RZ 3801 (# Z1103-001) 03/18/2011
Computer Science 20 pages

Research Report

System Performance Issues in Flash-Based Solid-State Drives

Werner Bux

IBM Research – Zurich
8803 Rüschlikon
Switzerland

E-mail: wbu@zurich.ibm.com

System Performance Issues in Flash-Based Solid-State Drives

Werner Bux
IBM Research – Zurich
8803 Rüschlikon, Switzerland

Abstract

In this report, we study the performance of solid-state drives that employ flash technology as storage medium. Our aim is to understand how the main performance measures, throughput and latency, are affected by the system parameters and design choices, in particular the garbage-collection mechanism and the priority scheme used for scheduling the various functions performed by a flash die. We observe and explain the occurrence of SSD-specific phenomena, such as latency spikes, and suggest techniques to avoid them.

1. Introduction

Flash memory is a non-volatile solid-state memory technology that can be electrically programmed, erased, and reprogrammed [1]. Its low power consumption and good shock resistance have made it very popular for portable devices, such as digital cameras, portable music players, mobile phones, and handheld computers. Moreover, the computer industry has started to use flash technology in so-called solid-state drives (SSDs) as hard-disk replacement in workstations and even enterprise systems.

Solid-state drives are built around multiple flash dies, which are used to store and retrieve user data. A certain number of dies, together with a channel controller and a bus interconnecting the flash dies and the controller, form a subsystem called flash channel. An SSD may contain several flash channels which function in parallel. Further building blocks are a central processor, random-access memory, host interface(s), DMA controllers, and a central bus or other interconnect mechanism to provide high-speed connectivity among the various subsystems. Fig. 1 shows an illustration of a generic SSD.

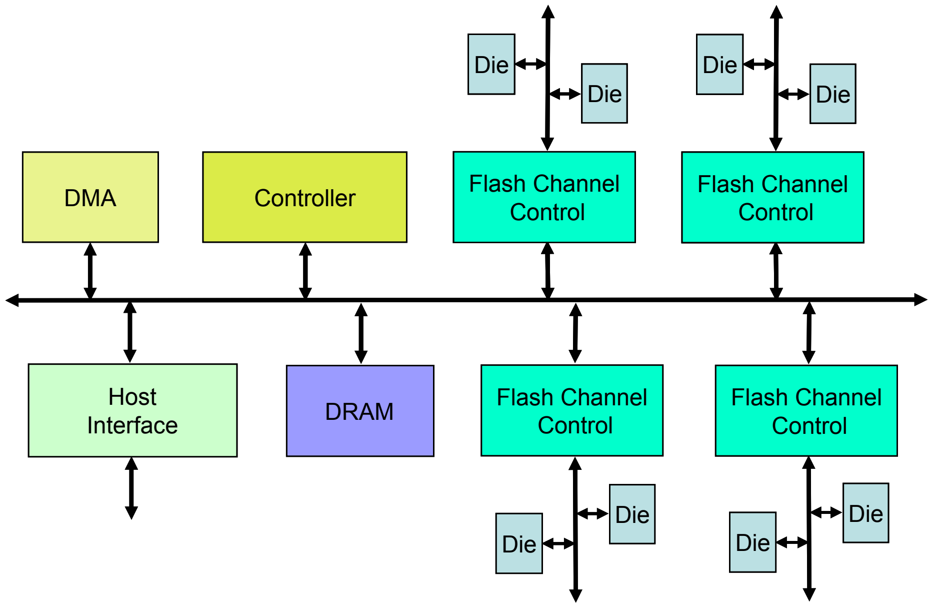

Fig. 1: Generic structure of a flash-based solid-state drive

In this report, we focus on the operation and performance of a single flash die within a solid-state drive. While the die is part of a complex system, it is, under certain conditions, feasible to cut out a single-die submodel and study its behavior in isolation. Our aim is to understand how key performance measures, especially throughput and latency, are affected by the system parameters and main design choices, such as properties of the garbage-collection method or the priority scheme used for scheduling the various functions performed by a flash die.

2. SSD Organization and Operation

A NAND flash memory is partitioned into blocks, where each block has a fixed number of pages (typically 64), and each page is of a fixed size (typically 4 KByte). Data are written in a unit of one page, and the erase is performed in a unit of one block. Reads are performed in units of pages. A page can be either programmable or unprogrammable. A programmable page, also called "free page", becomes unprogrammable once it has been written (programmed). A written page contains either valid or invalid data.

In flash-based systems, out-of-place write is used: When a page or a fraction thereof needs to be rewritten, the new data does not overwrite the memory location where the data is currently stored. Instead, the new data is written to another page and the old page is marked invalid. Over time, the SSD accumulates invalid pages, and the number of free pages decreases. To make space for new or updated data, invalid pages must be reclaimed. The only way to reclaim a page is to erase the entire block the page pertains to. Reclamation or garbage collection happens in multiple steps: First, one of the blocks is selected for reclamation. Second, the valid pages of this block are copied to a free block. which thereby becomes part of the device's actively used storage space. Third, the reclaimed block is erased and joins the free-block pool.

3. Performance Model of a Flash Device

3.1 Scope of the Model

Although an SSD system contains a multiplicity of flash dies, the focus of this study is on modeling the operation of a single die. We make the assumption that the bus interconnecting the dies of a flash channel and the corresponding channel controller represent no bottleneck, in other words, the dies are not slowed down by their neighbor dies or the channel controller.

A further important feature of our model is that in the course of garbage collection, the valid pages that need to be copied from the selected block will all be written into a free block on the same die. Note that this is a prerequisite of using the copy-back function available in many flash technologies. Copy-back represents an efficient way of moving data between flash cells on the same die without burdening the bus.

If, for whatever reason, copy-back is not used for page relocation, the relocated data could be moved over the bus to a die on the same flash channel. Alternatively, it could be copied to a die on a different channel, which would involve multiple bus transfers. The extent to which data is moved affects delay and throughput, and impacts the system's energy consumption.

When new user data is written to flash, it is good practice for the SSD controller to select the target flash die such that over time all dies are equally utilized. Moreover, it generally makes sense to exploit the parallelism among multiple flash channels. However, when it comes to selecting the target die for relocating valid pages, it is an open question whether the benefits of load balancing and parallelism warrant the price of data movement paid in terms of performance loss and added energy consumption.

3.2 Model Structure and Parameters

As shown in Fig. 2, our model consists of a single server, which represents the flash die. There are four different queues in front of the server to temporarily hold the read, write, copy, and erase requests.

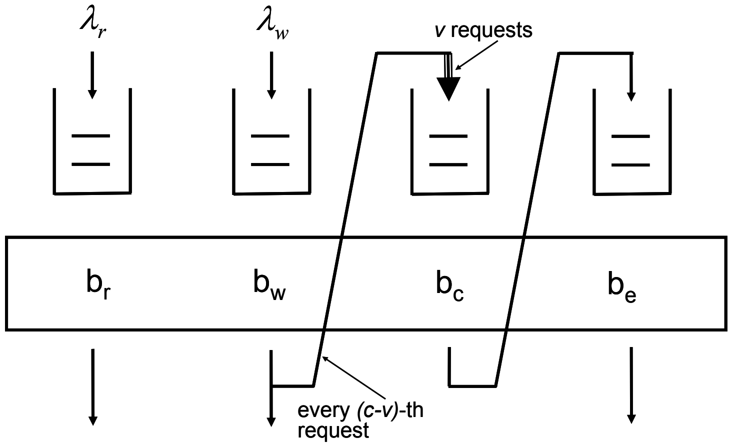

Fig. 2: Queueing model of single flash die

The stream of write arrivals represents the portion of the total SSD user write workload which the system controller decides to direct to the particular flash die considered. The stream of read arrivals are the user read requests for data stored on the die. The user read and write request arrivals are modeled as independent Poisson processes with rates λ_r and λ_w, respectively. Of course, our simulation model also supports arrival processes other than Poisson. In the context of this report, however, we will limit the discussion to this simple model.

In our model, user read and write operations occupy the die for a certain time to either read the data from the flash cells or program the cells. In addition, time is needed to transfer the data over the bus interconnecting the flash dies and the flash channel controller. During that time, also the die is busy. We assume that the total time for which the flash die is needed to process a user read request is constant and equal to b_r; similarly a user write request keeps the flash die busy for a constant time b_w.

As mentioned above, flash-based SSDs need to regularly perform garbage collection to free up storage space. When all *c* pages of a block have been written, the controller selects a new block from a free-block pool and initiates garbage collection to free up a hitherto used block. Using a specific selection mechanism, such as the "greedy" garbage-collection algorithm [2], the garbage-collection process selects one of the blocks for relocation. As the block selected usually contains a certain number *V* of pages with valid data, these valid pages first have to be copied into a free block, the new "write block". The remaining $c - V$ pages in the write block are available for storing user data. When all of these pages have been written, a new garbage-collection cycle is started, and the process repeats itself.

The garbage-collection process is modeled in the following way. In [2], it was shown that for large SSDs using greedy garbage collection, the number of valid pages copied per garbage collection typically varies very little. Although our simulation model supports more complex schemes, we assume for this report that the number of copied pages is constant and equal to *v*. This means that after $c - v$ user write requests have been processed, a total of *v* copy requests are generated and placed into the copy queue. Each of these requests represents the copying of a valid page, which is assumed to take a constant time b_c. Once all *v* copy operations have been performed, the server is busy for a time b_e to perform the erase function. In our model, we assume that the garbage-collection process keeps pace with the arrival of user write requests. In other words, we do not consider scenarios jn which the operation is slowed down by a momentary shortage of free blocks. We will come back to this issue in Section 6.

One of the critical SSD design issues is the assignment of priorities to the different types of requests competing for the flash die. A possible priority scheme, labeled RWP for "read/write priority", serves user reads and writes with higher non-preemptive priority over copy requests. At the other end of the spectrum is the "copy/erase priority" (CEP) scheme, in which the copy and ensuing erase operations take non-preemptive priority over user reads and writes. The impact of these two schemes on the performance of the system will be discussed in detail below. For the analysis described in this report, we assume that user read and write requests are served in strict first-come, first-served order, irrespective of their (read or write) nature.

4. Maximum Throughput

The traffic load on the flash die consists of four components:

(i) die utilization due to user writes

$$\rho_w = \lambda_w b_w \qquad (1)$$

(ii) die utilization due to user reads

$$\rho_r = \lambda_r b_r = s\lambda_w b_r \quad \text{with } s = \frac{\lambda_r}{\lambda_w} \tag{2}$$

(iii) die utilization due to valid-page copies

$$\rho_c = \lambda_c b_c = \frac{v}{c-v}\lambda_w b_c \tag{3}$$

(iv) die utilization due to block erases

$$\rho_e = \lambda_e b_e = \frac{1}{c-v}\lambda_w b_e \ . \tag{4}$$

The total die utilization is the sum of the above four components:

$$\rho = \rho_w + \rho_r + \rho_c + \rho_e = \lambda_w\left(b_w + sb_r + \frac{v}{c-v}b_c + \frac{1}{c-v}b_e\right) . \tag{5}$$

The write throughput approaches its maximum as the total utilization approaches 100%. This limit is reached for a maximum write arrival rate of

$$\lambda_w^{\max} = \left(b_w + sb_r + \frac{v}{c-v}b_c + \frac{1}{c-v}b_e\right)^{-1} . \tag{6}$$

The maximum total throughput is given by

$$\lambda^{\max} = \lambda_w^{\max} + \lambda_r^{\max} = (1+s)\lambda_w^{\max} \ . \tag{7}$$

Inserting (6) into (7) yields

$$\lambda^{\max} = \frac{(1+s)}{b_w + s\,b_r + \frac{v}{c-v}b_c + \frac{1}{c-v}b_e} \tag{8}$$

For storage systems, such as flash-based SSDs or log-structured arrays or file systems [5, 6], that use out-of-place write and garbage collection, the efficiency of the garbage-collection mechanism can be characterized by the so-called “write amplification” *A*, i.e., the ratio of the total write requests on the system and the user write requests [3]. For a system with *c* pages per block and a number of *v* valid pages that need to be copied per garbage collection, the write amplification is given by

$$A = \frac{c}{c - v} \quad . \tag{9}$$

Inserting (9) into (8) yields

$$\lambda^{\max} = \frac{(1+s)}{b_w + s\,b_r + (A-1)\,b_c + \frac{A}{c} b_e} \tag{10}$$

Inspection of Eqs. (8) and (10) leads to the following basic insights: The maximum throughput of a flash die decreases for longer write/read/copy/erase times. It also decreases as the write amplification increases. The throughput is not totally independent of, but quite insensitive to, the number of pages per block.

5. User Read and Write Latencies

In this section, we study the latencies of user read and write requests occurring in the flash die for a variety of design options and workloads. For the results included in this report, we used the model parameters listed in Table 1, which are based on the characteristics of a typical MLC flash die.

Table 1: Flash characteristics underlying the performance results

page size	B	4320
data transfer rate	MB/s	166
pages per block		256
page read	µs	76.3
page write	µs	926.4
page copy	us	950.7
block erase	µs	3000.3

We first discuss the basic latency characteristics with the help of plots showing simulated waiting times and queue lengths as a function of time. Then we address the issue of mean waiting times under different priority schemes.

5.1 Basic Latency Characteristics

For the examples discussed in this section, we assume a workload consisting of a mixture of 67% reads and 33% writes, i.e., s = 2. The mean inter-arrival time of reads is 1 ms and that of writes is 2 ms. This results in a die utilization of 0.54 by user reads and writes, and a total die utilization of 0.71 (including the garbage-collection overhead). We assume a moderate write amplification A of 1.33, which at 256 pages per block translates into *v* = 64 valid pages to be copied per garbage-collection cycle.

Figure 3 shows the outcome of a simulation run in which the valid-page copy and the ensuing block-erase operations were processed with higher non-preemptive priority over the user reads and writes, i.e., with the so-called CEP scheme introduced above. Shown in the chart are the individual delays of reads and writes waiting to access the die. The figure exhibits a distinctly cyclical behavior. There are phases of "normal" operation, during which the delays randomly oscillate around an average in the milliseconds range. However, when garbage collection is performed, the first user read or write request affected may have to wait until all *v* (64) valid pages in the relocated block have been copied and the block has been erased. Consequently, that first read or write request may suffer a delay of *v* times the duration of a copy operation plus one block erase time. Subsequent reads or writes will see gradually decreasing delays, unless the arrival of additional user reads or writes leads to an increase of the individual delays. As the figure shows, after garbage collection, it takes a considerable amount of time until the delays are back to their normal level. As additional simulations have shown, latency spikes of the kind shown in Fig. 3 occur also at much lower die utilizations, e.g., of 0.2 or less.

Figure 4 shows the behavior of the read/write queue length, which exhibits the same cyclical characteristic as the waiting times in Fig. 3.

In Fig. 5, we show the number of copy operations waiting to be processed by the die for the same experiment and observation window. As the copy operations are served with higher priority than the other requests, the length of the copy queue exhibits an almost completely deterministic behavior. Comparison of Figs. 3 and 4 with Fig. 5 reveals that read/write waiting times and queue lengths peak at that point in time when the last copy operation of a garbage-collection cycle and the ensuing block erase have terminated.

Clearly, the occurrence of pronounced latency spikes like the ones in Fig. 3 are highly undesirable and should be avoided. One possible way to avoid the spikes is to reverse the priorities and give user reads and writes higher non-preemptive priority than valid-page copies and block erases, i.e., to use the so-called RWP (read/write priority) scheme.

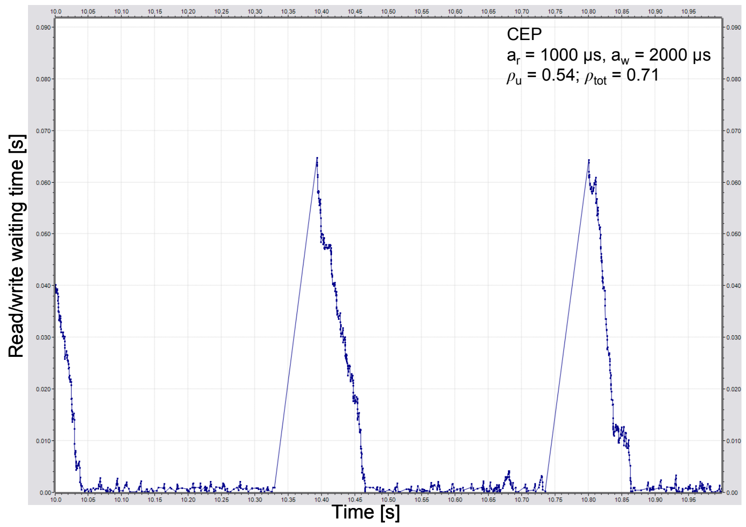

Fig. 3: User read and write waiting times for the CEP scheme

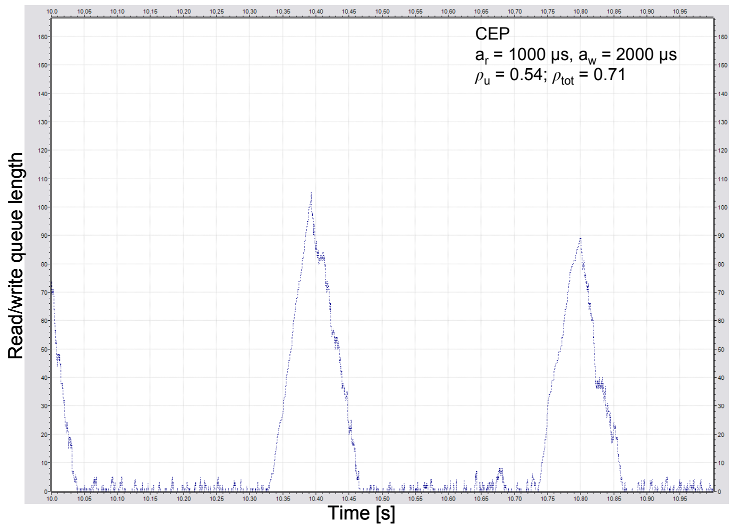

Fig. 4: User read and write queue lengths for the CEP scheme

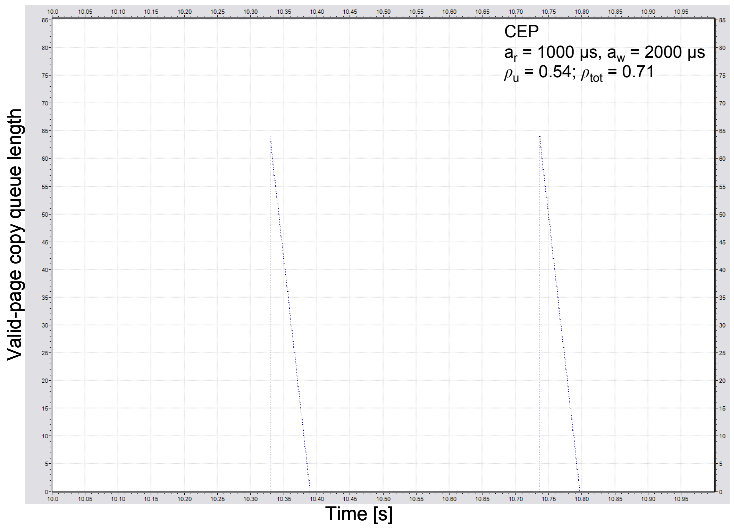

Fig. 5: Valid-page copy queue length for the CEP scheme

Figure 6 shows how the change of priorities affects the waiting times of the read and write requests. We observe a dramatic reduction of the delays together with a complete elimination of the latency spikes and cyclical behavior observed for the CEP scheme. (Please note the different scales on the vertical axes of Figs. 3 and 6.) This behavior is also reflected in the dynamics of the read/write queue length depicted in Fig. 7.

Figure 8 shows the behavior of the valid-copy queue for the RWP scheme. In contrast to the CEP scheme (Fig. 5), the decrease of the copy queue length under RWP may get randomly delayed by newly arriving user reads or writes. Consequently, it may take considerably longer until the last copy operation of a garbage collection cycle has been performed.

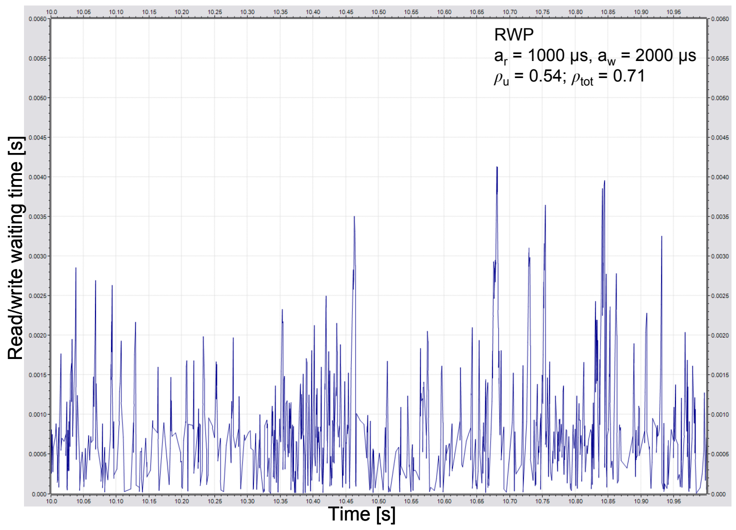

Fig. 6: User read and write waiting times for the RWP scheme

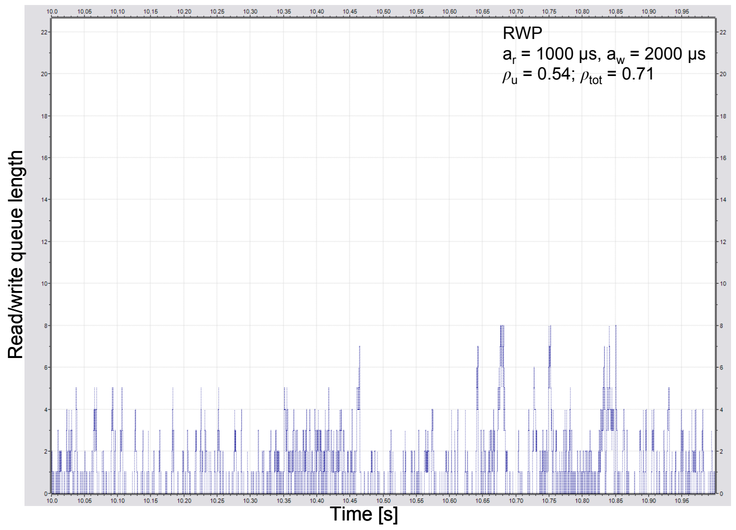

Fig. 7: User read and write queue lengths for the RWP scheme

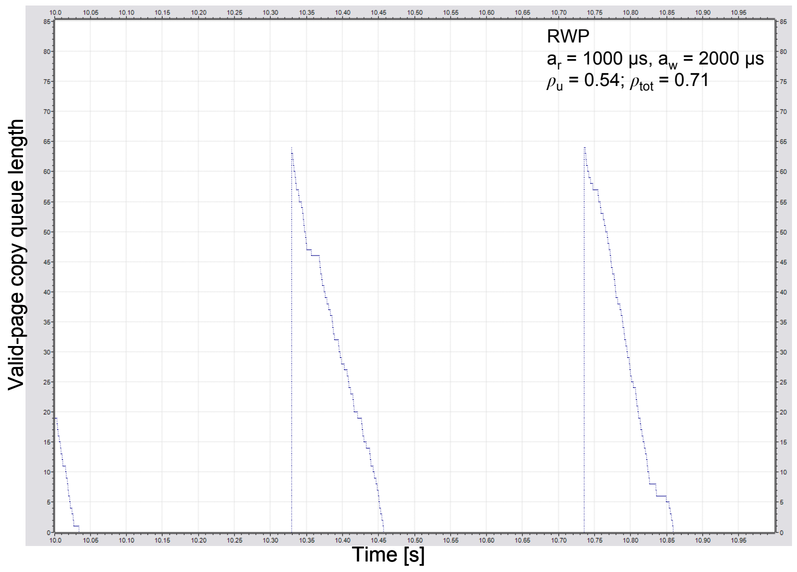

Fig. 8: Valid-page copy queue length for the RWP scheme

5.2 Mean Read and Write Waiting Times

For a better understanding of the system performance, it would be helpful to have a closed-form expression for the expected read and write waiting times for the two priority schemes RWP and CEP. To this end, we will simplify the model by cutting the coupling of the copy requests with the user writes. While in reality *v* copy requests are generated after the service completion of every (*c* – *v*)-th user write, we assume that there is no such coupling, but an independent arrival stream of batches of copy requests with batch size *v*. The arrival rate of the copy batches is

$$\lambda_{cb} = \frac{\lambda_w}{c-v} \ , \tag{11}$$

and the total arrival rate of copy requests is

$$\lambda_c = \frac{v}{c-v}\lambda_w \ . \tag{12}$$

5.2.1 Priority Scheme RWP

Under RWP, an arriving user read or write request can either start service immediately when the server is idle or has to wait for the residual service time of a read or a write request. It then has to wait until all reads and writes queued in front of it have been served. Yet another possibility is that it has to wait until a copy request or an erase request that is currently being served has been completed.

If we denote by w_{RWP} the expected waiting time under the RWP scheme, the expectations of the above times can be captured by the following relationship:

$$w_{\mathrm{RWP}} = \frac{\lambda_r b_r^2}{2} + \frac{\lambda_w b_w^2}{2} + \Omega_r b_r + \Omega_w b_w + \frac{\lambda_c b_c^2}{2} + \frac{\lambda_e b_e^2}{2} \tag{13}$$

In Eq. (13), Ω_r and Ω_w stand for the mean queue lengths of read and write requests, respectively. Using Little's law, we obtain

$$\Omega_r = w_{\mathrm{RWP}} \lambda_r \quad and \quad \Omega_w = w_{\mathrm{RWP}} \lambda_w \ . \tag{14}$$

Inserting (14) into (13) and solving for w_{RWP} finally yields

$$w_{RWP} = \frac{1}{2(1-\lambda_r b_r - \lambda_w b_w)} \left(\lambda_r b_r^2 + \lambda_w b_w^2 + \frac{v}{c-v} \lambda_w b_c^2 + \frac{1}{c-v} \lambda_w b_e^2 \right) . \tag{15}$$

Figure 9 shows both simulation and analytic results for the mean waiting time under the RWP scheme. As can be seen, our closed-form expression (15) yields amazingly accurate results up to rather high utilizations.

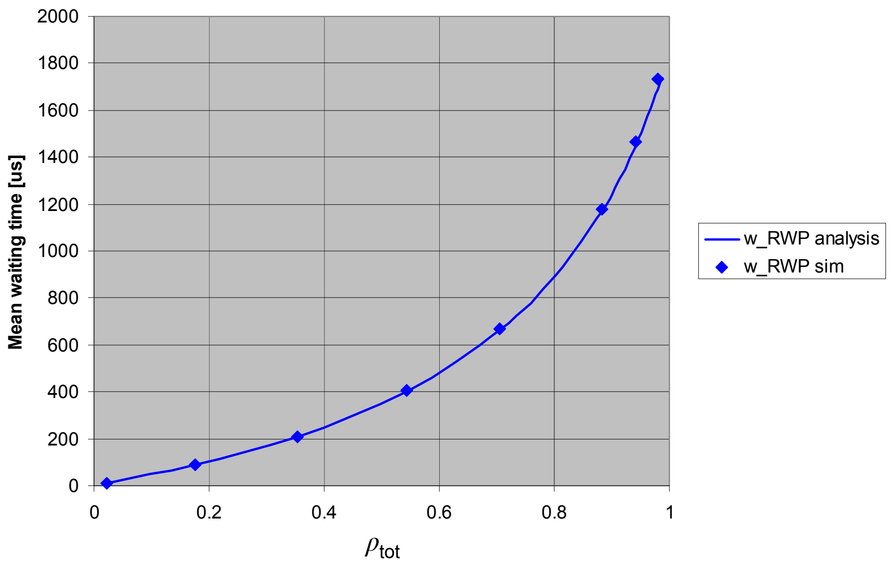

Fig. 9: Mean waiting time as a function of the total system utilization for the RWP scheme

5.2.2 Priority Scheme CEP

In Fig. 10, we show simulation results for the mean waiting time under the CEP scheme. Comparison with Fig. 9 reveals that, as expected, the CEP scheme leads to dramatically higher user delays than RWP.

To come up with a closed-form expression for the mean waiting time under CEP, we have used the same approach as above, namely, to cut the coupling of the copy and write requests and to look at the model as an M/G/1 queue with two (non-preemptive) priority classes, where the arrival rate of the higher-priority class would be set equal to the arrival rate of the copy batches and the service time of the higher-priority class to the sum of the copy service times and the erase time. In this approximate model, the mean waiting time of the read and write requests would be equal to the mean waiting time of the lower-priority class, which is well known [7]. However, comparison with simulation shows that this approach systematically overestimates the mean waiting times of the read/write requests under the CEP scheme. The primary reason for the mismatch is the assumption of exponentially distributed interarrival times of the copy request batches. In reality, this time corresponds roughly to the sum of *v* independent write interarrival times, i.e., an Erlangian distribution with parameter *v*. To the best of our knowledge, there is no theoretical result or analytical approach available in the literature that could be straightforwardly applied to this kind of queuing model.

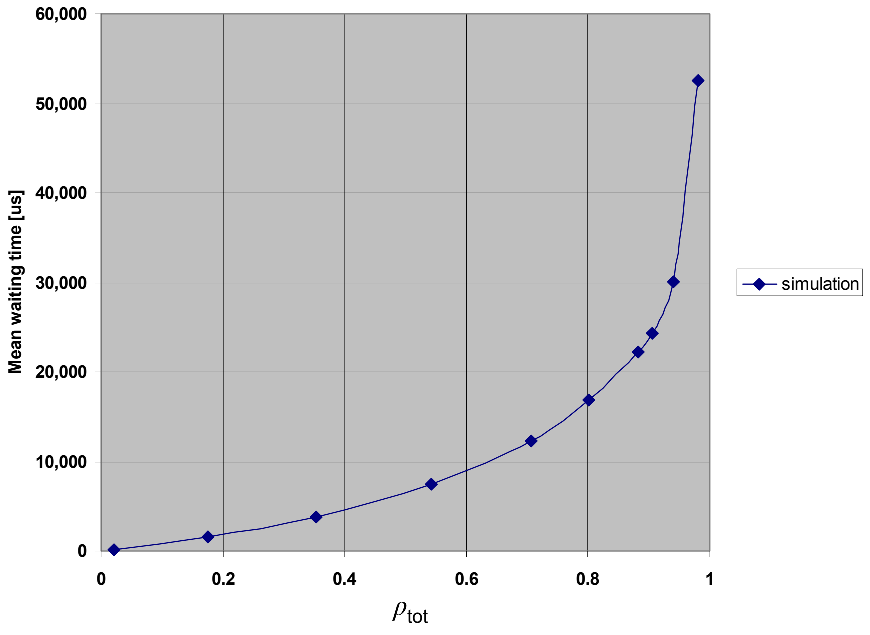

Fig. 10: Mean waiting time as a function of the total system utilization for the CEP scheme

6. The Duration of Garbage Collection

6.1 Basic Observations

As explained above, it is essential for the functioning of the SSD device that free blocks are available for storing user data in a timely fashion. In this section, we therefore study the question how long it takes an SSD system to perform garbage collection depending on its characteristics and the priority scheme employed.

We denote by “garbage-collection duration” the time from the initialization of garbage collection until all valid pages of the selected block have been relocated and the block has been erased.

Under the CEP scheme, garbage collection takes the minimum amount of time possible, viz. the aggregate of *v* copy operations and one erase operation:

$$\gamma_{CEP} = vb_c + b_e \quad , \tag{16}$$

Figure 11 shows a typical example of simulated garbage collection durations under the CEP scheme. The majority of the measured durations correspond to (16); a few are prolonged by either one read or one write processing time. These small variations are an artifact of our specific simulation implementation, which at the start of a garbage-collection action serves a previously scheduled read or write request despite its lower priority, before starting the (high-priority) page-copy operations. Modulo this small anomaly, the simulation reconfirms the ideal behavior of the CEP scheme relative to the garbage-collection duration.

In comparison, the dynamics of garbage collection under the RWP scheme are much more complicated, because waiting user read and write requests may delay the start of the valid-page relocations and, in addition, read or write requests arriving during the garbage-collection activity may delay its completion in a random fashion. Figure 12 shows the corresponding results for the RWP scheme. The comparison of the two figures illustrates that garbage collection under RWP takes distinctly longer than under CEP. (Note the different scales on the vertical axes.)

In Fig. 13, we show how the mean garbage-collection duration varies as a function of the total die utilization. (The latter is the total utilization caused by user reads and writes, valid-page copies, and erases.) While the garbage collection duration under CEP stays at its minimum value, the RWP duration remains reasonably small up to relatively high loads, but increases sharply as the total load approaches the system capacity. This is a distinct shortcoming of the RWP scheme, which suggests that pure, unmodified RWP is no viable priority scheme for SSDs. It has to be complemented by a scheme that enforces finite garbage-collection periods under any workload.

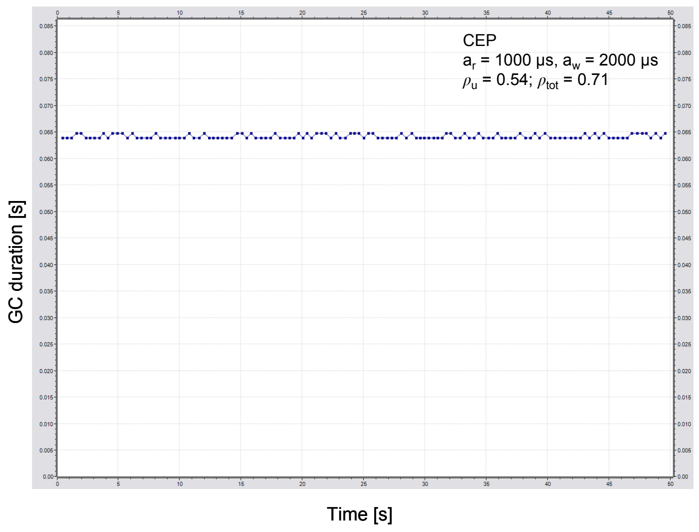

Fig. 11: Garbage-collection duration for the CEP scheme

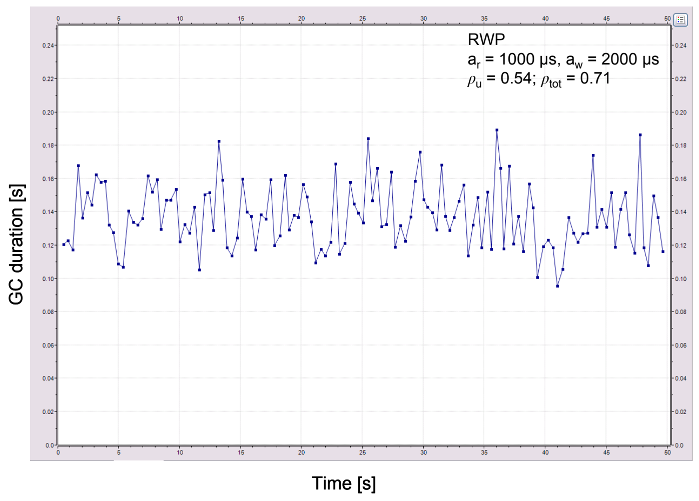

Fig. 12: Garbage-collection duration for the RWP scheme

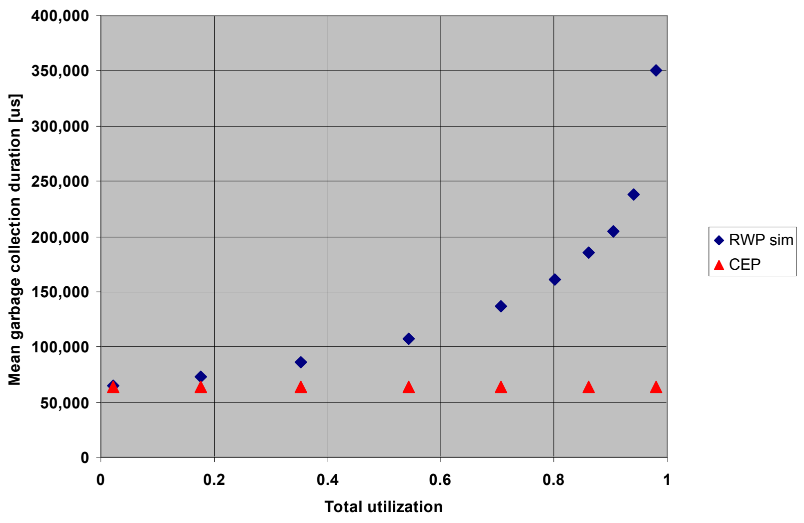

Fig. 13: Mean garbage-collection duration as a function of the total system utilization for the RWP and CEP schemes (simulation results)

6.2 Analysis of the RWP Garbage-Collection Duration

The aim of this section is to derive a closed-form expression for the mean garbage-collection duration under the RWP scheme. For simplicity, we limit the analysis to the traffic load region for which garbage collection periods do not overlap. More precisely, we assume that, at the arrival instance of the last write request before garbage collection is triggered, the preceding garbage collection has been completed. For this to be true, the offered user traffic must be well below the maximum throughput value derived in Section 4. In other words, we should not expect the analytic result to be applicable to high loads.

First we determine the expected time from the arrival of the last user write before garbage collection is triggered until the end of the garbage collection.

As user-write arrivals follow a Poisson process, the "last" user write will observe unfinished work in the system whose expectation corresponds to the mean waiting time of an M/G/1 queue with FIFO service discipline and two types of customers: reads arriving with rate λ_r and constant service time b_r, and writes arriving with rate λ_w and constant service time b_w. Therefore the initial unfinished work amounts to

$$\tau_0 = \frac{\lambda_r b_r^2 + \lambda_w b_w^2}{2(1-\lambda_r b_r - \lambda_w b_w)} \quad . \tag{17}$$

The expected time τ from the arrival of the "last" user write until the completion of the valid-page relocation consists of the following components:

- The initial unfinished work τ_0,
- the service time of the "last" user write b_w,
- the service times of the *v* valid-page copies vb_c,
- the expected service times of the user reads arriving during this period ($\tau\ \lambda_r b_r$), and
- the expected service times of the user writes arriving during this period ($\tau\ \lambda_w b_w$).

Setting the sum of these components equal to τ, inserting (17) and solving for τ yields:

$$\tau = \frac{1}{1-\lambda_r b_r - \lambda_w b_w}\left(b_w + vb_c + \frac{\lambda_r b_r^2 + \lambda_w b_w^2}{2(1-\lambda_r b_r - \lambda_w b_w)} \right) \quad . \tag{18}$$

To determine the mean garbage collection duration, two more steps are required:

1. We need to subtract from τ in (18) the time from the arrival of the “last” write request until its service completion. Because according to our assumption the prior garbage-collection backlog has ended before the arrival of the “last” write request, the only requests waiting in the system, if any, are read or write requests. Hence the time until the “last” write request has been served corresponds to the waiting time of a request in a FIFO M/G/1 queue with two Poisson arrival streams of rate λ_r and λ_w and service times b_r and b_w, respectively, plus the service time b_w of the last write request. The expectation of that time is equal to τ_0 in (17) plus b_w.
2. We need to add the mean service time for the block erase operation *b_e*.

Subtracting b_w and τ_0 from τ and adding b_e yields the following result for the mean garbage-collection duration:

$$\begin{aligned}\gamma_{RWP} &= \frac{1}{1-\lambda_r b_r - \lambda_w b_w}\left(b_w + v b_c + \frac{(\lambda_r b_r^2 + \lambda_w b_w^2)}{2(1-\lambda_r b_r - \lambda_w b_w)} \right) - \frac{(\lambda_r b_r^2 + \lambda_w b_w^2)}{2(1-\lambda_r b_r - \lambda_w b_w)} - b_w + b_e \\ &= \frac{1}{1-\lambda_r b_r - \lambda_w b_w}\left((\lambda_r b_r + \lambda_w b_w) b_w + v b_c + \frac{(\lambda_r b_r + \lambda_w b_w)(\lambda_r b_r^2 + \lambda_w b_w^2)}{2(1-\lambda_r b_r - \lambda_w b_w)} \right) + b_e \end{aligned} \quad ,(19)$$

In Fig. 14, we present analytic and simulation results for the mean garbage-collection duration for the same parameter values as before. The analytic results computed with Eq. (19) perfectly match the simulation results up to a total die utilization of 0.9. At higher loads, the analysis yields too small durations, which is, of course, a consequence of the underlying assumption that garbage-collection periods do not overlap.

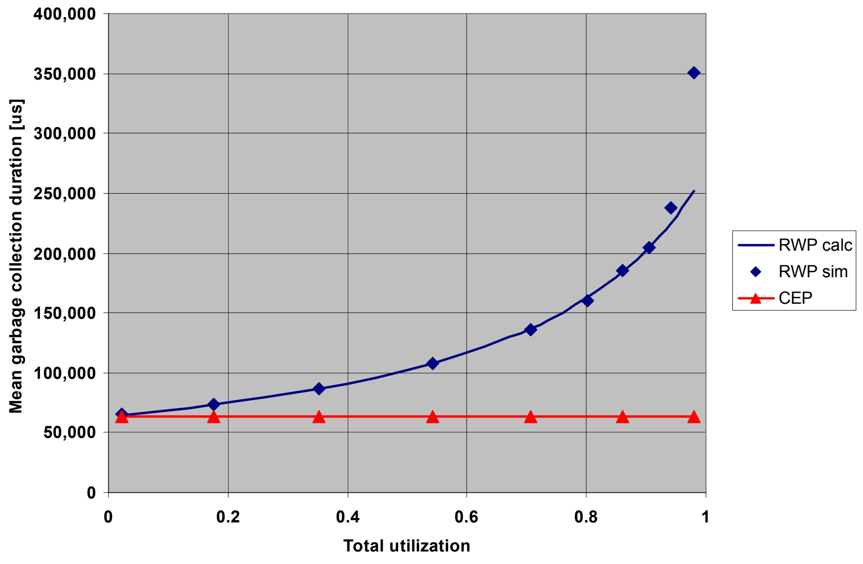

Fig. 14: Mean garbage-collection duration as a function of the total system utilization for the RWP and CEP schemes (comparison of analytic and simulation results)

7. The Garbage Collection Backlog Duration

In this section, we look at the behavior of the SSD system from a different angle. The issue we want to address is how long it takes until the impact of a garbage-collection action is no longer felt. Specifically, we will ask ourselves the question: What is the duration of the "Garbage Collection Backlog"? We define the latter as the time from the start of garbage collection until the next instance the die becomes idle. The die is idle when all valid-page copies, erases, user reads and writes – present at the start of garbage collection or arriving thereafter while the die is still busy – have been processed.

For the same set of parameters as in the earlier examples, Fig. 15 shows the backlog durations measured during the first 50 sec of a simulation experiment. Driving the system with identical samples of the read/write request arrival process, the experiment was performed for both priority schemes. The measured backlog durations turned out to be exactly identical for both schemes.

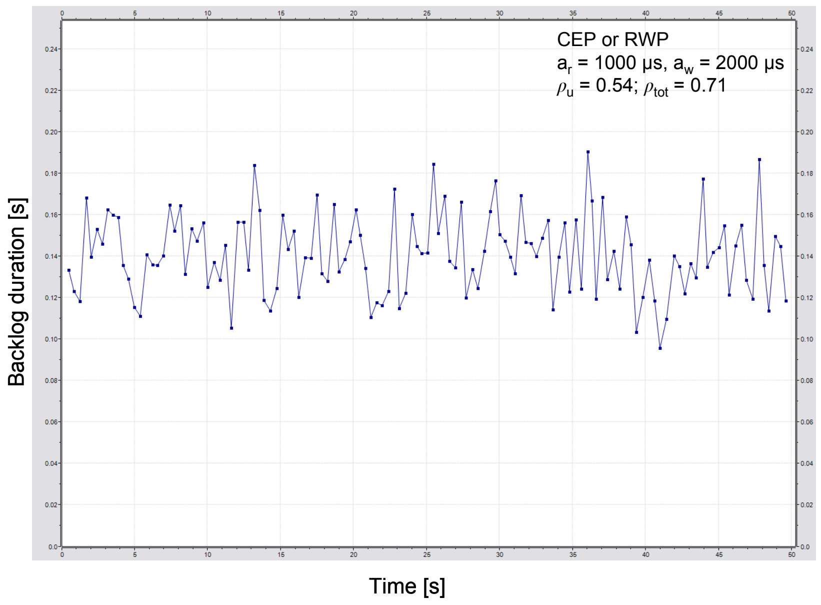

Fig. 15: Garbage-collection backlog duration for the CEP or RWP schemes (0.71 total utilization)

Before interpreting these results, we present an additional example in which the same experiment as before was conducted, but under a 33% higher workload. For the CEP scheme, Fig. 16 shows that a considerable fraction of the backlogs last longer than the time between two successive garbage-collection instances. As the total die utilization is 0.94, this does not come as a surprise. What is more interesting, though, is that if we repeat the same experiment under the RWP scheme, we obtain almost exactly the same results, see Fig. 17.

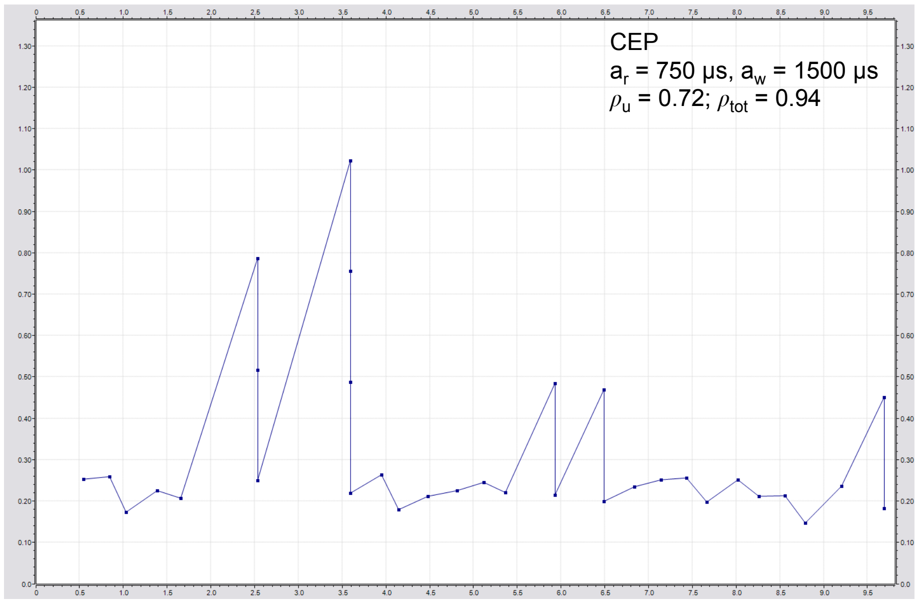

Fig. 16: Garbage-collection backlog duration for the CEP scheme (0.94 total utilization)

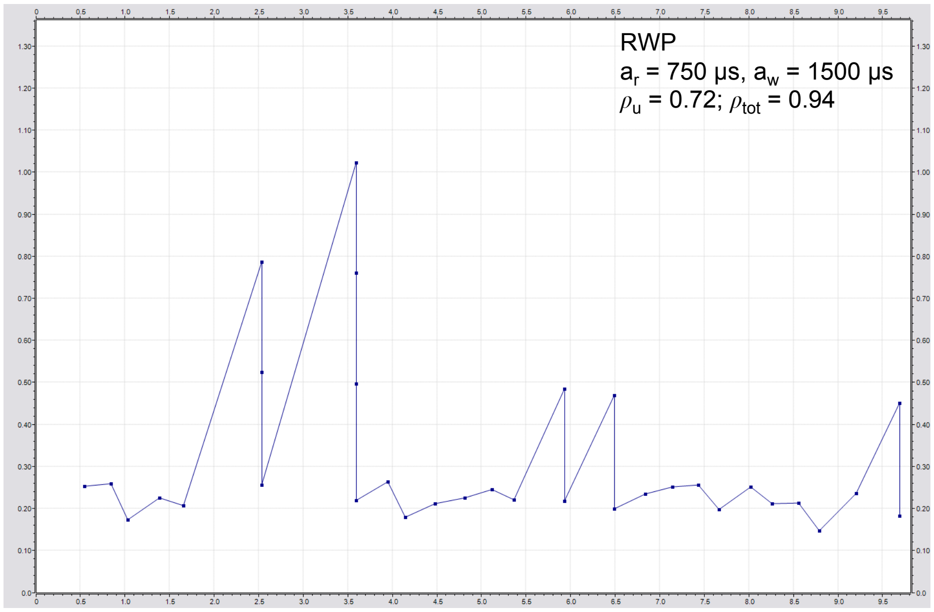

Fig. 17: Garbage-collection backlog duration for the RWP scheme (0.94 total utilization)

The only small differences, which are hardly visible when comparing the two figures, are in the durations of backlogs that started at points in time when the backlog from the preceding garbage collection had not completely finished. For these cases, the backlog is slightly shorter under the CEP scheme. This effect can be explained as follows: For both schemes, the last write request before garbage collection sees exactly the same total backlog upon its arrival, but the mixture of the unfinished work will differ for the two priority schemes. Specifically, under CEP, there will likely be more pending reads and writes in the system than under RWP. Moreover, as the remaining waiting times for the queued-up reads or writes tend to be longer, if they are served with lower priority, the waiting time of the “last” write will be longer. Hence it will take longer until the “last” write request has been served, and garbage collection commences. We have already seen that the backlog durations end at exactly the same points in time under both schemes. Consequently, the garbage-collection backlogs under CEP, which tend to start later but end at the same point in time as those under RWP, will tend to be shorter.

The invariance of the garbage-collection backlog with respect to the priority scheme employed should not come as surprise because of its close relationship with the unfinished work in a work-conserving single-server queueing system. It is well known that the unfinished work, also called work backlog, is independent of the order of service [4].

8. Summary and Conclusions

The main findings of the work described in this report can be summarized as follows:

- The maximum throughput of a flash die decreases for longer write, read, copy, and erase times. It also decreases as the write amplification increases. The throughput is not totally independent of, but quite insensitive to, the number of pages per block.

- Relative to user read and write latencies, we found that (i) RWP results in distinctly shorter mean waiting times than CEP, and (ii) garbage collection under CEP may cause massive latency spikes, even under light load.

- Our findings for the garbage-collection duration can be summarized as follows: (i) Under CEP, garbage collection takes the minimum amount of time possible, independent of the workload. (ii) Under RWP, the garbage-collection duration remains small up to high utilizations. However, it increases sharply (and theoretically without limits), as the load approaches the system capacity.

- The garbage collection backlog durations are invariant with respect to the priority assignment to reads/writes, valid-page copies, and erases, with the exception of garbage-collection periods commencing when previous backlogs are still present. This holds for all work-conserving service disciplines

Based on these observations, we reach the following basic conclusions:

- Because of its consistently short garbage-collection duration, CEP is a robust priority strategy. However, it is deficient relative to user read and write latencies.
- RWP yields the best user read/write latencies and, in particular, avoids latency spikes.
- Pure RWP is no viable priority strategy for SSDs, because under overload, garbage collection could last arbitrarily long. It needs to be complemented by a mechanism which ensures that, under any workload, garbage collection is performed (i.e., initiated <u>and</u> completed) in sync with the consumption of free blocks.
- The effect of garbage collection on the total unfinished work in the system ("garbage-collection backlog") cannot be influenced by the priority strategy. Therefore the need for overload protection/backpressure is the same for any (work-conserving) priority scheme

References

[1] J. Brewer, M. Gill (ed.), Nonvolatile Memory Technologies with Emphasis on Flash: A Comprehensive Guide to Understanding and Using Flash Memory Devices, Wiley-IEEE Press, 2008.

[2] W. Bux and I. Iliadis, Performance of Greedy Garbage Collection in Flash-Based Solid-State Drives, Performance Evaluation 67(2010), pp. 1172-1186.

[3] X.-Y. Hu, E. Eleftheriou, R. Haas, I. Iliadis, and R. Pletka, Write Amplification Analysis in Flash-based Solid State Drives, in: Proceedings of the Israeli Experimental Systems Conference (SYSTOR) (Haifa, Israel), 2009, pp. 1–9.

[4] L. Kleinrock, Queueing Systems, Vol. I: Theory. John Wiley & Sons, New York, 1975.

[5] J. Menon, A Performance Comparison of RAID-5 and Log-structured Arrays, in: Proceedings of the 4th International Symposium on High Performance Distributed Computing (HPDC) (Charlottesville, VA), 1995, pp. 167–178.

[6] M. Rosenblum and J. K. Ousterhout, The Design and Implementation of a Log-structured File System, ACM Trans. Comput. Syst. 10(1) (1992) 26–52.

[7] H. Takagi, Queueing Analysis, Vol. 1: Vacation and Priority Systems, Part 1. North-Holland, Amsterdam, 1991.