Mlcached: Multi-level DRAM-NAND Key-value Cache

I. Stephen Choi, Byoung Young Ahn, and Yang-Suk Kee
Samsung Memory Solutions Lab

Abstract

We present Mlcached, multi-level DRAM-NAND key-value cache, that is designed to enable independent resource provisioning of DRAM and NAND flash memory by completely decoupling each caching layers. Mlcached utilizes DRAM for L1 cache and our new KV-cache device for L2 cache. The index-integrated FTL is implemented in the KV-cache device to eliminate any in-memory indexes that prohibit the independent resource provisioning. We show that Mlcached is only 12.8% slower than a DRAM-only Web caching service in the average RTT with 80% L1 cache hit while saving two-thirds of its TCO. Moreover, our model-based study shows that Mlcached can provide up to 6X lower cost or 4X lower latency at the same SLA or TCO, respectively.

1 Introduction

Various Web services commonly deploy in-memory caches such as Memcached [9] or Redis [15] to reduce latency and/or lower total cost of ownership (TCO) of the back-end databases. Due to the strong data locality in the real world [4], a low-cost caching server hides the latency to the back-end servers for data retrieval or computation and saves much higher cost for scaling up the back-end system to accommodate all requests. Assuming the latency (i.e. response time) differences, more than two orders of magnitude, between a caching server and a back-end system, achieving a higher hit rate is critical to enhance user experience and to reduce the overall cost of operations. For example, increasing hit rate by 1% would reduce the average latency by 25% [6]. Therefore, higher hit rates from Web caching services are highly desirable.

Unfortunately, scaling up key-value (KV) caching services to achieve higher hit rates, i.e. increasing DRAM capacity of a given node or nodes, is challenging due to its proportionally increasing energy consumption and the DRAM capacity limits. DRAMs are expensive and power-hungry from the resource perspectives in data centers. For example, memory accounts for more than 25% of data-center energy [13] and takes up to 20% of the cost per node [5].

Moreover, scaling DRAM density is challenging and a DIMM count per channel is limited [12]. This, *memory-capacity wall*, indicates that only limited amount of DRAM can be equipped in commodity servers. As a result, to put more DRAM per node, the node needs to get larger, i.e. more expensive machines. Therefore, the overall cost and power consumption is an exponential function of memory capacity as shown in Figure 1. Unfortunately, this will only get worse because DRAM capacity scaling is slower than logic [12]. Hence, reducing the DRAM requirement is critical to reduce the overall cost of Web caching services.

Meanwhile, the speed and cost effectiveness of non-volatile memory technology have been improved significantly. In particular, solid-state drives (SSD) that utilize NAND flash memory bridge the gap between memory and traditional rotating disks, i.e. HDD, due to its superior performance, energy efficiency, and energy proportionality over HDDs [17]. In fact, there have been significant work to utilize SSDs for a KV cache/store [3, 2, 7, 8, 11, 16, 10, 14]. However, hybrid approaches, i.e. mixed usage of DRAM and SSDs to implement a KV cache/store, are prone to impose *redundant* mappings to store indexes for accessing KV pairs on SSDs introducing large memory usages.

Such in-memory indexes prohibit efficient resource utilization of memory. This is because memory cannot be effectively used to store KV pairs in memory due to the indexes for flash memory resulting in significant waste of memory as the average size of values are relatively small. Moreover, *independent* resource provisioning is hard to achieve due to such dependency between different resources, DRAM and flash memory, in data centers.

In this paper, we propose Mlcached, multi-level

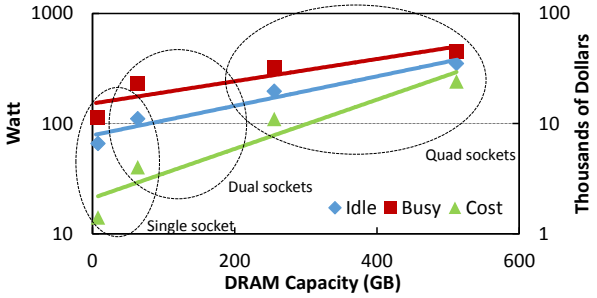

Figure 1: Power consumptions while running Memcached and system costs of *single-*, *dual-*, and *quad-*socket Dell servers.

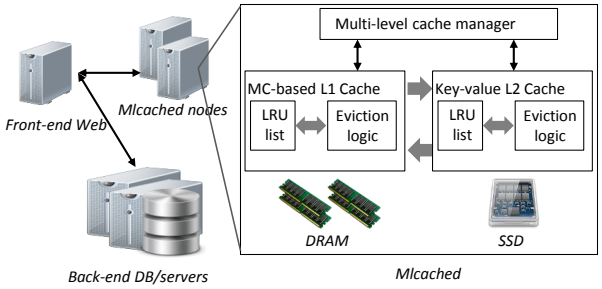

Figure 2: Mlcached design.

DRAM-NAND KV cache that is designed to enable independent resource provisioning of DRAM and NAND flash memory. We take a holistic approach to achieve optimal latency/cost of Web caching services by fully decoupling two caching layer and Mlcached has three main attributes of: 1) *multi-level cache hierarchy*: our multi-level caching service consists of two layers: DRAM-based L1 and NAND-flash-based L2 caches, 2) *completely decoupled levels*: each cache level solely depends on single technology and our KV cache device does not require any mapping areas or indexes in the host memory, 3) *indepedent resource provisioning*: we can control capacities of each cache layer independently allowing optimal resource allocations. Our model-based results show that Mlcached achieves 6X lower cost or 4X lower latency while meeting the service-level agreement (SLA) or TCO constraint, respectively. Moreover, we show that Mlcached achieves a very competitive performance in latency, showing 12.8% overhead with 80% L1 cache hit while saving the TCO of the DRAM-only cache by two thirds, which exhibits arguably the best performance at much higher cost.

2 Mlcached Design

Another caching layer that is cheaper provides us a room for further improvements in an SLA-TCO optimization. In particular, this problem can be described as either of these: *minimize* $Cost(C_m,C_s)$ *such that,* $Lat(C_m,C_s) \leq SLA$ or *minimize* $Lat(C_m,C_s)$ *such that,* $Cost(C_m,C_s) \leq TCO$, while $Cost(C_m,C_s)$ and $Lat(C_m,C_s)$ are defined in Equation 1 and 2, respectively.

$$Cost(C_m,C_s)^1 = P_m C_m + P_s C_s \quad (1)$$

where C_m, C_s, P_m, and P_s are capacity of memory and SSD, and price per unit capacity of memory and SSD, respectively.

$$\begin{aligned} Lat(C_m,C_s) = {} & L_n + L_m + L_s(1-H(C_m)) \\ & + (1-H(C_m+C_s))(2L_n+L_b) + L_n \end{aligned} \quad (2)$$

where L_n, L_m, L_s, L_b, and $H(C)$ are latency of network, response time of DRAM-only, SSD-only, and back-end systems, and hit rate at capacity C, respectively (the network topology is shown in Figure 2). For example, $Lat(C_m,C_s)$ can be calculated in $2L_n+L_m$ with *all* cache hits in the memory.

Based on Equation 1 and 2, we observe that multi-level KV cache, made of DRAM and NAND flash memory, can achieve up to 6X lower cost or 4X lower latency as shown in Figure fig:conf-a while maintaining SLA or TCO constraint, compared to our baseline (details will be discussed in Section 4). This is mainly due to the strong data locality, and two caching layers with different latency and cost allowing further improvements compared to the DRAM-only cache in such optimization.

Unfortunately, existing approaches for utilizing SSDs as an another caching layer do not achieve such optimal solution as storing KV pairs on SSDs intrudes effective use of memory. Such memory overhead is caused by the mappings between a key space and storage addresses, which are mandatory to maintain due to the traditional way to cope with storage devices: block interface. As shown in Figure 3-(a), storage devices are commonly mounted as a block device which is addressed via logical block addresses (LBA). Regardless of utilizing OS file systems, there exist key-to-LBA mappings in the KV applications as the form of in-memory indexes [7, 11, 16, 10]. As a result, this mapping consumes memory and it can take a significant portion of the available memory prohibiting effective use of it (i.e. reduced space for in-memory KV pairs).

Moreover, this is significant waste in resource provisioning of DRAM and flash memory in data centers. This is because certain amount of DRAM must be allocated to utilize SSDs while preventing effective use of memory to cache/store KV pairs. Such dependency is, in fact, one of

[1]For exclusive L2 cache only.

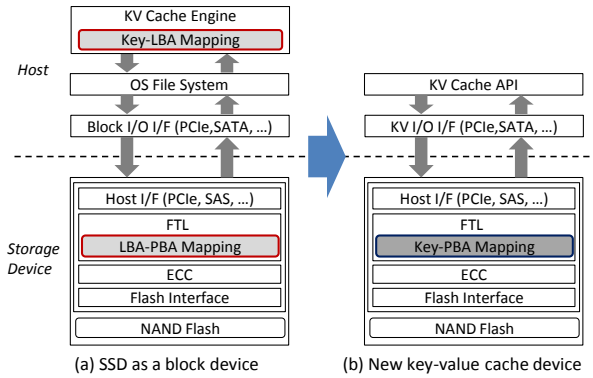

Figure 3: Redundant mappings when SSDs are used as a block device and our new key-value caching device to eliminate redundant mappings.

the key limiters towards a full utilization of the speed and cost effectiveness of NAND flash to complement DRAM memory. Therefore, we need to eliminate the wasteful memory overhead to achieve optimal SLA and TCO as described in Equation 1 and 2.

Mlcached removes such memory overhead completely and allows independent resource provisioning. In particular, Mlcached's cache hierarchy consists of two independent levels; L1 cache is similar to a common DRAM-based Memcached and L2 cache is a stand-alone caching layer made of a KV-cache device as shown in Figure 2. The KV-cache device is a commodity SSD that runs our new index integrated flash translation layer (FTL). As shown in Figure 3-(a), most FTLs in commodity SSDs are designed to translate LBAs to physical block addresses (PBA)[2]. In fact, such mapping inside FTL is redundant to the key-to-LBA mapping inside KV cache engine. As such, our new index-integrated FTL eliminates the redundancy while providing direct calls to the device using a key(s) as shown in Figure 3-(b). As a result, this device enables applications to directly send a key(s) instead of LBA to access KV pairs while eliminating redundant key-to-LBA mappings, i.e. in-memory indexes.

3 Mlcached Implementation

DRAM-based L1 Cache We utilize Memcached for our L1 caching layer because it is widespread and its protocol is simple enough to implement. We believe that any other DRAM-based KV cache can be easily fitted with minor efforts. Our prototype of Mlcached adopt

[2]This is because the unique characteristics of NAND flash; NAND flash does not allow overwrites while having asymmetric program/erase granularity (page for programming vs. block for erasure) and limited endurance (i.e. programing/erase cycles). Such characteristics require dynamic mappings between LBA and PBA. FTL also handles garbage collection and wear leveling

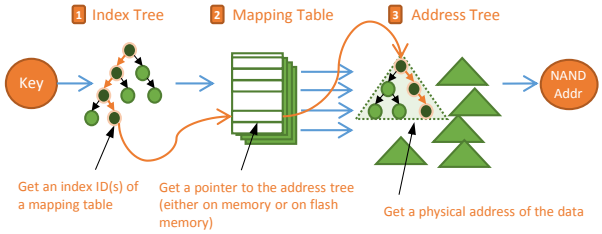

Figure 4: Index-integrated FTL. We employ a three-level data structure that consists of index tree, mapping table, and address tree.

the exclusive-inclusion property, i.e. each cache is exclusive. We extend *vanilla* Memcached's eviction logic to connect L1 and L2 caches.

NAND-based L2 Cache We utilize our new KV-cache device to implement a victim L2 cache. Our KV cache device uses a novel index-integrated FTL. Implementing key-to-PBA mappings inside FTL is challenging due to three main problems. First, in general, the size of a key space is much larger than LBA space. For example, 64-bit key has the combination of 2^{64} while 40 bits are enough to address 512 TB. Therefore, the size of indexes or mapping tables is much bigger than the size of traditional LBA-to-PBA mappings. Second, the average data size in KV caching services is much smaller than logical block or physical page sizes (512 B, 4/8 KB) [4]. This also results in much larger mapping table that cannot be directly loaded into the limited device DRAM. Lastly, it is critical to reduce the number of NAND accesses due to the difference in the access latencies of DRAM and NAND. In particular, it is necessary to guarantee the maximum number of NAND accesses to bound the latency in worst cases.

We employ a three-level tree-structure index to address such challenges. With a given key(s), it traverses through an index tree, a mapping table, and an address tree as shown in Figure 4. The index tree and mapping tables are guaranteed to reside on a device memory for a fast response time. Address trees can be placed either on the memory or on flash memory and recently accessed trees can be remained in the memory like a cache.

The other important aspects of our design are cache-algorithm integrated FTL and cache-aware garbage collection (GC). Our design employs an autonomous cache-eviction algorithm. That is, users can specify an eviction policy and our integrated FTL performs the cache replacements. Moreover, our GC takes advantage not only of expiration of KV pairs, if specified, but of a user programmable cut-off window for an aggressive capacity provisioning.

DRAM cost	10 ($/GB)	Latencies	
SSD cost	0.68 ($/GB)	Memcached	100 μs
Number of keys	4 Billion	KVD	200 μs
Size of data	256 Bytes	Back-end DB	10 ms
Distribution	Zipfian	Alpha of Zipf	3

Table 1: Parameters for model-based analysis.

4 Experimental Evaluation

Model-based Analysis We first evaluate cost-latency enhancements of Mlcached based on Equation 1 and 2, described in Section 2. We compare the *Pareto* optimal with cost or latency constraints of a traditional DRAM-only Memcached versus a completely decoupled multi-level DRAM-NAND KV cache. For this experiment, we use Zipfian distribution [1] to model a strong data locality. We fix the cost of DRAM to 10 and SSD to 0.68 dollar/GB. The target data set consists of four billion keys with 256-Byte value per each key, resulting in total data footprint of 1 TB (detailed parameters shown in Table 1). For comparisons, we fix the TCO and SLA to $1500 and 500 μs, respectively, considering common configurations[3] and requirement of a response time, i.e. sub-millisecond. Moreover, we set latencies, i.e. response times, of servers with DRAM-only Memcached, servers with KVD-only Memcached, and back-end DB to 100 μs, 200 μs, and 10 ms, respectively.

As shown in Figure 5, the multi-level approach outperforms the DRAM-only approach. This means that multi-level caching service can either save the TCO or reduce the SLA. In particular, a multi-level cache can improve the latency 4X or as low as 114 μs compared to the 460 μs latency of a traditional DRAM-only Memcached. In this case, the multi-level cache consists of 90 GB DRAM and 900 GB KVD, compared to the 143.4 GB DRAM for the traditional DRAM-only Memcached to meet the TCO. Meanwhile, for the fixed SLA of 500 μs, a multi-level cache can reduce the TCO by 6.4X, $206 compared to $1331. In this case, the multi-level cache employs 10 GB DRAM and 150 GB KVD while the other system requires 133.1 GB DRAM.

Measured System Performance We evaluate the overall system performance of Web caching service utilizing our Mlcached. We measure round-trip time (RTT) and throughput with random access distributions. Our prototype is evaluated on a single-socket Intel E5-1650 system. We disable Hyper Threading and Turbo features to get consistent results. To implement a prototype of Mlcached, we modified Memcached-1.4.24 which was the most recent stable release of Memcached. Also, we implement a prototype of our index-integrated FTL on a

[3]Facebook's standard servers Type-VI are equipped with 144 GB memory [18] and this costs about $1500 for the memory.

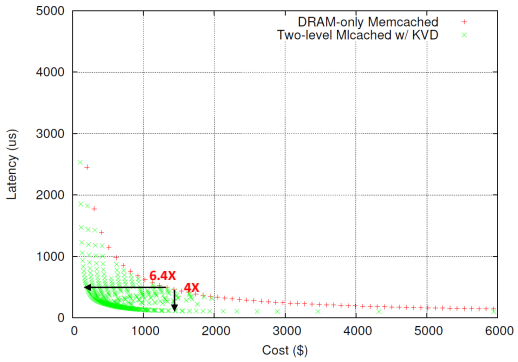

Figure 5: Benefits of Mlcached based on our cost-latency model.

Single-socket machine (cost: $1400)	
CPU	Intel Xeon E5-1650 @ 3.5 GHz 6 cores
Memory	Samsung DDR4 8 GB DRAM @ 2133 MHz
NIC	Intel X520-DA2 82599ES 10-Gigabit
Dual-socket machine (cost: $4000)	
CPU	Two Intel Xeon E5-2630 @ 2.3 GHz 12 cores
Memory	Samsung DDR3 64 GB DRAM @ 1333 MHz
NIC	Intel X520-DA2 82599ES 10-Gigabit
Software	Ubuntu 14.04 LTS Modified Memcached-1.4.24 Modified Facebook Mcblaster
Measurement	Yokogawa WT330 Digital Power Meter

Table 2: Experimental environments: server configurations.

Samsung NVMe PCIe SSD. We have two servers; one is for Mlcached and another for workload generation. Detailed configurations are shown in Table 2.

We measure the average RTT of Mlcached. Our baseline is the RTT of a dual-socket machine running *vanilla* Memcached-1.4.24 that we used for the implementation of Mlcached. On the other hand, Mlcached runs on a single-socket machine as shown in Table 2. We first measure the RTT of 0% L1 cache hit with 100% L2 cache hit, i.e. hit from the KVD. This presents how slow the KVD is compared to the RTT of DRAM-based Memcached. For this comparison, we set the capacity of L1 cache, i.e. DRAM capacity, to zero. Then, we vary the size of value. As shown in Figure 6, cache hits from KVD exhibit 2X additional latency resulting in 3X RTT for value size of 64 Bytes. Meanwhile, this gap decreases as the value size gets larger. For example, L2-cache hit show 73% longer RTT compared to the RTT of baseline for 4096-Byte values. The latency of KVD is somewhat constant up to the value size of 4 KB. This is mainly due to the NAND flash access time and its page size. KVD internally uses a page size larger than 4 KB (an actual page size varies depending on the SSD) and thus, accesses for data smaller than the page size will require the similar time needed to access the whole page.

Such higher L2 cache hit time might be negligible of-

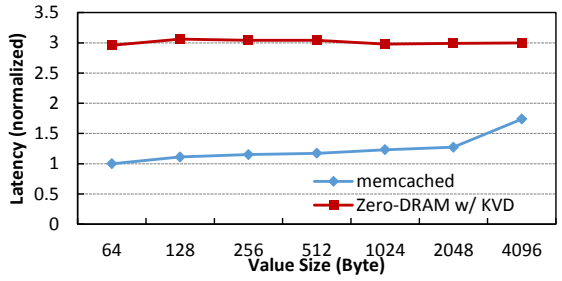

Figure 6: Round-trip time of Mlcached w/ 100% KVD hits.

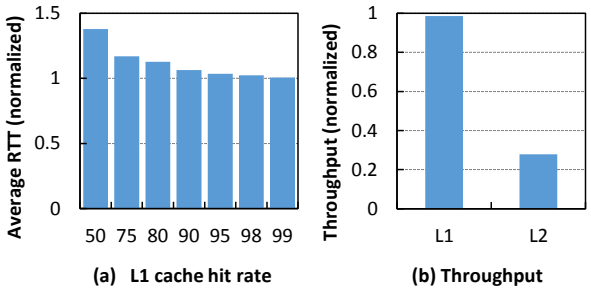

Figure 7: Average round-trip time with respect to the L1 cache hit rate and throughput of each caching layers.

ten times. This is because the strong data locality causes very high cache hit rates of L1 cache resulting in similar average RTTs. We measure the average RTTs by varying the hit rates of L1 cache. For this experiment, we vary the L1 capacity to control cache hit rates with 256-Byte values. As shown in Figure 7-(a), with higher L1-cache-hit rates, Mlcached exhibits lower RTTs. In particular, with 99% L1-cache hits, Mlcached shows less than 1% overhead while showing 38% with 50% L1-cache hits. It is notable that with the cache hit rate higher than 83%, Mlcached shows less than 10% increased RTTs.

We measure the throughput of each caching layer of Mlcached and compare it to our baseline, throughput of Memcached. As shown in Figure 7-(b), L1 cache of Mlcached shows the almost same throughput of Memcached. On the other hand, L2 cache of Mlcached, utilizing KVD, shows around 28% the baseline throughput. Considering the strong data locality (hence high cache hit rates of L1), such throughput is more than adequate to cope with the traffic, i.e. L1 cache misses.

5 Related Work

FlashStore [7] is utilizing an SSD as a cache between DRAM and HDD. It is designed to access a KV pair in the SSD by utilizing a RAM hash table index which stores pointers to KV pairs stored in the SSD. This enables a single SSD read to read values in the SSD. It also uses cuckoo hashing and compact key signatures to minimize DRAM usage for the hash table. In contrast, Mlcached completely removes a DRAM usage to store any kind of key-to-LBA mappings. This enables effective use of DRAM along with independent resource provisioning of DRAM and SSD.

SkimpyStash [8] and SILT [11] achieves very low memory footprint. SkimpyStash moves its index hash table into SSDs using a log structure while SILT combining three different types of key-value stores to accomplish such low memory usage. However, SkimpyStash requires multi-flash reads per lookup, resulting in a fraction of maximum performance of SSDs. Meanwhile, SILT consumes host CPU cycles and I/Os because it is utilizing a modular approach; first utilize a LogStore to achieve high write throughput, then the LogStore will be converted to a HashStore which is more memory efficient once the LogStore fills up (likewise, LogStores will be merged into a SortedStore). Such conversion and merging require background computation and heavy I/Os. However, Mlcached does not consume any I/Os or host CPU cycles for type conversion or merging.

Aerospike [16] is one of the most popular KV stores utilizing SSDs to achieve its high performance. Its data layer stores indexes in DRAM and it is designed to allow different configurations for its physical storage. For example, data can be stored any media among DRAM, SSDs, and HDDs. Moreover, SSDs can be used either as a block device without format or through VFS utilizing a large block to minimize latencies. It consumes a relatively large amount of memory per entry, 70 Byte/entry. McDipper [10] is an SSD-based photo caching layer of Facebook and provides good throughput while keeping the cost low. It also stores its hash bucket meta-data in the host memory while storing data, i.e. photos, in the SSDs. In contrast, Mlcached does not have memory overhead for indexing and improves the efficiency of SSD accesses by directly utilizing keys to access KV pairs on the SSD.

6 Conclusion

We designed Mlcached to provide a multi-level cache hierarchy for various Web caching services. Mlcached removes dependencies between DRAM and NAND flash memories utilizing our new KV device. We observe that such completely decoupled multi-level KV cache can achieve up to 6X lower cost or 4X lower latency while meeting the same SLA or TCO constraint. Moreover, our prototype of Mlcached shows competitive performance in its latency exhibiting only 12.8% of overhead with 80% L1 cache hit compared to the average RTT of a DRAM-only Memcached at one-third of its TCO.

References

[1] ADAMIC, L. A., AND HUBERMAN, B. A. Zipfs law and the internet. *Glottometrics 3*, 1 (2002), 143–150.

[2] ANAND, A., MUTHUKRISHNAN, C., KAPPES, S., AKELLA, A., AND NATH, S. Cheap and large cams for high performance data-intensive networked systems. In *NSDI* (2010), vol. 10, pp. 29–29.

[3] 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.

[4] ATIKOGLU, B., XU, Y., FRACHTENBERG, E., JIANG, S., AND PALECZNY, M. Workload analysis of a large-scale key-value store. In *ACM SIGMETRICS Performance Evaluation Review* (2012), vol. 40, ACM, pp. 53–64.

[5] CHOI, I. S., YANG, W., AND KEE, Y.-S. Early experience with optimizing i/o performance using high-performance ssds for in-memory cluster computing. In *Big Data (Big Data), 2015 IEEE International Conference on* (2015), IEEE, pp. 1073–1083.

[6] CIDON, A., EISENMAN, A., ALIZADEH, M., AND KATTI, S. Dynacache: dynamic cloud caching. In *Proceedings of the 7th USENIX Conference on Hot Topics in Cloud Computing* (2015), USENIX Association, pp. 19–19.

[7] DEBNATH, B., SENGUPTA, S., AND LI, J. Flashstore: high throughput persistent key-value store. *Proceedings of the VLDB Endowment 3*, 1-2 (2010), 1414–1425.

[8] DEBNATH, B., SENGUPTA, S., AND LI, J. Skimpystash: Ram space skimpy key-value store on flash-based storage. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data* (2011), ACM, pp. 25–36.

[9] FITZPATRICK, B. Distributed caching with memcached. *Linux journal 2004*, 124 (2004), 5.

[10] GARTRELL, A., SRINIVASAN, M., ALGER, B., AND SUNDARARAJAN, K. Mcdipper: A key-value cache for flash storage, 2013.

[11] LIM, H., FAN, B., ANDERSEN, D. G., AND KAMINSKY, M. Silt: A memory-efficient, high-performance key-value store. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles* (2011), ACM, pp. 1–13.

[12] LIM, K., CHANG, J., MUDGE, T., RANGANATHAN, P., REINHARDT, S. K., AND WENISCH, T. F. Disaggregated memory for expansion and sharing in blade servers. In *ACM SIGARCH Computer Architecture News* (2009), vol. 37, ACM, pp. 267–278.

[13] MALLADI, K. T., LEE, B. C., NOTHAFT, F. A., KOZYRAKIS, C., PERIYATHAMBI, K., AND HOROWITZ, M. Towards energy-proportional datacenter memory with mobile dram. In *ACM SIGARCH Computer Architecture News* (2012), vol. 40, IEEE Computer Society, pp. 37–48.

[14] NATH, S., AND KANSAL, A. Flashdb: dynamic self-tuning database for nand flash. In *Proceedings of the 6th international conference on Information processing in sensor networks* (2007), ACM, pp. 410–419.

[15] SANFILIPPO, S., AND NOORDHUIS, P. Redis, 2009.

[16] SRINIVASAN, V., AND BULKOWSKI, B. Citrusleaf: A real-time nosql db which preserves acid. *Proceedings of the VLDB Endowment 4*, 12 (2011).

[17] TSIROGIANNIS, D., HARIZOPOULOS, S., AND SHAH, M. A. Analyzing the energy efficiency of a database server. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data* (2010), ACM, pp. 231–242.

[18] TYLOR, J. Disaggregated rack, January 2013.