flash SSD just as they run on top of regular block devices. The FTL is responsible for mappings between blocks in the micro-controller's RAM and Flash pages, and for managing bad blocks. Several FTL schemes have been implemented such as paged-based, block-based, FAST [24], and DFTL [15].

While the basic principles of operation of Flash and the FTL layer are well understood, the various vendor design decisions and the implicated performance trade-offs are not documented. Flash vendors keep their internal Flash FTL algorithms unknown to the public. Moreover, the asymmetric performance of read and writes (writes are about 4-5 times slower than reads), the poor write performance, the wearing behavior of Flash memory, the need for the FTL layer to balance writes and manage bad blocks makes the understanding and modeling Flash SSD a challenging and complex task.

3. RELATED WORK

Numerous research efforts have attempted to understand the behavior of Flash SSDs and their overall performance in secondary storage [6, 28, 23]. New file systems optimized for the properties of Flash SSDs have been proposed [8, 9, 12], and efforts have also focused on improving the technological aspects of Flash SSDs with algorithms and data structures suited for the various operations on Flash devices such as wear-leveling algorithms, improving the write performance, and bad block management [13].

So far, very little work has been done to build a NAND-Flash SSD simulator capable of simulating various Flash SSD devices. At Microsoft Research, Agrawal et al. [5] built a NAND-Flash simulator based on the DiskSim simulation environment [14] from the CMU Parallel Data Lab. DiskSim is an open source disk simulator that emulates a hierarchy of storage components such as buses, controllers, and disks. It is driven by externally-provided I/O request traces or internally-generated synthetic workload. Agrawal et al's SSD simulator extends DiskSim by adding SSD features such as SSD latencies, multiple request queues, logical block maps, block erasure, and wear-leveling. This simulator has been built as an ideal SSD simulator and does not simulate a particular vendor's device. It also simulates only the page-based FTL scheme. CPS-SIM [22] is another Flash SSD simulator that is limited by a single FLT scheme. FlashSim [31] has an object-oriented design and supports simulating multiple FTL schemes using workload traces. FlashSim uses also DiskSim to simulate queuing effects.

Modeling a Flash device has received far less attention. In fact, to our knowledge, there is no prior work that develops models for Flash devices (work in this direction has been more focused on developing circuit level models [20]). In this paper, we propose a throughput model based on the Flash architecture and use it to develop a simulator. Compared to the discussed simulators, our simulator is capable of simulating several Flash SSD devices because its parameters can be extracted from any SSD. Also, our simulator is a kernel extension that can be configured on a running OS as a paging device. Hence, it does not require any traces as input. Therefore, it is a good choice to test (i) the Flash behavior of scientific and commercial applications and (ii) the OS policy changes for Flash devices.

4. MODELING A FLASH DEVICE

This section describes the working of a Flash device in detail and presents a model for its operation.

4.1 Flash Commands and Operation

4.1.1 Internal Architecture

Flash devices multiplex data, commands and addresses to the same I/O pins. The I/O control logic is able to distinguish data and addresses based on the command that is in progress. Figure 2 shows a functional overview of internal Flash architecture. Commands are received by the I/O Control logic which are latched into a command register to generate internal signals. Based on the command, addresses are latched (in consecutive cycles) and are sent to row and column decoders. Data transfer to/from the NAND Flash array is byte by byte through the data and the cache registers. The data register is closest to the memory array and acts as a data buffer for the NAND Flash memory array operation whereas the cache register is closest to I/O control circuits and acts as a data buffer for the I/O data. The granularity of read and program operations is a page and the granularity of erase is a block. During normal page operations, the data and cache registers are tied together and act as a single register. During cache operations, the data and cache registers operate independently to increase data throughput. (*Note: The above discussion is adapted from* [26])

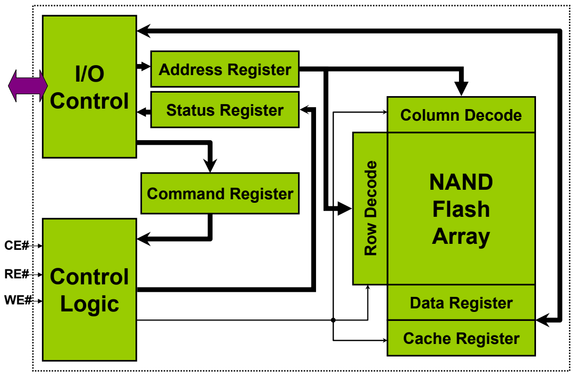

Figure 2: Functional Block Diagram of a Flash Device(Source: [26]).

4.1.2 *Read and Write Operations*

A read operation is triggered when the READ command gets latched into the command register. Read of a single page involves movement of the page from the array cells to the data register (typically tens of microseconds) followed by data output from the data register to the I/O bus. The data output rate is a device characteristic and is typically of the order of tens of nanoseconds per byte. Read of multiple pages occurs in 'cache mode' where data is first transferred to the data register and then moved to the cache register (movement happens in just a few microseconds). The cache register outputs the data directly onto the I/O bus as the next page is simultaneously transferred from the NAND array to the data register.

Writes occur in a similar fashion. Write operations involve only PROGRAM commands if sequential pages within the same block are being written. Programming a single page is of the order of couple of hundreds of microseconds (220μs for devices described in [26]). However, writing a random page (i.e., programming a random page) might require erasing a whole block of pages using the ERASE command which could take up to 2 milliseconds.

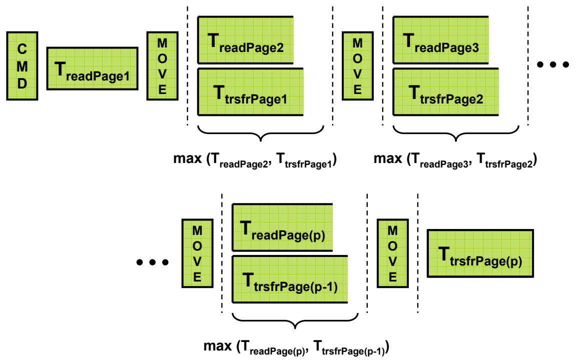

Figure 3: Timing Sequence of a Read Operation.

4.2 Modeling Reads and Writes

Based on the operation details of the Flash device described above, this subsection builds a model for the read-/write operations.

4.2.1 *Sequential Reads*

Let the time to read a page from the NAND array (memory cells) to the data register be T_{read}, the bus transfer time for the page (from the data or the cache register) be T_{trsfr}, the time to move the page from the data register to the cache register be T_{move} and the time to issue a command be T_{cmd}. Now, time to read one page from the device is given by

$$T_{OnePage} = T_{cmd} + T_{read} + T_{trsfr} \tag{1}$$

For sequential reads, Flash devices operate in cache-mode as explained above. So, time to read two pages in the cache mode would be

$$\begin{aligned} T_{twoPages} = \; & T_{cmd} + T_{readPage1} + T_{movePage1} \\ & + max[T_{trsfrPage1}, T_{readPage2}] \\ & + T_{movePage2} + T_{trsfrPage2} \end{aligned} \tag{2}$$

Note that in cache mode, the last page does need to be moved to the cache register even if there is no subsequent page for transfer [26]. Similarly, it can be derived that the time to read p pages would be

$$\begin{aligned} T_{pPages} = \; & T_{cmd} + T_{readPage1} + p * T_{move} \\ & + (p-1) * max[T_{trsfrPage}, T_{readPage}] \\ & + T_{trsfrPagep} \end{aligned} \tag{3}$$

Figure 3 shows the timing sequence for reading p pages. Note that all the terms in equation 3 are constants except p. In other words, time to transfer p pages (or equivalently n bytes, as page size is a constant) can be written as

$$T_{nBytes} = A + n \times B \tag{4}$$

where A and B are constants for the sequential read pattern that depend on chip characteristics (they will also include associated software overheads in the OS when determined through experimentation). Therefore, sequential read throughput for requests of size n bytes assumes the form:

$$Throughput_{sr}(n) = \frac{n}{A_{sr} + n.B_{sr}}$$

for constants A_{sr} and B_{sr}.

We determine constants A_{sr} and B_{sr} using experimentation and feed these into the simulator (described later in Section 6.1).

4.2.2 *Random Reads, Sequential Writes and Random Writes*

A random read of a single page does not involve the cache register. The total time to read is a constant and independent of the page number (unlike hard disks, where the distance between consecutive requests matters due to seek and rotational delays). Therefore, throughput for a random read of a single page is a constant. However, random reads of larger block sizes lead to a similar timing sequence as described above (request itself would be at a random block number in the SSD but large requests of multiple pages would be handled in cache mode). Random writes also exhibit similar behavior to reads during the program operations but erase operations involve additional block erase overhead.

Therefore, we stipulate that sequential writes, random reads and random writes would only change the constants of the above presented linear model. In summary, we model read and write throughputs (for both sequential and random patterns) according to the equation

$$Throughput_x(n) = \frac{n}{A_x + n.B_x} \qquad (5)$$

and determine the constants A_x and B_x using experimentation. Note that the subscript x is used here for generality. x will assume four different forms resulting in eight different constants for our model: (A_{sr}, B_{sr}) for sequential reads, (A_{rr}, B_{rr}) for random reads, (A_{sw}, B_{sw}) for sequential writes and (A_{rw}, B_{rw}) for random writes. *These constants constitute the parameters of this model, which we call the throughput model.* For the rest of the paper, these constants will be interchangeably called as *Throughput Parameters* or *Model Parameters*.

4.3 Throughput Model Parameter Extraction

This section describes the parameter extraction process and the micro-bechmarks that we used to extract parameters required for the model. Our micro-benchmarks do not need any details about the internal organization of a Flash chip or the adapter. In fact, they only need one parameter: *Total Size of the Flash Device.*

4.3.1 *Measuring the Throughput*

The entire device (SSD) is divided into $(2z+1)$ equal *zones* (odd number of zones). Starting from the second zone, we generate sequential and random accesses in every alternate zone for z number of zones. For example, we first generate sequential reads in a zone (for a given block size) and measure the total time. Using the number of requests generated, the request size (block size) and the total time, we compute the throughput for this zone. This would be the *sequential read throughput for this zone for this block size.* We then move to the next (alternate) zone and continue a similar process. Once we are done with all the zones, we take the average of all the throughputs that we have from every zone. This will be the *sequential read throughput for this block size for the device.* We do a similar process for sequential writes, random reads and random writes. For random accesses, we use a uniform random number generator. In all our experiments we have used the value of z=3 (seven zones). Figure 4 shows the high-level view of the extraction process. We then repeat the process for various block sizes until we reach a saturation point where block size does not have further effect on the throughput.

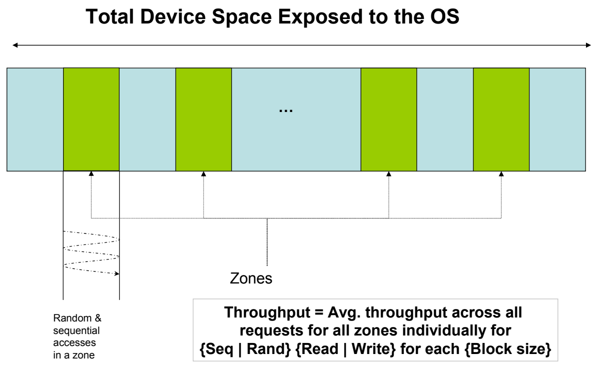
Total Device Space Exposed to the OS
...
Zones
Random & sequential accesses in a zone
Throughput = Avg. throughput across all requests for all zones individually for {Seq | Rand} {Read | Write} for each {Block size}

Figure 4: Parameter Extraction Process.

The basic idea here is that we want to exercise different areas of the device to prevent I/O requests from stressing only a particular set of blocks or pages (or being limited to a singe plane). At the same time, we do not want the extraction process to be very time consuming and insensitive to the wear and tear of the Flash device (due to the limited write endurance it has). Therefore, we partition the entire device into *odd* number of zones and exercise alternate zones. Designing micro-benchmarks that minimize the number of writes of a Flash device and effectively extracting device/-model/throughput parameters in minimal time itself can be a research problem and its detailed investigation is beyond the scope of this paper. In this paper, we use this simple technique and extract the throughput values. Note that one of the reasons for this method to be effective is that Flash devices are randomly accessible. The 'inter-request' distance does not matter so much as the block size. Yet, it is important to stress different areas of the Flash device, particularly for writes to avoid erasing the same set of blocks.

The throughput values are then used to fit the model described in the previous section. The measured throughput values, corresponding throughput parameters and details of the linear fit are all discussed in the evaluation section (Section 6).

5. FLASH DISK SIMULATOR DESIGN AND IMPLEMENTATION

This section introduces the Flash simulator, describes its overall design and details important features of it.

5.1 Overall Design

The Flash simulator is a dynamically loadable kernel extension (kernel module) on the AIX operating system that 'pins' a chunk of memory and simulates it as Flash memory. Once the extension is loaded, a Flash device appears on the system as a disk device and one can configure logical volumes and create paging devices on top of such a device. The OS simply knows the kernel extension as a driver for a disk device and passes I/O requests to the kernel extension strategy routine to handle read/write operations.

Internal to the simulator (kernel extension), every read-/write operation arrives as an I/O request (called *buf-struct*). Every buf-struct is examined to see if it is a read or a write operation and a delay for that operation is simulated (as explained below in Section 5.2). A dedicated kernel-process which is bound to a processor handles all these requests (resembling the notion of a Flash controller). In addition, the data transfer part is simulated using a high-performance *memcpy* operation implemented using firmware calls (*hypervisor* calls). This allows for minimal overhead in copy operations. It is also to be noted that the actual timing delay is simulated for a read/write operation after the copy operation is done, so that only the remaining delay is simulated. Delays themselves are simulated using high-resolution nanosecond granularity timer services that are provided in the kernel. These services are also used to maintain a plethora of statistics within the simulator which can then be reported using utilities that are exported as commands to the user. Installing and configuring the kernel extension, creating logical volumes and configuring paging devices is extremely simple, and can be done in a matter of a couple of minutes using just a few commands. Applications can then be run to access the Flash device as a Solid-State Disk.

The core of the Flash simulator is the delay simulation for I/O requests. The simulator has been implemented in a modular fashion to allow using various models for delay simulations. The throughput model is one example.

5.2 Simulating Read/Write Timing

Delay that is to be simulated for an operation is calculated by implementing the throughput model in the simulator. Minimal state information {*block size of the previous request, block number of the previous request, previous opreation (read/write)*} is maintained for this purpose. The simulator uses the throughput parameters that are fed to it using system calls. These values are extracted using micro-benchmarks that are designed for this purpose (as explained in Section 4.3).

Pseudo-code 1 shows a high-level view of implementing delay simulation. If the current operation is same as the previous one ($curOp == prvOp$) and if the current request is immediately adjacent to the previous request on the disk ($(prvBlk+prvSize) == curBlk$), sequential parameters are used for delay calculation (delay is represented by *waitTime* in the pseudo-code shown). Any other request is treated as a random request. The throughput parameters used in the pseudo-code are further discussed in detail in the perfor-

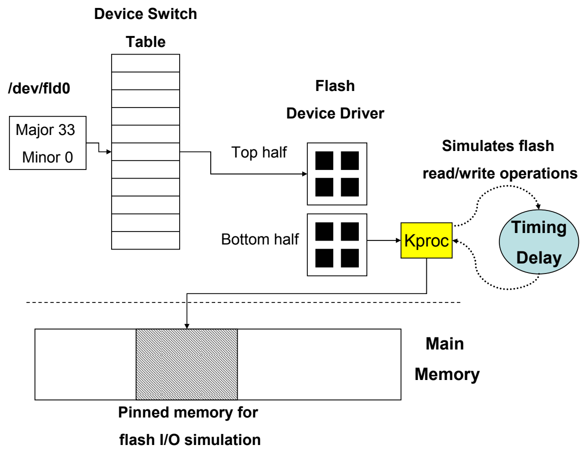

Figure 5: Overall Architecture of the Flash Simulator.

Pseudocode 1 : Pseudo-code implemented to simulate delays for Read/Write operations

```
Model Parameters:
{ $(A_{sr},B_{sr}),(A_{rr},B_{rr}),(A_{sw},B_{sw}),(A_{rw},B_{rw})$ }
State:
{ $prvBlk, prvSize, prvOp$ }

/* Procedure to calculate the delay to be simulated for an I/O op. */
$time\_t$ Delay($bufstruct\ b$)
{
  $\{curOp, curSize, curBlk\} \Leftarrow \{b \rightarrow op, b \rightarrow size, b \rightarrow blkNo\}$
  $access = RAND$
  if ($curOp == prvOp$) then
    if (($prvBlk + prvSize$) == $curBlk$) then
      $access = SEQ$
    end if
  end if
  if $curOp == READ$ then
    if $access == SEQ$ then
      $(A, B) \Leftarrow (A_{sr}, B_{sr})$
    else
      $(A, B) \Leftarrow (A_{rr}, B_{rr})$
    end if
  else
    if $access == SEQ$ then
      $(A, B) \Leftarrow (A_{sw}, B_{sw})$
    else
      $(A, B) \Leftarrow (A_{rw}, B_{rw})$
    end if
  end if
  $waitTime = A + B \times curSize$
  $\{prvOp, prvSize, prvBlk\} \Leftarrow \{curOp, curSize, curBlk\}$
  return $waitTime$
}
```

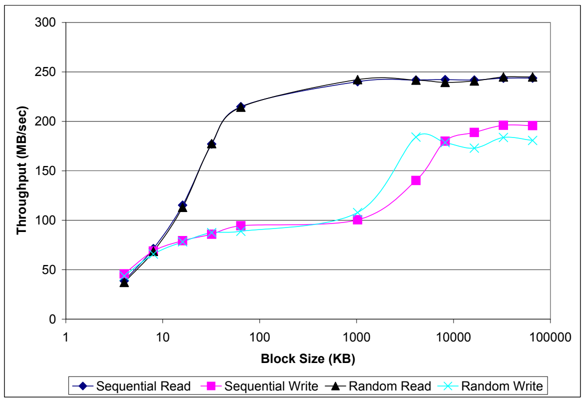

Figure 6: Throughput of the Zeus Flash Disk for Various Block sizes.

mance evaluation section (Section 6). Note that the pseudo-code shown here is just a high-level overview and omits many implementation details for simplicity and clarity.

6. PERFORMANCE EVALUATION

This section describes experimental setup and a series of experiments conducted to (i) validate the model presented in Section 4 (ii) extract model parameters and (iii) validate the simulator using raw-I/O and commercial workloads.

All the experiments were conducted on the IBM AIX version 6 Operating System running on the POWER6 [21] processor with 4.7 GHz clock speed, 2 physical CPUs, and 2 hardware threads (SMTs) per CPU. The Flash disks used were the STEC Zeus 70GB Solid-State drives.

6.1 Validating the Model

To validate the throughput model, we access the SSD in raw mode (i.e., without any caching in the OS or in the adapter), plot the observed throughput and fit to the proposed model using linear regression. The workloads were run according to the parameter extraction process explained in Section 4.3. Block size (size of the I/O requests) was varied from 4KB to 64MB, random and sequential requests were generated and the observed throughput was plotted.

	Seq Rd	Rand Rd	Seq Wr	Rand Wr
$A(\mu s)$	$127.5(A_{sr})$	$230.6(A_{rr})$	$2167(A_{sw})$	$770(A_{rw})$
$B(\mu s/KB)$	$4.005(B_{sr})$	$3.987(B_{rr})$	$4.96(B_{sw})$	$5.382(B_{rw})$
r^2	0.999997	0.999981	0.99932	0.999722
P_{value}	5.21×10^{-26}	1.39×10^{-22}	1.45×10^{-15}	2.54×10^{-17}

Table 1: Throughput Parameters Extracted for the STEC Zeus Flash Disk.

6.1.1 *Linear Regression Fit*

Figure 6 shows the throughput plot for sequential and random patterns. The curves from Figure 6 were used to fit the model described in Section 4. Table 1 shows the regression output obtained for each curve. All the A_x values (first row of Table 1) refer to the intercepts and B_x (second row) refer to the slope coefficients. Also shown are the P_{value} and r^2 (coefficient of determination) for each fit. From the table, firstly, $P_{value}s$ for all the curves are extremely small (last row of the table). The largest P_{value} that we have is of the order 10^{-15} which means that *regression is very significant*. Secondly, values of r^2 are greater than 0.999 for all the curves which means that the *correlation is very strong*. This shows that the linear model fits very accurately with experimental data and validates the throughput model.

The first two rows of the Table 1 (intercepts and the slope coefficients) are essentially the parameters of the model that are fed to the simulator.

6.1.2 *Further interpretation and analysis of the curves and the linear fit*

We explain the behavior of the Flash device here in detail for the sequential and random experiments conducted and also correlate this with the linear fit:

- Firstly (from Figure 6), throughput for reads and writes increases as the block-size increases and saturates at block-size of 8MB. This is true for both random and sequential patterns. One main reason for this behavior is that unlike hard disks, there are no seek and rotational delays in a Flash device. As a result, the time to access a page/block is fairly independent of the 'distance' between successive I/O requests (there is no head that needs to move from the current block to the next block) and the actual transfer size of the request gains much more prominence. Thus, throughput increases with increasing block size. This is also an implication of the fact that less commands need to be submitted to internal decoders if the block sizes are large. Also, software overhead involved in device driver queuing and I/O completion handling is minimized with larger block sizes. From the model itself, this is evident from the throughput formula shown in Equation 5. As the value of n increases, throughput saturates to $1/B_x$.

- Secondly, reads are faster than writes. This is clear from the throughput curves in Figure 6 and from Table 1. As explained in previous sections, writes usually consume more time because (i) per-page program time for the NAND array (from the data register) is of the order of couple of hundreds of μs whereas read time from the array is only tens of μs (ii) many writes need additional block erases which is $1-2ms$. This also explains the values for the intercepts(A_x) in the Table 1. Values for sequential and random write curves ($A_{sw} = 2167\mu s$ and $A_{rw} = 770\mu s$) are much higher compared to those of reads ($A_{sr} = 127.5\mu s$ and $A_{rr} = 230\mu s$). Note that although intercept for random write (A_{rw}) is lower than that of sequential write (A_{sw}), its x coefficient is greater (it can be seen that random write time will be more than sequential write time for requests $> 3.3K$).

- Thirdly, sequential operations are faster than random operations. This is particularly true for writes. As explained in Section 4, Flash devices internally allow

I/O Operation	Average Percentage Error (%)
Random Read	4.6
Sequential Read	5.31
Random Write	4.10
Sequential Write	6.57

Table 2: Average Percentage Error of the Raw I/O Throughput of Read and Write operations between the Flash Simulator and the Zeus Flash Disk.

data retrieval of next page from memory cells to cache buffer as the previous page is being sent out to the controller (similar method for writes) [26]. Due to this overlap in data transfer for sequential operations, they are faster. Table 1 shows this in terms of x coefficients (B_x). B_{rw} has the highest value($5.382\mu s$) which means that per-page cost of doing a random write is the maximum of all per-page costs in a Flash device.

In summary, sequential and random experiments conducted on the Zeus Flash SSD using micro-benchmarks validate our throughput model. This also gives us a set of model parameters that have been extracted on this Flash SSD that can be fed to the simulator.

6.2 Validating the Simulator

This section presents the validation results for the simulator. We first validate the simulator using benchmarks that generate sequential and random I/O requests on disks. We then configure a paging device on top of the simulator and use two representative commercial applications: SPECJbb 2005 [2] and DayTrader [1] to validate the simulator.

6.2.1 Validation using Raw I/O Benchmarks

For the raw I/O benchmarks, we generate sequential/random request patterns over the entire device for varying block sizes (4KB to 8MB). We first do this on the real Flash disk and then on the Flash simulator (running with the throughput model and extracted parameters).

Figure 7 compares the throughput observed on the Flash simulator and the real Flash disk for various block sizes. Throughput behavior exhibited by the simulator closely matches that of real SSD. Figure 7 also shows bars of the percentage error values of the Flash simulator based on three simulator runs. Table 2 summarizes the average percentage error observed during the various raw I/O operations on the Flash simulator. The percent errors are on average less than 7%.

6.2.2 Validation using Paging

After validating the Flash simulator using raw I/O workloads, we configured the Flash simulator and the real Flash disks as paging devices. Only one type of paging device was used in each experiment. All other paging devices were deactivated to allow paging to either the Flash simulator or the Flash disk. We used a paging space configuration that consists of one AIX logical volume with a size of $2GB$. Two representative commercial benchmarks were then executed on this infrastructure. To trigger paging, the real memory size available to these applications was decreased using AIX's dynamic reconfiguration (DR) [17] operation for memory remove. The DR memory operations in AIX allow a given amount of memory to be added or removed dynamically from the underlying AIX OS instance.

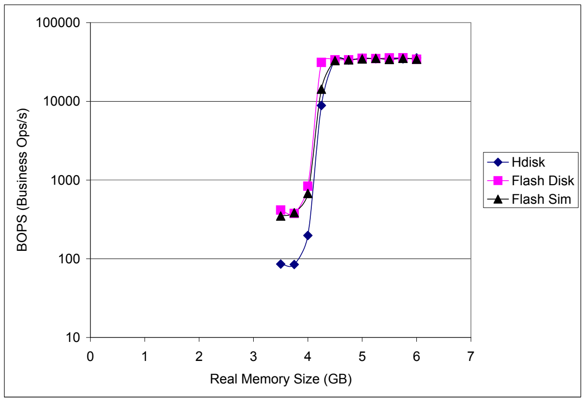

Figure 8: Performance of the SPECJbb 2005 Benchmark Using the Flash disk, the Flash simulator and the Hard disk as Paging Devices (Paging space size = 2GB, Y-axis in log scale).

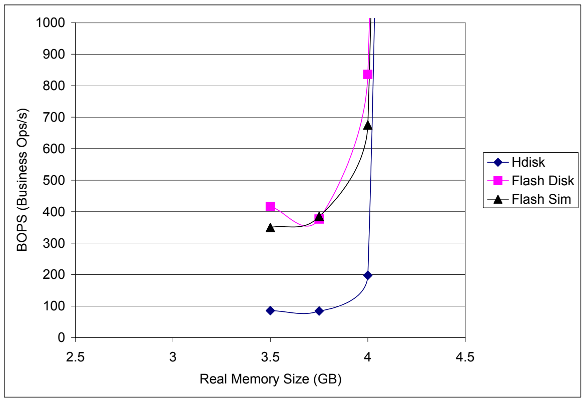

Figure 9: Performance of the SPECJbb 2005 Benchmark While Paging to the Flash Disk, the Flash Simulator, and the Hard Disk.

SPECJbb Paging Experiments

SPECJbb 2005 [2] is a benchmark from the Standard Performance Evaluation Corporation (SPEC) [3] based on the TPC-C benchmark specifications. It emulates a 3-tier system in a JVM with emphasis on the middle tier. SPECJbb was executed with 10 warehouses. Each experiment was run for 240 seconds and the main metric of measurement was BOPS (Business Operations Per Second).

Figure 8 shows the effect of memory size on BOPS. Memory size was varied from 3.5GB to 6GB, in the increments of 0.25GB. For a memory of 6GB, SPECJbb BOPS is around 35K and does not depend on the paging device (whether it is hard disk, SSD or the flash simulator) as there is no paging.

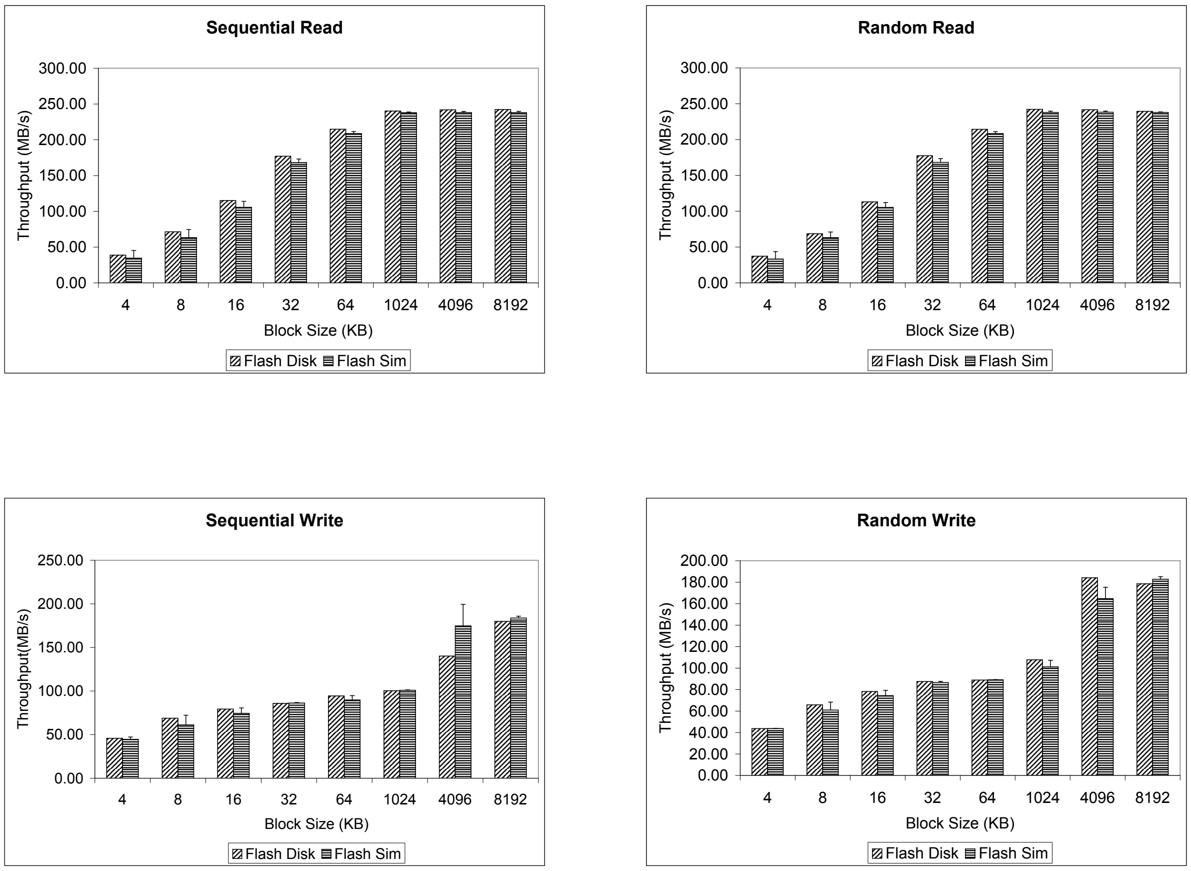

Figure 7: Read and Write Behavior of the Zeus Flash Disk and the Flash Simulator for Various Block Sizes.

As the memory size is decreased, paging starts happening around 5GB and BOPS starts dropping. It can be seen from the figure that as memory decreases from 4.5GB to 3.5GB, BOPS for the hard disk drops very rapidly. Flash disk and Flash simulator do sustain BOPS for another 0.25GB but then give up to the thrashing workload. However, it is clear from the graph that the BOPS achieved using a solid state disk is significantly higher than the BOPS observed using a hard disk. In fact, under memory pressure, BOPS achieved using Flash disk is roughly 10 times higher, corroborating the fact that SSD performs better than a hard disk. Also seen is the curve for the Flash simulator which almost completely overlaps with that of the real Flash disk. Flash simulator performs very close to the Flash disk for all the data points. Note that at each data point, the read and write operations are dependent on SPECJbb behavior under that memory pressure. Thus, this benchmark is a representative of varying read/write percentages and parallelism. Flash simulator, using its throughput model is able to accurately capture the behavior of Flash disks for SPECJbb. Figure 9 further zooms into the area between Memory (3.5-4GB) and BOPS (0-1000) from Figure 8. It can be seen that Flash simulator performs a little worse than the real Flash disk. This is attributed to overhead involved in simulation, particularly overhead involved in timing measurements.

DayTrader Paging Experiments

IBM Websphere application server (WAS) [4] is a Java-based web application server framework that is designed to deploy electronic business applications across multiple computing platforms. DayTrader [1] is a Websphere benchmark application that emulates an online stock trading system. It was originally developed at IBM and donated to the Apache Geronimo community in 2005. The application allows users to perform typical trading operations such as login, viewing portfolios, looking up stock quotes, and buying or selling stock shares. Several Web-based load drivers such as Mercury LoadRunner, Rational Performance Tester, or Apache JMeter, provide realistic workload scenarios that drive the application. DayTrader is traditionally used to measure and compare the performance of Java Platform and Enterprise Edition (Java EE) application servers. DayTrader is composed of a set of Java EE technologies that includes Java Servlets and JavaServer Pages (JSPs) for the presentation layer and Java database connectivity (JDBC), Java Message Service (JMS), Enterprise JavaBeans (EJBs) and Message-Driven Beans (MDBs) for the back-end business logic and persistence layer.

Similar to the SPECJbb benchmark, we run DayTrader under a stressed memory environment to trigger paging to the Flash disk or the Flash simulator. We then measure the performance of the application as seen by the simulated users as the (i) number of web pages serviced per second (throughput) and (ii) response time seen by the end user for each request. Figure 10 shows the experimental setup used to run paging experiments with DayTrader. Two machines were used in this setup. The DayTrader server is a dedicated-processor logical partition (LPAR) running the AIX version 6 OS on an IBM POWER6 processor with 2 CPUS and 2 hardware threads per CPU and a maximum memory size of 10GB. The DayTrader server was hosting an instance of the DB2 database and an instance of the WebSphere Application Server (WAS). The DayTrader client is

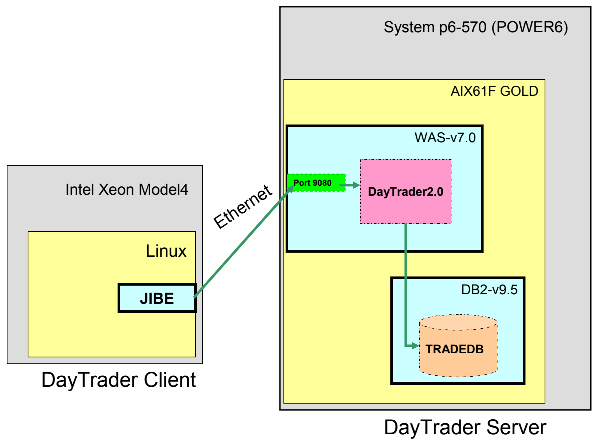

Figure 10: Experimental Setup for the DayTrader Benchmark.

a Linux Intel Xeon machine running the JIBE (WebSphere Studio Workload Simulator), which simulates a specifiable number of concurrent browser clients. We simulated 500 clients for stressing the DayTrader application running on the DayTrader server. This number of clients was sufficient to keep the server continuously busy with waiting requests to be processed. Each client sent various types of requests (login, query account, update account; get portfolio, quote; buy, sell, etc.) to the server.

Figure 11 shows the throughput of DayTrader while the application was paging using the two paging space configurations (one on real Flash disk and the other on the Flash simulator) under stressful memory conditions. Due to paging, the throughput of DayTrader was very low (an average of 14 pages/s) during both paging configurations. It can be seen from the figure that the throughput values while paging to the Flash simulator and the Flash disks are very close (vary between the values of 13 pages/s and 15 pages/s). The variance in the throughput for either of the curves is attributed to other OS component disturbances and noise in network path and scheduling. Figure 12 shows the performance of DayTrader in terms of response time observed by the user. Again, the results show comparable behavior of the Flash disk and the Flash simulator in most cases with a response time that varies between 31 and 32 seconds.

7. CONTRIBUTIONS

A fundamental way in which a Solid-State Disk made out of Flash memory differs from a hard disk is that there are no heads and there is no spin. Therefore, unlike hard-disks where inter-request track/sector distance contributed to significant portion of the I/O time (due to seek and rotational delays), transfer size of a request gains much more prominence in Flash devices.

Research presented in this paper has exploited this observation and made the following contributions: (i) A model for read/write transfer time (and throughput) of a Flash device for sequential and random access patterns is presented. To our knowledge, this is the first model being proposed for internal workings of a Flash device. (ii) The model is

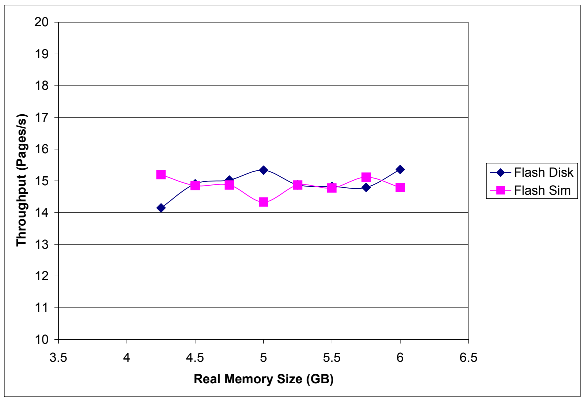

Figure 11: Throughput of the DayTrader Application While Paging to Flash Disk and Flash Simulator (Paging Space Size = 2GB).

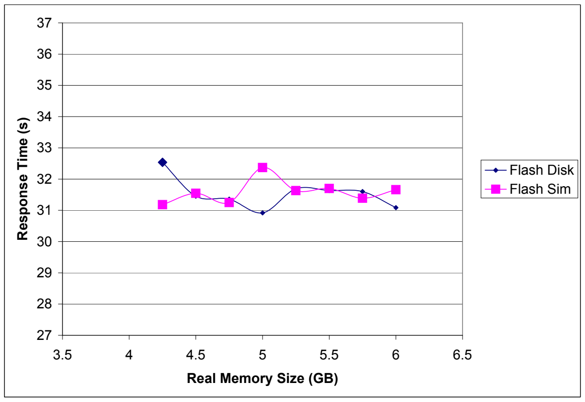

Figure 12: Response Time of the DayTrader Application While Paging to Flash disk and the Flash Simulator (Paging Space Size = 2GB).

validated on STEC Zeus Solid-state Disk and model parameters for the Zeus SSD are extracted (iii) A method to build micro-benchmarks for parameter extraction is presented (iv) A technique to build an efficient simulator using the model and extracted parameters is proposed. An important feature of this technique is that it maintains minimal state information and simulates delays using the {*size*, *sequentiality*, *I/O direction*} properties of an I/O request (v) A simulator is built as a kernel extension in the AIX operating system. This simulator can be configured in a matter of minutes and commercial/scientific applications could be run without any change or trace-collection (vi) Finally, the simulator is validated using raw-I/O and commercial workloads.

8. CONCLUSIONS AND FUTURE WORK

This paper presents a method of modeling a Flash device and building a Flash simulator. This method exploits the nature of Flash devices that they do not have rotational delays and capitalizes on the throughput behavior of the Flash disk. A linear model was constructed to model a Flash device. This paper also shows that one can simulate Flash based SSDs without simulating every minor detail and internal organization of a Flash device. A throughput-based simulation of benchmarks is within 7% error range compared to a real Flash disk. However, it is to be noted that research to come up with precise ways of simulating Flash devices has just started. One could think of better ways to combine a more detailed Flash device internal organization and the throughput model. While we are excited about this simple way of simulating Flash devices, work is underway to improve this model, particularly to incorporate layout and endurance related statistics and metrics, to derive them from a real Flash disk, and further validate this using a much wider set of benchmarks.

9. ACKNOWLEDGMENTS

The authors would like to thank their colleagues Sarma Burugula and Niteesh Dubey for their feedback and help in simulator design, and sharing their invaluable knowledge and insights into kernel extensions' behavior, kernel level debugging and paging device design. The authors would also like to thank Priya Nagpurkar for her help in experimental setup through her experience which indeed sped up our experimentation process and Joe Edwards from IBM in Rochester for his great help in configuring the STEC Zeus SSD adapters.

10. REFERENCES

[1] Apache DayTrader Benchmark Sample: `http://cwiki.apache.org/GMOxDOC20/daytrader.html`.

[2] SPECJbb2005(Java Performance Benchmark) , `http://www.spec.org/jbb2005/`.

[3] Standard Performance Evaluation Corporation : `http://www.spec.org/`.

[4] WebSphere software , `http://www-01.ibm.com/software/websphere/`.

[5] N. Agrawal, V. Prabhakaran, T. Wobber, J. D. Davis, M. Manasse, and R. Panigrahy. Design tradeoffs for SSD Performance. In *ATC'08: USENIX 2008 Annual Technical Conference on Annual Technical Conference*, pages 57–70, Berkeley, CA, USA, 2008. USENIX Association.

[6] L. Bouganim, B. T. Jónsson, and P. Bonnet. uFLIP: Understanding Flash IO Patterns. In *CIDR*. www.crdrdb.org, 2009.

[7] Y.-H. Chang, J.-W. Hsieh, and T.-W. Kuo. Endurance enhancement of Flash-memory Storage Systems: An efficient static wear leveling design. In *DAC '07: Proceedings of the 44th annual Design Automation Conference*, pages 212–217, New York, NY, USA, 2007. ACM.

[8] M. Charles. YAFFS: The NAND-specific Flash File System - Introductory Article (`http://www.yaffs.net/`). 2002.

[9] H. Dai, M. Neufeld, and R. Han. ELF: An Efficient Log-structured Flash File System for Micro Sensor

Nodes. In *SenSys '04: Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems*, pages 176–187, New York, NY, USA, 2004. ACM.

[10] C. Dirik and B. Jacob. The performance of PC Solid-state disks (SSDs) as a function of Bandwidth, Concurrency, Device architecture, and System organization. In *ISCA '09: Proceedings of the 36th Annual International Symposium on Computer architecture*, pages 279–289, New York, NY, USA, 2009. ACM.

[11] F. Douglis, R. Cáceres, F. Kaashoek, K. Li, B. Marsh, and J. A. Tauber. Storage Alternatives for Mobile Computers. In *Proceedings of the 1st Symposium on Operating Systems Design and Implementation (OSDI)*, pages 25–37, 1994.

[12] E. Gal and S. Toledo. A Transactional Flash File System for Microcontrollers. In *ATEC '05: Proceedings of the annual conference on USENIX Annual Technical Conference*, Berkeley, CA, USA, 2005. USENIX Association.

[13] E. Gal and S. Toledo. Algorithms and Data Structures for Flash Memories. *ACM Comput. Surv.*, 37(2):138–163, 2005.

[14] G. R. Ganger, B. L. Worthington, and Y. N. Patt. The DiskSim Simulation Environment - Version 2.0 Reference Manual. Technical report, 1999.

[15] A. Gupta, Y. Kim, and B. Urgaonkar. DFTL: A Flash Translation Layer employing demand-based selective caching of page-level address mappings. In *ASPLOS '09: Proceeding of the 14th international conference on Architectural support for programming languages and operating systems*, pages 229–240, New York, NY, USA, 2009. ACM.

[16] W. Hutsell, J. Bowen, and N. Ekker. Flash Solid-State Disk Reliability. Technical report, Nov 2008. Texas Memory Systems White Paper.

[17] J. Jann, N. Dubey, R. S. Burugula, and P. Pattnaik. Dynamic reconfiguration of CPU and WebSphere on IBM pSeries Servers. *Softw. Pract. Exper.*, 34(13):1257–1272, 2004.

[18] T. Kgil, D. Roberts, and T. Mudge. Improving NAND Flash Based Disk Caches. In *ISCA '08: Proceedings of the 35th International Symposium on Computer Architecture*, pages 327–338, Washington, DC, USA, 2008. IEEE Computer Society.

[19] H. Kim and S. Ahn. BPLRU: A Buffer Management Scheme for Improving Random Writes in Flash Storage. In M. Baker and E. Riedel, editors, *Conference on File and Storage Technologies: FAST*, pages 239–252. USENIX, 2008.

[20] L. Larcher, I. R. A. Padovani, P. Pavan, G. M. A. Calderoni, F. Gattel, and P. Fantini. Modeling NAND Flash Memories for Circuit Simulations. In *Simulation of Semiconductor Processes and Devices (SISPAD)*, pages 293–296, Italy, 2007. Springer Vienna.

[21] H. Q. Le, W. J. Starke, J. S. Fields, F. P. O'Connell, D. Q. Nguyen, B. J. Ronchetti, W. M. Sauer, E. M. Schwarz, and M. T. Vaden. IBM POWER6 Microarchitecture. *IBM J. Res. Dev.*, 51(6):639–662, 2007.

[22] J. Lee, E. Byun, H. Park, J. Choi, D. Lee, and S. H. Noh. CPS-SIM: Configurable and accurate clock Precision Solid State drive Simulator. In *SAC '09: Proceedings of the 2009 ACM symposium on Applied Computing*, pages 318–325, New York, NY, USA, 2009. ACM.

[23] S.-W. Lee, B. Moon, C. Park, J.-M. Kim, and S.-W. Kim. A case for Flash memory SSD in Enterprise Database Applications. In *SIGMOD: Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, pages 1075–1086, New York, NY, USA, 2008. ACM.

[24] S.-W. Lee, D.-J. Park, T.-S. Chung, D.-H. Lee, S. Park, and H.-J. Song. A Log buffer-based Flash Translation Layer using fully-associative sector translation. *ACM Trans. Embed. Comput. Syst.*, 6(3):18, 2007.

[25] Lenovo. Lenovo Products, 2009. `http://lenovo.com/us/en/index.html`.

[26] Micron. NAND FLASH Memory: 4Gb, 8Gb, and 16Gbx8 NAND Flash Memory. MT29F4G08AAA, MT29F8G08BAA, MT29F8G08DAA, MT29F16G08FAA, 2006.

[27] Micron. Industry leading 43nm NAND, 2009. `http://www.micron.com/innovations/process_tech/`.

[28] D. Narayanan, E. Thereska, A. Donnelly, S. Elnikety, and A. Rowstron. Migrating server storage to SSDs: Analysis of Tradeoffs. In *EuroSys '09: Proceedings of the Fourth ACM European Conference on Computer Systems*, pages 145–158, New York, NY, USA, 2009. ACM.

[29] Samsung. NAND Flash: living in NAND Flash world, 2009. `http://www.samsung.com/global/business/semiconductor/products/flash/Products_NANDFlash.html`.

[30] Texas Memory Systems. Ramsan from Texas Memory Systems, 2009. `http://www.ramsan.com`.

[31] K. Youngjae, T. Brendan, G. Aayush, and U. Bhuvan. FlashSim: A Simulator for NAND Flash-based Solid-State Drives. In *The First International Conference on Advances in System Simulation*, 2009.