Scaling out to a Single-Node 80Gbps Memcached Server with 40Terabytes of Memory

Michaela Blott, Ling Liu, Kimon Karras
Xilinx Research, Dublin

Kees Vissers
Xilinx Research, San Jose

Abstract

Current web infrastructure relies increasingly on distributed in-memory key-value stores such as memcached whereby typical x86-based implementations of TCP/IP compliant memcached yield limited performance scalability. FPGA-based data-flow architectures overcome and exceed every other published and fully compliant implementation in regards to throughput and provide scalability to 80Gbps, while offering much higher power efficiency and lower latency. However, value store capacity remains limited given the DRAM support in today's devices.

In this paper, we present and quantify novel hybrid memory systems that combine conventional DRAMs and serial-attached flash to increase value store capacity to 40Terabytes with up to 200 million entries while providing access at 80Gbps. This is achieved by an object distribution based on size using different storage devices over DRAM and flash and data-flow based architectures using customized memory controllers that compensate for large variations in access latencies and bandwidths. We present measured experimental proofpoints, mathematically validate these concepts for published value size distributions from Facebook, Wikipedia, Twitter and Flickr and compare to existing solutions.

1 Introduction

Distributed in-memory key-value stores (KVS) have become a critical piece of middleware in most common web infrastructures, typically implemented with commodity servers which have known performance limitations. Critical bottlenecks include the interrupt-intensive nature of the TCP/IP stack, limited instruction cache sizes, synchronization overheads and L3-cache ineffectiveness due to lack in data locality[13].

In 2013, data-flow architectures for KVS have been introduced demonstrating 10Gbps line-rate implementations on FPGAs (Field Programmable Gate Arrays)[4], processing 13.02 million request per seconds (RPS), with a power efficiency of 254.8KRPS/Watt and round-trip latencies of below 4.5 microseconds (us). Furthermore, these architectures can provide scalability to much higher data rates by for example increasing data path width from the existing 64b to a 512b implementation operated at 156MHz. This is feasible given that current implementation consumes less than 3% of the available resources of today's largest Xilinx FPGA device (VU440). However, value store capacity, a key figure of merit, remains limited by the FPGA's total amount of IO pins as each DRAM channel requires around 150 IO pins.

In this paper, we demonstrate how hybrid memory systems can be leveraged to scale storage capacity to achieve multiple Terabytes (TB) while maintaining 80Gbps of access bandwidth. The key idea is to leverage high capacity memory that connects via high-speed serial interfaces in combination with DRAM to create a mixed storage of DRAM and flash. For example, flash-based solid state drives (SSDs) provide up to 1TB for 4IO pins using SATA3 protocol, while being lower in power consumption and cost. (Write endurance is in this context of less concern as most key-value store use cases have a low update requirement, however it will impact lifetime.) Similarly, when the FPGA is hosted in one of IBM's OpenPower systems[1], memory access can be expanded to 1TB of the Power8's host memory through their coherent accelerator processor interface (CAPI) without taking up additional IO pins. However, these types of hybrid memory systems are not further considered in this paper.

This concept would cause a significant performance hit with standard x86 architectures where multithreading cannot effectively hide the large access latencies of flash (See discussion in section 5.2). However, in conjunction with data-flow architectures and application-specific memory controllers, we are able to effectively amortize large access latencies and handle large variations (10s of nanoseconds for DRAM, and up to over 100usec for SSD).

Many different object distribution schemes are possible. For example object distribution on basis of popularity are frequently used where highly popular (hot) objects are stored or cached in DRAM and less popular items in flash. This only works well for use cases with data local-

[1] www.openpowerfoundation.org

Figure 1: First prototype system

ity where most hot objects can be contained within the actual DRAM storage. In this paper, we consider a distribution on the basis of object size. The idea is that all values that are larger than an SSD's page size are stored in SSDs, and only smaller values in DRAM[2]. By accessing SSDs in page sizes, we can service at significantly higher bandwidth from flash [9]. Furthermore, for many use cases [3, 13], smaller objects tend to be more popular and therefore require higher access bandwidth which works well as the overall available SSD bandwidth is less than a third of the available DRAM bandwidth. To satisfy aggregate bandwidth requirements we can introduce over-provisioning as discussed in section 2. Simultaneously, it is the large size objects that contribute to the majority of the storage requirements and quickly exhaust DRAM capacity (see use cases in Table 2). Thereby this scheme plays to the strengths of each memory type and is key to scale to high access bandwidth while providing large capacity. As part of our initial investigation, we have built an actual 10Gbps prototype system, as shown in figure 1, validating that the proposed dataflow architecture and customized memory controllers, as shown in figure 2, can effectively handle the large variance in bandwidth and latency. We measured throughput on all data paths including DRAM and SSD access and use these measurements to subsequently extrapolate accurate performance and capacity for 80Gbps systems.

We validate the architecture and the distribution scheme for a wide range of well-known memcached value distributions [3, 13], with regards to access bandwidth and storage capacity for SSD and DRAM, thereby demonstrating how 80Gbps KVS implementations with 40TB of memory can be created in a single server. In addition to this significantly increased compute and storage capacity, we expect a reduction in system latency and power.

In section 2, we introduce the proposed architecture and present the first experimental proofpoint in section 5. In section 4, we derive the formulas to calculate capacity and access bandwidth requirements for the memory system given a specific value size distribution. We then describe example 80Gbps system configurations for a set of published workloads, compare them to existing implementations. Finally, we discuss related work in section 6 before concluding the paper.

[2]In fact the threshold is completely configurable and can be adjusted to suit the use case.

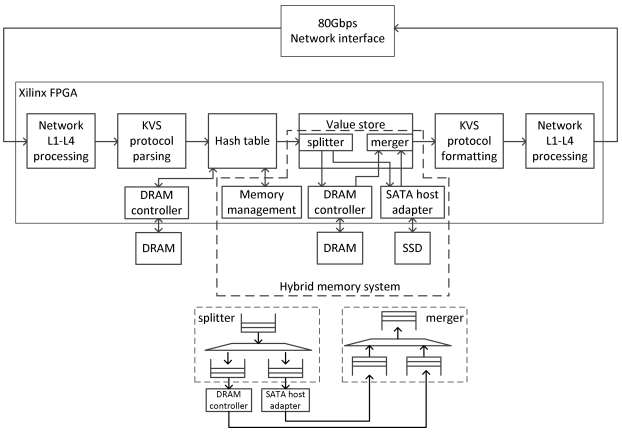

Figure 2: System architecture for hybrid memory system

2 System architecture

In this section, we explain the system-level architecture of the proposed 80Gbps key-value-store, implemented through a combination of data-flow architecture, customized memory controller and hybrid memory system. As shown in figure 2, packets are first received through a *network interface* which handles all related layer 1 to 4 processing including UDP and TCP, then processed by a *parser*, *hash table*, *value store* and *response formatter* before being transmitted by *network interface*[4]. Through this arrangement, many packets can be processed back-to-back, exploiting task-level parallelism, while memory synchronization overheads are minimized through the in essence serialized execution of all memory accesses.

For the hybrid memory system, we partition the value store into two main areas, one that resides within DRAM and one within SSD. The *memory management* logic is responsible for distributing values between DRAM and SSDs and allocates/de-allocates addresses in DRAM or SSD on the basis of a value's size. Within the value store, the data path contains a *splitter* which routes memory requests to either DRAM or SSD and a *merger* which simply aggregates the results back. Slab and cache management strategies are implemented in basic form and not further addressed within this paper. The key challenges when mixing memory types are differing access speeds and latencies. Figure 2 illustrates that this can be managed by introducing an out-of-order memory controller with a split in the data path that allows faster, lower latency, accesses to bypass slower, higher latency, accesses. This is acceptable with the KVS context as request order does not have to be preserved. Furthermore, we budget conservatively access bandwidth to SSD and DRAM for predefined sets of use cases assuming a uniform distribution of requests. Extreme asymmetric access cases that fall outside the predefined limits, can exhaust access bandwidth in individual memory or SSD channels and thereby throttle performance. To address this, multiple SSDs, with optionally duplicated contents for popular objects, can be used in parallel to essentially overprovision access bandwidth limits [15].

3 First experimental results

All of our prototyping efforts were conducted on AlphaData's ADM_PCIE_7V3 platform which uses a combination of 2 DDR3 channels of 8GB each, 2 SATA3 SSDs of 128GB each and one 10Gbps Ethernet interface. We used a Spirent testcenter C1 [21] for traffic generation and monitoring. Given the platform's constraints, we were able to validate only the typical value size distribution for Twitter at 10Gbps link rate serviced from DRAM and 1 SSD, the latter servicing all 4kB values. After an initial ramp-up, we observed that the response traffic fully saturates a 10Gbps Ethernet interface whereby the read bandwidth from the SSD is below 25% utilization. In regards to latency, we measure a total round-trip latency between 2.0us (for 2B objects) and 115.8us (for 4096B objects) independent of network load and write intensity. Power consumption of total system including power regulators, FPGA, DRAM and SSD amount to 27.2Watt. Section 4 and 5 demonstrates how we can extrapolate from these measurements.

4 Memory requirements

In this section, we derive the equations for memory access bandwidth and capacity for both DRAM and SSD for different value size distributions, whereby both value store and hash table accesses are considered. Here and in the following equations, "ht" denotes hash table, "vs" value store, "d()" the capacity and "bw()" the bandwidth function. The required capacity for the hash table is derived as the product of the number of entries (*entries*) and the size of one entry, which is the sum of value store address (*vs*) and the maximum size of a key (*max_ks*) multiplied by the maximum bucket size *bs* [10]: *d(ht)* = *entries* * (*vs* + *max_ks*) * *bs*.
Assuming that $prob_{s_i}$ represents the likelihood with which a value of size s_i appears within a given distribution, then the following equation derives the capacity requirements for SSD (DRAM is equivalent, only difference that we need to add capacity of hash table.)[3]

$$d(SSD) = \Big(\sum_{s_i \in value\_size\_pool} d_{s_i}(SSD)\Big) * entries \quad (1)$$

$$d_{s_i}(SSD) = \begin{cases} 0, & \text{if } s_i < ps \\ s_i * prob_{s_i} & \text{otherwise} \end{cases}$$

In regards to bandwidth calculations, the most relevant operations are GET and SET operations. The hash table memory bandwidth requirement needs to consider a read from the hash table for both GET and SET operations and a write-back for SETs. The value store memory bandwidth is comprised of either reading (GET operation) or writing of the value (SET operation). For a given value size s_i and its distribution $prob_{s_i}$, whereby $prob_{SET}$ represents the percentage of SET operations, the following equations calculate the bandwidth requirements for the hash table and value store:

$$\begin{aligned} bw_{s_i}(ht) &= (bw_{s_i}(get\_read\_ht) * (1 - prob_{SET}) \\ &\quad + (bw_{s_i}(set\_read\_ht) + bw_{s_i}(set\_write\_ht)) \\ &\quad * prob_{SET}) * prob_{s_i} \\ bw_{s_i}(get\_read\_vs) &= pkt\_rate(max(get\_rsp\_pkt, get\_rqt\_pkt)) * s_i * 8 \\ bw_{s_i}(set\_write\_vs) &= pkt\_rate(max(set\_rsp\_pkt, set\_rqt\_pkt)) * s_i * 8 \end{aligned}$$

where,

$$\begin{aligned} bw_{s_i}(get\_read\_ht) &= pkt\_rate(max(get\_rsp\_pkt, get\_rqt\_pkt)) \\ &\quad * average\_ks * 8 * bs \\ bw_{s_i}(set\_read\_ht) &= pkt\_rate(max(set\_rsp\_pkt, set\_rqt\_pkt)) \\ &\quad * average\_ks * 8 * bs \\ bw_{s_i}(set\_write\_ht) &= bw_{s_i}(set\_read\_ht) \\ bw_{s_i}(vs) &= (bw_{s_i}(get\_read\_vs) * (1 - prob_{SET}) \quad (2) \\ &\quad + bw_{s_i}(set\_write\_vs) * prob_{SET}) * prob_{s_i} \quad (3) \end{aligned}$$

From this, we can derive the bandwidth requirements for SSD and DRAM separately:

$$bw(DRAM) = \sum_{s_i \in value\_size\_pool} (bw_{s_i}(DRAM) + bw_{s_i}(ht)) \quad (4)$$

$$bw_{s_i}(DRAM) = \begin{cases} 0, & \text{if } s_i \geq ps \\ bw_{s_i}(vs) & \text{otherwise} \end{cases}$$

$$bw(SSD) = \sum_{s_i \in value\_size\_pool} bw_{s_i}(SSD) \quad (5)$$

$$bw_{s_i}(SSD) = \begin{cases} 0, & \text{if } s_i < ps \\ bw_{s_i}(vs) & \text{otherwise} \end{cases}$$

5 Evaluation

To evaluate our scale-out concept, we use our measured data from the prototype and consider a selection of typical use cases from recent publications [13, 3]. From their respective value size distribution and percentage of SET operations, we derive DRAM and SSD access bandwidth and capacity requirements assuming a 80Gbps line rate implementation. Due to lack of details in the published material, we assume a uniformly distributed access probability across object size. We check these results against the physical limitations of the FPGA-based platform. In case of a Virtex Ultrascale (VU440), we can extrapolate the following constraints: *bw(DRAM)* = 492Gbps, *bw(SSD)* = 128Gbps, *d(DRAM)* = 256GB, *d(SSD)* = 40TB [4]. This we compare then against standard x86 and FAWN-style systems and show first experimental results.

Table 1 summarizes a broad range of memcached value size distributions. Each row represents a different use case, and the individual fields represent $prob_{s_i}$ and $prob_{SET}$. The first use case, which is dominated by small value sizes, is derived from numbers published by Facebook in [3], where key and value size characteristics of memcached workloads are given in cumulative distribution function diagrams. The second use case shows the value size distribution of Twitter, where values, comprised of a tweet itself and associated meta-data, create

[3] As discussed in section 2, when s_i is smaller than page size *ps*, then the value resides within DRAM, otherwise in SSD.

[4] assuming a maximum of 8 DDR4 DRAM 32GB DIMM interfaces and 40% effective throughput on the 2400Mbps interfaces, 40 SATA3 SSDs and 8 high-speed serial IOs used for Ethernet connectivity

Value size (Bytes)	128	256	512	768	1024	4096	8192	32K	1M	$prob_{SET}$
Facebook	0.55	0.075	0.275	0	0	0	0	0	0.1	3%
Twitter	0	0	0	0.1	0.85	0.05	0	0	0	20%
Wiki	0	0	0.2	0.1	0.4	0.29	0.008	0.001	0.001	1%
Flickr	0	0	0	0	0	0.9	0.05	0.03	0.02	0%

Table 1: Value size distribution and write rates in published memcached use cases

Use case	d(DRAM) [GB]	d(SSD) [GB]	DRAM bw utilization	SSD bw utilization	Maximum entries[M]	SSDs
Facebook	254	20,000	17%	12%	200	20
Twitter	238	25	16%	61%	120	2
Wiki	244	343	12%	92%	150	8
Flickr	250	6,000	1%	98%	240	25

Table 2: Hybrid memory system evaluation results

average object sizes of 1KB with little variance. The average size of a Wikipedia object, an individual article in HTML format, is 2.8KB with high variance, whereby typical Flickr values, low resolution photo objects like thumbnails, are on average 25KB [13].

5.1 Evaluation of hybrid memory system

The evaluation shows that all use cases can be catered for. Assuming $average\_ks$ = 32B, max_ks = 256B, vs = 4B, bs = 4, and ps =4096B, and that we saturate DRAM capacity first, we can calculate the maximum number of entries that can be supported using the given platform constraints and applying them to the equations from section 4. With this, we can then determine the overall effective value store capacity and access bandwidth requirements. Table 2 summarizes the results: For the presented use cases, we can support up to 200 millions of entries and effective value stores up to 20TB out of a possible 40TB, thereby significantly increasing the capacity of a single memcached server. The added benefit varies with the use case. For example for Twitter the added benefits of the SSDs implementing a minimum of 25GB is marginal. However, in all cases, the proposed system can sustain the 80Gbps access rate. As all collective interface bandwidths of every partition (SSD or DRAM partition) exceed the worst case bandwidth requirements for 80Gbps incoming network traffic, we can cater any value size distribution with the proposed memory system as well as any level of write intensity. The only potential bottleneck is when data locality produces asymmetric access bandwidth requirements where multiple consecutive accesses hit the same address space. As discussed in section 2, additional SSDs and DRAM channels can be used to over-provision for this. Finally, for value size distributions with larger values, effective value storage can scale up to 40TB.

5.2 Comparison to existing solutions

In this section, we compare the FPGA-based implementation to existing key-value store solutions comprised of x86 and FAWN architectures in combination with DRAM and flash. For FPGA-based solutions, we show 3 numbers: *80Gbps design* represents the theoretical maximum numbers for a 80Gbps 40TB KVS implementation; *80Gbps facebook* shows achievable numbers on the basis of the previously defined Facebook use case. Only the effectively used storage area and performance (calculated as the with $prob_{s_i}$ weighted sum of the maximum transaction rate for different object sizes) are given. In both cases, power consumption is extrapolated as follows: We assume 30W total for a standalone base platform (derived from prototype) augmented with a small embedded processor for memory management, and 0.878Watt/GB for DRAM and 4.5Watt maximum per SSD [6, 23, 20]. The third FPGA number, *10Gbps prototype*, shows actual performance and power measurements of the in section 3 described prototype.

For comparison, we consider FlashStore [6], which is as far as we know the highest performing flash-based key-value store using x86 and SSDs. FlashStore numbers are given for specific use cases with different SET:GET ratios and different object size distributions whereby the best case numbers are referenced here. Please note that performance numbers refer to a dual-socket installation which is compared to a single-chip FPGA solution. As second comparison pointer, we use MICA [12], which we believe to be the highest performing key-value store so far on the basis of a dual-socket x86 system with DRAM. Again, we reference the highest published number given for a specific use case (uniform workload with 50% GETs). In regards to power consumption, we assume thermal design power per specified processor on a dual-socket motherboard [14], typical power consumption per network interface card [18] multiplied by the number of cards (8), and 878mWatt per GB of DRAM. The final comparison data pointer is a FAWN architecture with SSDs as described in [2] whereby the quoted performance nunber represents one specific workload with 256B queries.

Table 3 summarizes the results: When comparing with MICA, performance of 1 FPGA is roughly comparable to 2 Xeon E5-2680 processors with some variation depending on use case specifics. The key differentiator is the memory capacity, as the FPGA can offer the performance in conjunction with flash and thereby scale to 40TB and provide much improved power efficiency per GB. FlashStore is perhaps the closest comparison pointer and illustrates the significant performance drop that multicore processors exhibit in conjunction with SSDs. FAWN similarly demonstrates power efficiency with flash technology in comparison to DRAM-based solutions, however significantly lacks in performance.

One of the concerns of this approach is fault tolerance, as the storage capacity per node is condensed and the time to backup/recover a complete node becomes large.

Platforms	GB	KRPS	Watt	KRPS/Watt	GB/Watt
FPGA (80Gbps design)	40,000	104,000	434.8	239.2	92.0
FPGA (80Gbps facebook)	20,254	32,657	343	95.2	59.0
FPGA (10Gps prototype)	272	1,340	27.2	49.2	10.0
Dual x86 (MICA)[12]	64	76,900	477.6	161	0.1
Dual x86 (FlashStore)[6]	80	57.2	83.5	0.7	1.0
FAWN (SSD) [2]	32	35	15	2.3	2.1

Table 3: Comparsion

Considerations of this will be part of future work.

6 Related work

KVS and NoSQL databases such as memcached, Redis [16] and MongoDB [19] have been an active field of research in the past years. Many x86 optimizations [11, 12] have been published which overcome the TCP/IP stack bottleneck by using RDMA or Intel's DPDK[5] instead. Accelerating memcached in heterogeneous systems has been investigated by others such as Chalamalasetti et al.[5], Lim et al. [13] and Lockwood [17]. However, none of these publications consider the use of hybrid memory systems. Schooner's optimized memcached implementation has found its way into products from IBM [8], Fusion IO [1] and SanDisk [24] offering a performance optimized, highly-concurrent version of memcached which runs on x86 processors and has been specifically designed to work with up to 1TB of flash storage, however offers much less performance in comparison to the proposed implementation. Further flash-based KVS implementations exist and have been described in [6, 22, 7]. Again, these solutions are limited by the x86-based implementation and service much lower transaction rates as explained in [13]. As far as we know, we are the first to consider hybrid memory systems in conjunction with data-flow architectures for KVS which leverage customized memory controllers to amortize larger access latencies, thereby serving very high request rates and while offering Terabytes of storage.

7 Conclusion

In this paper, we measured an actual prototype and presented a method on how to scale out to a single-server, FPGA-based data-flow architecture combined with an SSD-assisted hybrid memory system which achieves sustained 80Gbps while serving 100s of millions of entries and up to 40TB in storage capacity. The key contributions are around the distribution of values between DRAM, SSDs and host memory, the customized memory access architecture which can handle the differences in access bandwidth and latency, and a formalized approach, that allows to quantify the optimal system specifications for given value size distributions. This has been validated for a group of well-known use cases and compared to existing platforms.

[5] dpdk.org

References

[1] Fusion io, using membrain as a flash-based cache: `http://www.fusionio.com/blog/scale-smart-with-schooner`.

[2] ANDERSEN, D. G., FRANKLIN, J., KAMINSKY, M., PHANISHAYEE, A., TAN, L., AND VASUDEVAN, V. Fawn: A fast array of wimpy nodes. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles* (2009), ACM, pp. 1–14.

[3] ATIKOGLU, B., XU, Y., FRACHTENBERG, E., JIANG, S., AND PALECZNY, M. Workload analysis of a large-scale key-value store. *SIGMETRICS Perform. Eval. Rev. 40*, 1 (jun 2012), 53–64.

[4] BLOTT, M., KARRAS, K., LIU, L., VISSERS, K., ISTVAN, Z., AND BAR, J. Achieving 10Gbps line-rate key-value stores with FPGAs. In *Procedings of HotCloud '13 (5th USENIX Workshop on Hot Topics in Cloud Computing), June 25-26, 2013,* (San Jose, CA, USA, 2013).

[5] CHALAMALASETTI, S. R., LIM, K., WRIGHT, M., AUYOUNG, A., RANGANATHAN, P., AND MARGALA, M. An FPGA memcached appliance. In *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays* (New York, NY, USA, 2013), FPGA '13, ACM, pp. 245–254.

[6] DEBNATH, B. K., SENGUPTA, S., AND LI, J. Flashstore: High throughput persistent key-value store. *PVLDB 3*, 2 (2010), 1414–1425.

[7] FACEBOOK. Mcdipper: A key-value cache for flash storage: https://www.facebook.com/notes/facebook-engineering/mcdipper-a-key-value-cache-for-flash-storage/10151347090423920.

[8] IBM. Schooner appliance for memcached: http://www-03.ibm.com/systems/x/solutions/schooner/memcached.html.

[9] ISAACS, R., AND ZHOU, Y., Eds. *2008 USENIX Annual Technical Conference, Boston, MA, USA, June 22-27, 2008. Proceedings* (2008), USENIX Association.

[10] ISTVÁN, Z., ALONSO, G., BLOTT, M., AND VISSERS, K. A. A flexible hash table design for 10gbps key-value stores on FPGAS. In *FPL* (2013), pp. 1–8.

[11] KALIA, A., KAMINSKY, M., AND ANDERSEN, D. G. Using RDMA efficiently for key-value services. In *Proceedings of the 2014 ACM Conference on SIGCOMM* (New York, NY, USA, 2014), SIGCOMM '14, ACM, pp. 295–306.

[12] LIM, H., HAN, D., ANDERSEN, D. G., AND KAMINSKY, M. Mica: A holistic approach to fast in-memory key-value storage. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)* (Seattle, WA, Apr. 2014), USENIX Association, pp. 429–444.

[13] LIM, K. T., MEISNER, D., SAIDI, A. G., RANGANATHAN, P., AND WENISCH, T. F. Thin servers with smart pipes: designing SoC accelerators for memcached. In *ISCA* (2013), pp. 36–47.

[14] ONLINE. `http://ark.intel.com/products/64583`.

[15] ONLINE. `http://www.sandisk.com/assets/docs/wp004_overprovisioning_whyhow_final.pdf`.

[16] ONLINE. `http://redis.io/`.

[17] ONLINE. `http://www.hoti.org/hoti22/tutorials/#tut3`.

[18] ONLINE. `http://www.intel.ie/content/dam/doc/product-brief/ethernet-server-adapter-x520-t2-brief.pdf`.

[19] ONLINE. `http://www.mongodb.org/`.

[20] ONLINE. `http://www.samsung.com/us/system/consumer/product/mz/7k/e1/mz7ke1t0bw/850PRO.pdf`.

[21] ONLINE. `http://www.spirent.com/Ethernet_Testing/Platforms/C1_Chassis`.

[22] OUYANG, X., ISLAM, N. S., RAJACHANDRASEKAR, R., JOSE, J., LUO, M., WANG, H., AND PANDA, D. K. SSD-assisted hybrid memory to accelerate memcached over high performance networks. In *ICPP* (2012), pp. 470–479.

[23] PARK, S.-Y., KIM, Y., URGAONKAR, B., LEE, J., AND SEO, E. A comprehensive study of energy efficiency and performance of flash-based SSD. *Journal of Systems Architecture - Embedded Systems Design 57*, 4 (2011), 354–365.

[24] SANDISK. Accelerate end-user response times with a flash-optimized enterprise cache: `http://www.sandisk.com/products/enterprise-software/membrain/`.