CS265 Design Document

Mali Akmanalp, Sophie Hilgard, Andrew Ross

March 19, 2017

1 Project Description

The goal of the project is to explore optimal allocations of a given amount of memory to cache, memtable, and bloom filters for different workloads in an LSM-tree.
To explore this, we first consider models that show the cache/memtable/bloom filter tradeoffs. We additionally have implemented a cache hit/cache miss simulation in Python for different workloads that we believe demonstrates what behavior we can expect in a variety of situations.
In particular, we expect that for workloads with random accesses for read spikes, the cache will be largely unhelpful and the system will likely benefit from leaving most memory as memtable and bloom filters. We expect to see similar results for bloom filter allocation as in the MONKEY example in this case.
However, if some portions of the database are more likely to be read more frequently, it may make sense to enable caching or optimize the bloom filters for these items.
Success in this project would be being able to quantify the workloads under which specific memory allocations are preferable and showing an improvement in results from an adaptive memory system as compared to standard RocksDB benchmarks.

2 Modeling

2.1 Base Case

Assume n_t items in total DB
n_c items that fit in cache
n_m items that fit in memtable
R ratio between layers of LSM tree such that
$L1 = R * n_m$
$L2 = R^2 * n_m \ldots$

We can solve for j the total number of layers required to store all the data:

$$n_m * \frac{1 - R^j}{1 - R} = n_t$$

$$j = \frac{log(1 - n_t * \frac{1-R}{n_m})}{logR}$$

The average cost of a write remains the same as for the basic LSM tree case:

$$\log_R \frac{n_t}{n_m}$$

The average cost of a read, we consider probabilistically over all possible locations of the read item, assuming a random distribution of reads:
Probability that read is in memtable $= \frac{n_m}{n_t} = p(mt)$
Probability that read is in cache $= \frac{n_c}{n_t} = p(cache)$
Probability that read is in L1 but not in cache $= \frac{n_m * R - \frac{n_m * R}{n_m * \frac{1-(R^j-1)}{1-R}} * n_c}{n_t} = p(L1)$
Where the numerator is the number of items that are in the first layer

$$n_m * R$$

minus the proportion of items from that layer that are probabilistically in the cache already

$$\frac{n_m * R}{n_m * \frac{1-(R^j-1)}{1-R}} * n_c$$

Probability that read is in L_i but not in cache $= \frac{n_m * R^i - \frac{n_m * R^i}{n_m * \frac{1-(R^j-1)}{1-R}} * n_c}{n_t} = p(L_i)$
Where the numerator is the number of items that are in the first layer

$$n_m * R$$

minus the proportion of items from that layer that are probabilistically in the cache already

$$\frac{n_m * R}{n_m * \frac{1-(R^j-1)}{1-R}} * n_c$$

Where here the $R^j - 1$ comes from the fact that items already in memtable (L0) are not allowed to occupy the cache.

Expected cost of read $= p(mt) * 0 + p(cache) + 0 + \sum_{i=1}^{j} p(Li) * i$

2.2 Skewed Reads

Now consider the case for skewed reads, where we say d_{hf} (d_{lf}) percent of the data receives r_{hf} (r_{lf}) percent of the reads (where $d_{hf} + d_{lf} = 1$ and $r_{hf} + r_{lf} = 1$). On average, we can assume that the cache contains $r_{hf} * n_c$ items from $d_{hf} * n_t$ and $r_{lf} * n_c$ items from $d_{lf} * n$. Then the expected cost of a read is dependent on whether the data item being read is in $d_{hf} * n_t$ or $d_{lf} * n$ as the probability of a cache hit varies.
For data in $d_{hf} * n$,

Probability that read is in memtable $= \frac{n_m * d_{hf}}{d_{hf} * n_t} = p(mt)$
Probability that read is in cache $= \frac{r_{hf} * n_c}{d_{hf} * n_t} = p(cache_{hf})$
Probability that read is in L1 but not in cache $= \frac{n_m * R * d_{hf} - \frac{n_m * R}{n_m * \frac{1-(R^j-1)}{1-R}} * r_{hf} * n_c}{d_{hf} * n_t} = p(L1_{hf})$
Expected cost of read on item in d_{hf}: $E[C_{hf}] = p(mt) * 0 + p(cache_{hf}) + 0 + \sum_{i=1}^{j} p(Li_{hf}) * i$

Concretely, consider where we have 3 levels and 800 total items with a cache of size 10 and a ratio of 2 (for L0=100, L1 = 200, L2 = 400 items), with $d_{hf} = .2$ and $d_{lf} = .8$ and $r_{hf} = .8$ and $r_{lf} = .2$. Then the cache on average contains 8 items from $d_{hf} * n$ and 2 items from $d_{lf} * n$. If we execute a read on one of the 200 items in d_{hf}, then, there is a $\frac{8}{200}$ chance that that item is in the cache. If we execute a read on one of the $200 * \frac{1}{4} = 50$ items of $d_{hf} * n_t$ in L1, we expect that $\frac{2}{6} * 8$ of those items would have actually been found already in cache, as this level contains $\frac{2}{6}$ of all of the items not in the memtable. Then the probability that a read is found in L1 is the proportion of the $d_{hf} * n_t = 160$ items that will reside in L1 but not in the cache, which is $\frac{40-\frac{2}{6}*8}{160}$.

The expected cost of a read on an item in d_{lf} can be enumerated analogously, and we combine the expectation of reads in d_{hf} and d_{lf} as:
Expected cost of read $= r_{hf} * E[C_{hf}] + r_{lf} * E[C_{lf}]$

We can also define r_{lf} in terms of r_{hf} as $r_{hf} - 1$ and d_{lf} in terms of d_{hf} as $d_{hf} - 1$. (Doing this will make the effect that moving these parameters in one direction or the other has more obvious in the total overall formula.)

2.3 Bloom Filters

For a Bloom filter of k bits with h independent hash functions $h_1, h_2, ...h_h$, the probability that a given bit is still set to 0 after inserting n keys is

$$(1 - \frac{1}{k})^{n*h}$$

Then the probability of a false positive is

$$(1 - (1 - \frac{1}{k})^{n*h})^h \approx (1 - e^{-hn/k})^h$$

We can minimize this over h to find the optimal number of hash functions, which is $h = \ln(2) * \frac{k}{n}$. Assuming that this is the number of hash functions h we will use, the probability of a false positive as a function of the number of bits is then

$$(1 - e^{-\ln(2)*k/n*n/k})^{\ln(2)*\frac{k}{n}} = (\frac{1}{2})^{\ln(2)*\frac{k}{n}} \approx (.6185)^{\frac{k}{n}}$$

For an item in any any level L_i of the LSM tree with $i \geq 2$ we can reduce the expected cost of accessing that item from i by the number of Bloom filter negatives at any level $j < i$.

Then the expected cost of accessing an item at $L_i = \sum_{j=1}^{i-1} p(fp_j) * 1 + 1$ Where $p(fp_j)$ is the probability of a false positive for that key at level j and 1 is the cost of actually accessing the item at level i assuming fence pointers that lead us to the correct page.

2.4 Variable Cache Size, Constant Number of Layers

To analyze a variable cache/memtable allocation with a given memory size n_v, let n_l be the size of the first layer when $n_v = n_m$ and $n_c = n_v - n_m$.

In the Base Case, if we assume some fixed base layer size n_l, which is the size of the memtable if it exists.

$$p(mt) * 0 + p(cache) + 0 + \sum_{i=1}^{j} p(Li) * i$$

In the extreme case where $n_m = 0$ (no memtable), the formula in the numerator of the sum simplifies to be over n the total number of items, as there is no memtable layer and the probability of the first layer now has a cost of 1. However, we now have to add a number of items to each level of the tree that sum to the amount that were in L0. We add them as a geometric series per layer to maintain the structure

$$\sum_{i=i}^{j} \frac{(n_l) * R^i + \frac{(n_l)*R^i}{n} * n_v - \frac{(n_l)*R^i}{n} * n_v}{n_t} * (i)$$

$$\sum_{i=1}^{j} \frac{(n_l) * R^i}{n_t} * i$$

In the extreme case of no cache,

$$\sum_{i=1}^{j} \frac{(n_l) * R^i}{n_t} * i$$

In general, for any selection of n_m and n_c we have

$$\sum_{i=1}^{j} \frac{(n_l) * R^i - \frac{(n_l)*R^i}{(n_l)*\frac{1-(R^j-1)}{1-R}} * (n_v - n_m) + \frac{(n_l)*R^i}{(n_l)*\frac{1-(R^j-1)}{1-R}} * (n_v - n_m)}{n_t} * i$$

and so with a random workload, the choice is irrelevant assuming a constant number of layers. In actuality, with a larger memtable we will be able to decrease the total number of layers needed.

In the Skewed Read case, Now consider the case for skewed reads, where we say d_{hf} (d_{lf}) percent of the data receives r_{hf} (r_{lf}) percent of the reads (where $d_{hf} + d_{lf} = 1$ and $r_{hf} + r_{lf} = 1$). On average, we can assume that the cache contains $r_{hf} * n_c$ items from $d_{hf} * n_t$ and $r_{lf} * n_c$ items from $d_{lf} * n$.
For the extreme case of no cache, the result will be as before.
For the extreme case of no memtable, the expected cost of a read is dependent on whether the data item being read is in $d_{hf} * n_t$ or $d_{lf} * n$ as the probability of a cache hit varies.
For data in $d_{hf} * n$,

Probability that read is in cache = $\frac{r_{hf}*n_c}{d_{hf}*n_t} = p(cache_{hf})$
Probability that read is in L_i but not in cache =

$$\frac{n_l * R^i * d_{hf} - \frac{n_v*R^i}{n_l*\frac{1-(R^j-1)}{1-R}} * r_{hf} * (n_v - n_m) + \frac{n_v*R^i}{n_l*\frac{1-(R^j-1)}{1-R}} * d_{hf} * (n_v - n_m)}{d_{hf} * n_t} = p(Li_{hf})$$

Expected cost of read on item in d_{hf}: $E[C_{hf}] = p(cache_{hf}) + 0 + \sum_{i=1}^{j} p(Li_{hf}) * i$

2.5 Number of Layers Varies with Memtable Size