Performing Sound Flash Device Measurements: Some Lessons from uFLIP

Matias Bjørling
University of Copenhagen
Copenhagen, Denmark
silverwolf@diku.dk

Lionel Le Folgoc
INRIA Rocquencourt
Le Chesnay, France
lionel.le_folgoc@inria.fr

Ahmed Mseddi
INRIA Rocquencourt
Le Chesnay, France
ahmed.mseddi@inria.fr

Philippe Bonnet
IT University of Copenhagen
Copenhagen, Denmark
phbo@itu.dk

Luc Bouganim
INRIA Rocquencourt
Le Chesnay, France
luc.bouganim@inria.fr

Björn Þór Jónsson
School of Computer Science
Reykjavík University
Reykjavík, Iceland
bjorn@ru.is

ABSTRACT

It is amazingly easy to get meaningless results when measuring flash devices, partly because of the peculiarity of flash memory, but primarily because their behavior is determined by layers of complex, proprietary, and undocumented software and hardware. In this demonstration, we share the lessons we learnt developing the uFLIP benchmark and conducting experiments with a wide range of flash devices. We illustrate the problems that are actual obstacles to sound performance and energy measurements, and we show how to mitigate the effects of these problems. We also present the uFLIP web site and its on-line visualization tool that should help the research community investigate flash device behavior.

Categories and Subject Descriptors

B.3.2 [**Memory Structures**]: Design Styles—*mass storage (flash devices)*; B.8.2 [**Performance and Reliability**]: Performance Analysis and Design Aids

General Terms

Measurement, Performance, Experimentation

Keywords

Flash devices, Benchmarking, Methodology, uFLIP, SSD, Energy Measurement

1. INTRODUCTION

Many different types of flash devices are finding their way into the memory hierarchy of data management infrastructures, from SSD to PCI-based racks (e.g., fusionIO and RamSan) and energy efficient FAWNs [1]. Indeed, flash technology has great potential, promising increased throughput with reduced energy consumption. While flash chip behavior is very precisely specified, commercially available flash devices do not behave as flash chips and are both complex and undocumented. It is important for practitioners and researchers alike to understand the performance characteristics of these devices in order to a) compare the performance of competing devices, b) understand which type of flash devices are best fitted to a given usage pattern, c) adapt usage pattern to a given type of flash device, and d) influence the design of future flash devices by exposing their shortcomings for handling specific usage patterns.

We have designed a benchmark, called uFLIP, to cast light on all relevant usage patterns of current, as well as future, flash devices [2]. uFLIP is a set of nine micro-benchmarks based on IO patterns, or sequences of IOs, which are defined by 1) the time at which the IO is submitted, 2) the IO size, 3) the IO location (logical block address, or LBA), and 4) the IO mode, which is either read or write. Each micro-benchmark is a set of experiments designed around a single varying parameter, that affects either time, size, or location. In [2], we measured and summarized the response time for individual IOs. More recently, we have devised a mechanism for measuring the energy consumption of flash devices, shown in Figure 1. While energy consumption cannot be traced to individual IOs, we can associate energy consumption figures to IO patterns, which helps us understand further the behavior of the devices.

As it turns out, it is very easy to get meaningless results when running an experiment because a) the initial state is not well defined or stable, b) an experiment is not long enough to capture the performance variations of the device under study (as response time is not uniform), or c) consecutive runs interfere with each other.

Our goals with this demo are to point out the methodology challenges that exist when measuring the performance or energy consumption of a flash devices, and to illustrate the approaches developed in uFLIP to tackle these challenges. We also aim at broadening the uFLIP user base. It is in the interest of the community that many devices get benchmarked on different servers. We have thus made the benchmark software available for both Windows and Linux,

SIGMOD'10, June 6–11, 2010, Indianapolis, Indiana, USA.

Figure 1: Energy Measurement Setup.

and made it easy to share and analyze benchmark results by developing an on-line visualization tool available on the uFLIP web site (http://www.uflip.org/).

2. FLASH DEVICES

The flash devices that now emerge as a replacement for mechanical disks are complex devices composed of flash chip(s), controller hardware, and proprietary software that together provide a block device interface via a standard interconnect (e.g., USB, IDE, SATA, PCI). While flash chips are very precisely specified—they have interesting properties (e.g., read/program/erase operations, no updates in-place, random reads are equivalent to sequential reads)—commercially available flash devices are not. The only certainty is that they do not behave as flash chips. They provide a block interface, where data is read and written in fixed sized blocks, and they integrate layers of software that manage block mapping, wear-leveling and error correction.

In all flash devices, the core data structures of the block manager are two maps between blocks, represented by their logical block addresses (LBAs), and flash pages. A direct map from LBAs to flash pages is stored on flash and in RAM to speed up reads, and an inverse map is stored on flash to re-build the direct map during recovery. There is a trade-off between the improved read performance due to the direct map and degraded write performance due to the update of the inverse map (updates of bookkeeping information for a page may cause an erase of an entire block). The software layer responsible for managing these maps both in RAM (inside the micro-controller that runs the block manager) and on flash is called flash translation layer (FTL). Using the direct map, the FTL introduces a level of indirection that allows trading expensive writes-in-place (with the erase they incur) for cheaper writes onto free flash pages.

Each update on a free flash page, however, leaves an obsolete flash page (that contains the before image). Over time such obsolete flash pages accumulate, and must subsequently be reclaimed synchronously or asynchronously. As a result, we must assume that the cost of writes is not homogeneous in time (regardless of the actual reclamation policy). Some block writes will result in flash page writes with a minimum bookkeeping overhead, while other block writes will trigger some form of page reclamation and associated erase(s). Assuming a flash device contains enough RAM and autonomous power, the FTL may be able to cache and destage both data and bookkeeping information.

Since the physical layout of data on flash devices is stored and manipulated via the direct map data structure, which manages the devices at some fixed granularity, there is no direct correspondence between an arbitrary IO request to the flash device and its translation to a physical request to a flash chip. Instead, the physical request is based in a complex manner on the current state of the direct map, which in turn is based on the entire history of previous IO requests (see [3] for further illustration of this point).

While the principles of the flash translation layer described above are well known, the design decisions and the associated performance trade-offs are typically not documented: Flash devices are black-boxes.

3. THE uFLIP METHODOLOGY

There are three characteristics of flash devices that make benchmarking particularly hard. In the following, we discuss each characteristic and our methodology to mitigate its effect.

A. The State of the Device Impacts Performance. In order to obtain repeatable results, we should run the micro-benchmarks from a well-defined initial state, which is independent of the complete IO history. Forcing a flash device into a well-defined and stable state, however, is not a trivial problem. Explicitly resetting a device to its factory settings is only possible for high-end PCI-based products (and it destroys all data on the device). For all other devices, the device state is only controlled by the history of submitted IOs. The problem is thus twofold: a) what sequence of IOs should be submitted before a benchmark to enforce a well-defined state, and b) how to submit IOs during the benchmark to retain a stable state.[1]

In uFLIP, we make the following assumption: Writing the whole flash device completely yields a well-defined state. The rationale is that following a complete write of the whole flash device, both the direct and indirect maps managed by the FTL are filled and well-defined. In [2], we prepared each device by performing random IOs of random size (ranging from 0.5KB to the flash block size, 128KB) on the whole device. The advantage of this method is that it is quite stable, as only sequential writes disturb the state significantly. In order to limit the impact of sequential writes during the benchmark, we directed them to distinct target spaces when running the micro-benchmarks. The downside is that this method is slow. In [3], the authors relied on sequential writes to enforce a well-known state, which is much faster than random writes. However, stability is lower because random writes, badly aligned IOs, or IOs of different sizes impact a sequential state much more significantly than a random state. Studying in detail the impact of the initial state on performance is still a topic for future work.

B. Response Time is Not Uniform. We propose a two-phase model to capture response time variations within a micro-benchmark run. In the first phase, which we call start-up phase, response time is cheap. Such a

[1]Note that tractability becomes a key issue as it may take hours (or days) to enforce a well-defined state (depending on the capacity and performance of the device). For that reason, enforcing a well-defined state between micro-benchmark runs is not an option.

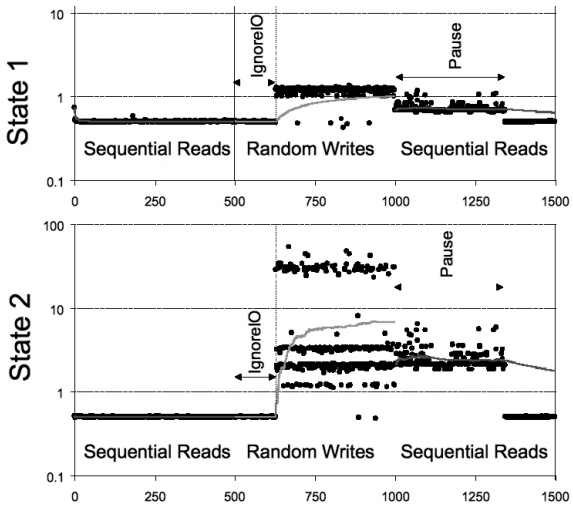

Figure 2: Performance of the Same Sequence of IOs on the Same Flash Device in two Different States.

start-up phase can occur when expensive operations are delayed, e.g., due to buffering or lazy garbage collection. In the second phase, which we call running phase, response time is typically oscillating between two or more values (e.g., because pages are reclaimed or garbage collection is activated).

For each experiment, we define two parameters to account for the size of the start-up phase (*IOIgnore*) and the size of the running phase (*IOCount*, which includes the start-up phase). First, we run four baseline patterns (SR, RR, SW and RW) with a very large number of submitted IOs. By plotting the IO costs, we can then identify the two phases for each pattern and derive upper bounds across the patterns for *IOIgnore* and *IOCount*. Note that the value of *IOCount* has a direct impact on both the time it takes to run an experiment and on state stability.

C. Asynchronous Activity May Cause Interference. Consecutive benchmark runs should not interfere with each other. Consider a device that implements an asynchronous page reclamation policy. Its effects should be captured in the running phase defined above. We must make sure, however, that the effect of the page reclamation triggered by a given run has no impact on subsequent, unrelated runs. For each experiment, we therefore define a third parameter to account for the pause that should be introduced in between runs to avoid interferences (*Pause*).

To evaluate the *Pause* parameter, we rely on the following experiment. We submit sequential reads, followed by a batch of random writes, and sequential reads again. We count the number of sequential reads in the second batch which are affected by the random writes. We then use this value to compute a lower bound on the pause between consecutive runs. Note that, when benchmarking a device with unknown properties, this is only an educated guess and therefore we propose to significantly overestimate the length of the pause.

Figure 2 illustrates the three parameters for two different states of the same device.

4. DEMONSTRATION SCENARIO

We will bring the energy measurement setup shown in Figure 1, which consists of a measurement server, along with a number of flash devices, and a measurement client to visualize the energy readings. Additionally, we will use the uFLIP web-site (a local copy, to avoid network problems) to visualize performance results. We plan to organize our demonstration around the following topics:

Motivation: Using a poster, we will discuss the characteristics and potential of flash chips and flash devices.

Bad Measurements: We will show uFLIP results, where the size of the start-up and running phases or the interferences are not accounted for (one or more of the three parameters is too small). We will show that results are unstable and difficult to analyze/understand. Note that enforcing random state takes too long for a live demonstration.

Setting Parameters: We will run experiments to determine the value of the *IOIgnore*, *IOCount* and *Pause* parameters for a given device (possibly provided by an attendee).

Good Measurements: We will compare uFLIP results obtained with the appropriate parameter settings to the previous results obtained without them.

Energy Measurements: We will illustrate how we measure energy consumption and discuss the additional difficulties from a methodological point of view. More specifically, we will show why it is difficult to determine the boundaries of the running phase from an energy consumption viewpoint.

uFLIP Results: Using the web interface for navigating uFLIP results we will comment our main findings. While all flash devices used in [2] showed a consistent behavior (poor random writes, unless if they are focused on a ten megabytes target, good performance for sequential writes, even if they are done concurrently in up to ten partitions, no parallelism), newer devices such as Fusion IO, or Intel X25 show very different behaviors (see appendix B). Interestingly, for this new class of devices, random writes are sometimes faster than sequential writes and submitting IOs in parallel is beneficial.

5. CONCLUSION

This demonstration, with its strong focus on methodological issues, is based on the lessons learned benchmarking flash devices with uFLIP. uFLIP was designed without making any assumption on the device behavior and usage and this design allows to benchmark successfully highly different class of devices, from USB keys to PCI-based racks, following the same methodology. We believe that these lessons are relevant to any researcher or practitioner that intends to measure the performance of an algorithm or a system that relies on flash devices, or to integrate energy consumption as a metric in an experimental framework.

6. REFERENCES

[1] D. G. Andersen, J. Franklin, M. Kaminsky, A. Phanishayee, L. Tan, and V. Vasudevan. Fawn: a fast array of wimpy nodes. In *SOSP*, pages 1–14, 2009.

[2] L. Bouganim, B. T. Jónsson, and P. Bonnet. uflip: Understanding flash io patterns. In *CIDR*, 2009.

[3] R. Johnson, M. Athanassoulis, R. Stoica, and A. Ailamaki. A new look at the roles of spinning and blocking. In *DaMoN*, pages 21–26, 2009.

[4] D. Tsirogiannis, S. Harizopoulos, and M. Shah. Analyzing the energy efficiency of a database server. In *SIGMOD*, 2010.

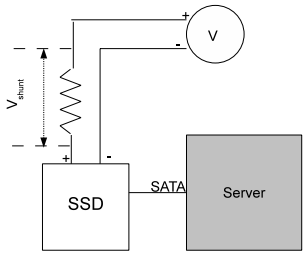

Figure 3: Diagram of the current shunt insertion used for high resolution measurement of the SSD energy consumption.

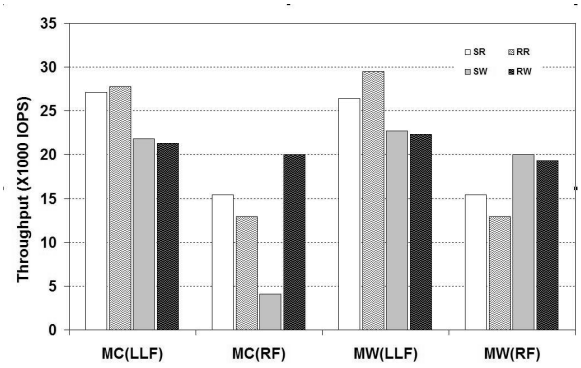

Figure 4: Baseline Patterns micro-benchmark on FusionIO's ioDrive

APPENDIX

A. ENERGY MEASUREMENT SETUP

Energy efficiency is emerging as a key metric for data management systems as power and cooling dominate the cost of ownership. Recent work has focused on server power breakdown resulting in key insights about the balance between idle and active modes, as well as CPU, hard disks and flash-based SSDs[4]. Those breakdown results were obtained with a clamp meter, i.e., a device that is used to directly measure the current flowing through a conductor (e.g., a wire). Such a method is appropriate for first-order reasoning about power (how many watts are consumed by a flash device or a hard disk in idle mode?). We are however interested in detailed time-based energy profiles for each run in the uFLIP benchmark. We need to achieve high-resolution measurements both in time and in current amplitude.

We rely on a current shunt insertion, where we measure the voltage drop across the shunt, which is proportional to the current flowing through the flash device. We use an oscilloscope equipped with a data logger to sample the voltage drop. The diagram in Figure 3 illustrates our set-up. The current shunt is installed on the high-side of an independent power source to guarantee stable voltage (the flash device is connected to the server via the data lines of the SATA connector).

We use a 1 Ω resistor ($\pm$5% error) as a shunt, and we sample voltage drop at 100 MHz, i.e., at the order of ten microsecond, which allows us to oversample the flash SSD IOs that are performed in tens of μs. This does not give us the energy consumption of each IO, but we are working at the appropriate resolution to reason about the impact of IO patterns on energy consumption.

B. uFLIP RESULTS

It is tempting to reduce flash devices to a single characteristic, e.g., random are slow while the other operations are fast. Our benchmark shows that such a simplification is just wrong. We need to be very careful when choosing the assumptions that underlie the design of systems relying on flash devices. There are significant differences in terms of performance characteristics across flash devices, and across states for a given device.

We illustrate this point with measurements that we conducted on FusionIO's ioDrive device[2]. Figure 4 shows throughput (IOs per second) for four basic patterns (RR: random read, SR: sequential read, RW: random writes, SW: sequential writes) at a granularity of 4K. We configure the flash device with either Maximum Capacity (MC) Max Write Performance (MW) configuration, and we format it either with low-level format (LLF: an erase command is sent to each flash page to get back to factory settings, i.e., no dirty block/page) or random format (RF: the entire device is written over randomly with blocks of different sizes).

The graph shows that, on FusionIO's ioDrive, the assumption that random writes are slower than other IO operations does not hold. In fact, with Max Capacity/Random Format, random writes are the fastest operations. More generally, there is nothing homogeneous about the performance of this device. The performance profile with factory settings (LLF) is similar with Max Capacity and Max Write, while it differs significantly (specially sequential writes exhibit different behaviour) in the worst case scenario of the random format (RF). This means that the performance profile of this device changes in time as its state evolves. Further work is needed to capture the evolution of flash device state and reason about their performance characteristics.

This result is obviously an idiosyncracy of the ioDrive implementation, but this is our point: We should be careful when designing algorithms and systems to define the class of device that we target. Not all flash devices will match a given set of assumptions. We refer interested readers to [2] for a description of the benchmark results, and to the uFLIP web site for the results obtained with more than twenty devices.

[2]http://www.fusionio.com/