Problem Set 2 Solution

15-440/15-640 Distributed Systems Spring 2017

Assigned: Tuesday February 21, 2017

Due: 10:00 am on Tuesday February 28, 2017

Question 1 (15 points)

Impressed by your performance in 15-440/15-640 at CMU, DropBox hires you to replace their poorly-performing full-replica design with a new product that uses a genuine caching mechanism. This new product is targeted at a group of companies in the movie animation industry whose workloads are video reviewing (read-only) and video editing (read-write). Video files range from 50 MB to 150MB in size, with a mean of 100 MB. About 90% of the opens are for video reviewing, and about 10% are for video editing. A typical review session takes 30 minutes, while a typical editing session only takes 10 minutes. The companies that will use this system are globally distributed across the Internet, with a mean end-to-end bandwidth of 10 Mbps and a mean RTT of 100 ms. They all use Linux clients.

For the new DropBox product, you architect a write-through whole-file caching system. On a close after a video editing session, the entire contents of the just-edited file are transmitted to the server. You use read and write leases on whole files as the cache consistency mechanism.

A. Suppose you would like most sessions to complete with a single lease request. In other words, you would like to reduce the number of lease renewal requests. How long should read leases and write leases be in order to achieve this goal? State and justify any assumptions that you make.

B. Your DropBox team has implemented and deployed this system, using the lease periods stated in your answer to A. Now consider a situation where a new lease request arrives at the server when there are 5 other requests already waiting ahead of it in the FIFO queue for a lease. What is the worst case waiting time for this new request, before it receives its lease?

C. For B, what is the best case waiting time?

Solution:

Note: A key observation is that read leases are not exclusive, but write leases are exclusive (so only one owner of a copy is allowed).

A. Accepted answers:

- (We prefer this) 31.33 minutes for read lease (RTT + transmission delay + 30 minute session),

and 12.66 minutes for write lease (RTT + 2*(one-way transmission delay) + 10 minute session). **Note: if the student uses 150MB instead of 100MB, this will be 32min and 14min.**

- 30 minutes (a typical reading session length) for read lease, and 10 minutes for write lease, with **explicit** mention of the assumption that this time does NOT include the transmission time. A answer with only "30 min" (or 10) but with no reference to the transmission delay will be given only partial credits.

- Any other larger-than-30 minute read lease (e.g. infinite time), with **explicit** mention of the protocol details, such as the server can decide whether to revoke the read lease when a write requests come. Note that this is reasonable since 90% of the requests are reads.

B. The key part of this question is to recognize that write leases are **exclusive**, and are thus bottlenecks. The worst case is when the lease queue contain 5 leases all pending to operate on the same file as the new lease, with alternating fashion (W, R, W, R, W). The lease times should be consistent with part A.

   Note: Another valid answer should be a (W, W, W, W, W) queue. This is a bad case, but not the worst (unless the student mentions that the read leases can be revoked at any time). A partial credit is received for this.

C. A (R, R, R, R, R) queue. The server can send out everything together, yielding essentially no waiting time (i.e. waiting time = 0).

Question 2 (15 points)

An Internet service makes available non-sensitive, anonymous user data that can be used for data mining purposes by customers. This data is stored in a read-only in-memory database consisting of one million equal-sized blocks of 32KB each. The blocks map to the following non-overlapping sets: 500 descriptor blocks, 1,000 tag blocks, and 5,000 index blocks, with everything else being data blocks. Database code executing on clients can directly access these blocks over the Internet.

A client implements a cache of size 10,000 blocks. Accessing a block takes 5 ms on a cache hit, and 200 ms on a cache miss. In a typical database operation at the client, 5% of the accesses are uniformly distributed over the descriptor blocks, 5% of the accesses are uniformly distributed over the tag blocks, 10% of the accesses are uniformly distributed over the index blocks and the remaining 80% of the accesses are uniformly distributed over the data blocks.

For the following questions, you may need to make simplifying assumptions. Be sure to clearly state them, and explain why they are reasonable assumptions.

A. How would you best use the available cache space?
   (*Hint: You may want to combine static and dynamic policies.*)

B. The cache starts out completely cold. What is the point at which you would consider the cache fully warmed up?
(Hint: a precise qualitative answer is acceptable.)

C. Once the cache has been warmed up (as defined in your answer to B), what is the average time for a typical client access?

Solution:

A. Static allocation of blocks in cache to descriptor (500 blocks), tag (1000 blocks) and index (5000 blocks) blocks (6500 blocks in total). The remaining blocks shall be dynamically allocated to data blocks with any reasonable eviction policy.

B. The cache is considered warmed up when all of the cache blocks have been filled with data, according to the distribution stated in part A. This could be after 50,000 client operations (using typical database operation stats mentioned in the question), since there would be about 2,500 uniform accesses to descriptor and tag blocks, 5000 uniform accesses to index blocks and the remaining would be uniform accesses to data blocks.

Note that the important point here is filling up the cache with uniform accesses of tag, index and descriptor blocks - now we can assume that all of these blocks are present in the cache.

C. All descriptor, tag and index blocks are in the cache at this point (since the cache is warmed up), and 3500 out of 993,500 data blocks are in the cache. So, the average time of a typical client access (as stated in the question) is as follows:

0.05 * (1*5ms + 0*200ms) + 0.05 * (1*5ms + 0*200ms) + 0.1 * (1*5ms + 0*200ms) + 0.8 * ((3500/993,500)*5ms + (1 - 3500/993,500)*200ms)
= 0.25 + 0.25 + 0.5 + 159.45
= 160.45 ms

Question 3 (15 points)

The *Alternative Facts Journal (AFJ)* has become one of the nation's largest sources of news and current information. On a typical day, news stories (entirely new, or updated versions of existing stories) come out at an average rate of once every 15 minutes. AFJ's popularity, nationwide distribution span, and (mostly) static content make it a perfect use case for a CDN. Your CDN startup is trying to get AFJ's business.

A. You are trying to convince AFJ management of the value of using a CDN. What two advantages of using a CDN (from AFJ's viewpoint) would you stress? A skeptic of CDNs on the AFJ management wants to derail your efforts. What is one potential shortcoming of a CDN that he could point out?

B. Once you succeed in winning AFJ's business, you have to design a cache consistency mechanism for their expected workload. You have 40 CDN sites, each of which receives an average number of 300 requests per second to fetch some news article. You first consider using check-on-use as the cache consistency mechanism — in other words, before handing out a cache copy of an article to a user, the CDN site checks with AFJ to make sure that the cache copy is up to date. Suppose it costs the AFJ IT infrastructure 0.01 cent to handle such a check-on-use request. What is the minimum annual IT budget of AFJ?

C. Terrified by the answer to B, AFJ's management begs you to reduce their IT expense. You therefore decide to use a callback-based mechanism in which a callback break message from AFJ pushes the bits of new articles (or the new version of an existing article). You negotiate a payment by AFJ of one cent per interaction with a CDN site. What is the new minimum annual IT budget of AFJ?

Solution:

A. A few advantages (full credits for other valid reasons as well):
   1. AFJ can reduce load on their servers, allowing AFJ to serve more articles more broadly.
   2. Caching across the US can improve load-times for location-specific articles.
   3. Location-based adverts can be inserted into AFJ's static content to increase ad revenue.
   4. … (other answers possible)

B. Disadvantages (full credits for other valid reasons as well)
   1. Depending on cache semantics, customers may not get articles immediately (cache propagation issues).
   2. Updates to articles may be problematic, as one area of the US may get updates sooner than others.
   3. … (other answers possible)

C. $40\ sites * 31536000 \frac{sec}{year} * 0.00001 \frac{\$}{req} = 37,843,200 \frac{\$}{year}$

D. $40\ sites * (60 / 15) \frac{req}{hour} * 8760 \frac{hour}{year} * 0.01 \frac{\$}{req} = 14016 \frac{\$}{year}$

Question 4 (15 points)

A server stores key-value pairs, where the value is an integer and the key is a unique identifier provided by the client which supplies that value. There are two RPC operations:

- `write(key k, int v)` changes the data object identified by key k to have new value v. If key k doesn't already exist, it creates a new object with value v.
- `read(key k)` returns the value of the object identified by key k if it exits.

Each client can go through one of 3 caching proxies: A, B, and C. For example, calling `B.write(K1, 4)` means write via proxy B. Assume each proxy has a LRU cache that is able to fit exactly 3 entries. For the following time sequence of operations, answer each of the questions below. Show your working.

Time:	***Operation***
01:	`A.write(K1, 1)`
02:	`B.write(K2, 1)`
03:	`C.read(K1)`
04:	`B.write(K1, 0)`
05:	`A.write(K1, 1)`
06:	`A.write(K2, 4)`
07:	`A.write(K3, 2)`
08:	`C.write(K3, 7)`
09:	`B.read(K1)`
10:	`C.write(K4, 10)`
11:	`B.read(K3)`
12:	`C.write(K2, 0)`
13:	`A.read(K4)`
14:	`A.write(K5, 2)`
15:	`C.read(K2)`

A. Suppose a callback-with-new-value mechanism for cache consistency is used (i.e., invalidation plus new value is provided by a callback break). What does the cache of A, B and C look like after t=4? What about after t=12?

B. Instead of callback, suppose the cache uses "check on use"? What will the contents of A, B, and C's cache be after t=4? What about after t=14?

C. Suppose faith-based caching is used, with cache entries being assumed valid for 10 time units without checking. When is the earliest that one-copy semantics is violated in the above operation sequence?

Solution:

Note: While we do prefer answers that show the ordering within a cache (e.g. MRU to LRU), out-of-order answers are acceptable.

A. After t=4,
- A's cache will be (K1, 0)
- B's cache will be (K1, 0), (K2, 1)
- C's cache will be (K1, 0)

After t=12,

- A's cache will be (K3, 7), (K2, 0), (K1, 1)
- B's cache will be (K3, 7), (K1, 1), (K2, 0)
- C's cache will be (K2, 0), (K4, 10), (K3, 7)

B. After t=4,

- A's cache will be (K1, 1)
- B's cache will be (K1, 0), (K2, 1)
- C's cache will be (K1, 1)

After t=14,

- A's cache will be (K5, 2), (K4, 10), (K3, 2)
- B's cache will be (K3, 7), (K1, 1), (K2, 1)
- C's cache will be (K2, 0), (K4, 10), (K3, 7)

C. At time 4, when B writes to K1, the cached copies in A and C are both invalid.

Question 5 (12 points)

Consider a Linux uniprocessor system that is managing an I/O buffer cache of 100 4KB blocks using the ARC (Adaptive Replacement Cache) algorithm. The total storage size is 4 GB, so there are 2^{20} total blocks (roughly one million) in the system. Answer the questions below by drawing the state of the ARC cache, using the notation from class. We expect to see lists T_1, B_1, T_2 and B_2 clearly identified in your illustrations, and their contents labeled if known.

(Hint: using the notation from class, the quantity c is 100 here.)

A. Initially, there is just a single process P_1 in the system. It executes in a tight loop whose working set is just 8 blocks. For simplicity, you can assume that these are blocks 0, 1, 2, … 7. Show the state of the ARC cache after one million iterations of the loop.

B. Suppose a second process P_2 is launched at the same time as P_1. Starting at block 100, this process sequentially scans all the blocks up to 100,000. The interleaving pattern of P_1 and P_2 will clearly influence the cache contents. Suppose Murphy influences the scheduling of P_1 and P_2 to make the cache state as bad as possible for P_1. Under these conditions, draw the state of the ARC cache after P_1 has completed 10 iterations.

C. For question B, suppose P_2 was launched after P_1 completes one million iterations. P_2's access pattern is the same as before, and Murphy tries just as hard to make the cache state as bad as possible for P_1. Under these conditions, draw the state of the ARC cache after P_1 has completed 10 iterations beyond the launch of P_2.

Solution:

Note: Assume B1=T1=B2=T2=0 initially. It is also valid to assume they start with capacity 50 (or other initial values), if such assumption is stated clearly. If the initial capacities are 50, then the student should be clear about how things get moved among T1, B1, T2 and B2.

A. T1: Empty
B1: Empty

T2: TOP[7, 6, 5, 4, 3, 2, 1, 0]BOTTOM
B2: Empty

T1 and B1 are empty because all data blocks accessed by P_1 were touched at least twice (in fact, exactly 125,000 times). B2 is empty because no more than 8 data blocks was used, which fit well in the T2.

B. Since P_2 accesses each block from #100 to #100,000 only once, the only way for the sabotage is to prevent P_1's blocks from getting into L2 (i.e. T2 and B2). A necessary and sufficient way to achieve this is then to keep filling the L1 list with the blocks from P_2, and access P_1's block only if this block has been evicted from both T1 and B1. For example, here is a sample procedure that would work (there are also other possible solutions):

1. Let P_2 access 100 blocks;
2. Let P_1 access 1 block;
3. Repeat step 1.

Before P2 finishes, clearly, none of the P_1 accesses were hits. At the end of the 10th iteration of P_1, P_2 has finished accessing 8000 blocks, so the ARC cache state is (stream access fill the LRU T1 cache):

T1: TOP[7, 8000, 7999, 7998, …, _, _, 7902]BOTTOM Size: 100
B1: Empty

T2: Empty
B2: Empty

Note: The answer may differ for different scheduling. But the key points are always: (1) run P_2 for long enough so that the block to be accessed by P_1 is in neither T1 or B1 (for example, step 2 of the procedure above can be "let P_1 access 8 blocks"); and (2) L2 is empty.

C. Note that the program continues from the state outlined in part A. At this time, all the blocks needed by P1 are in L2, so there is no way for Murphy to sabotage the cache accesses. One possible answer is:

T1: TOP[105, 104, 103, 102, 101, 100]BOTTOM
B1: Empty

T2: TOP[7, 6, 5, 4, 3, 2, 1, 0]BOTTOM
B2: Empty

Note: The answer may different for different scheduling. However, the states of T2 and B2 are always the same (the one above). T1 and B1 can be in any consistent state with all data entries are from P_2's blocks.

Question 6 (20 points)

Consider a LRU page cache that receives the following reference stream (the page number of each reference is shown):

27, 12, 15, 15, 15, 27, 34, 15, 12, 27, 34, 15, 15, 15, 12, 8

Suppose the cache size is 3 pages, and the cache is initially empty.

A. How many misses does the cache experience for this reference stream? Show your working.

B. Which are the pages left in the cache at the end of this reference stream?

C. If an optimal replacement policy (OPT) is used instead of LRU, how many misses will there be and what will be the cache contents at the end? Explain your answer.

D. Give a recurring reference stream for which LRU is suboptimal with a cache size of 3 pages, but is optimal with a cache size of 4 pages.

Solution:

A. m: miss, h: hit, e:eviction. Then the sequence will be:

27(m), 12(m), 15(m), 15(h), 15(h), 27(h), 34(m, e-12), 15(h), 12(m, e-27), 27(m, e-34), 34(m, e-15), 15(m, e-12), 15(h), 15(h), 12(m, e-27), 8(m, e-34)

So there are 10 misses.

B. 15, 12, 8

C. nc: not cached

27(m), 12(m-nc), 15(m), 15(h), 15(h), 27(h), 34(m), 15(h), 12(h), 27(h), 34(m), 15(h), 15(h), 15(h), 12(h), 8(m-nc)

6 misses. The final cache content can be [27, 15, 12] or [15, 12, 8]. While the first answer is based on the fact that no further accesses of 12 or 8 are observed, the student can assume any cache policy at the end since it doesn't affect the cache performance anyway.

D. 1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4…

You will find that every reference will be a cache miss in this stream for a cache size of 3 pages, but for a cache size of 4 pages, a warmed up cache will have no misses.

Question 7 (8 points)

Consider a file server that implements lease-based cache consistency. When the server becomes heavily loaded, would you suggest decreasing, increasing, or keeping intact the length of new leases it hands out to clients? Explain your answer and discuss the effects of this strategy on the server and clients.

Solution:

It depends on the kind of request load on the server.

In case of read-heavy workload, i.e. most of the requests are for read, lease time should be increased. Then, client won't need to renew the lease for a longer period of time. As a result, there would be fewer incoming requests on the server, which would help in reducing the load on the server.

In case of write-heavy workload, i.e. most of the requests are for write, lease time should be decreased. As there can be at most one granted write lease at a time (writer needs exclusive access), and if there are lot of write lease requests, a long lease time would increase the queue size on the server and decrease the throughput of the system. Therefore, lease time should be shorter.

Note: Points will be given even if only one case is mentioned, with proper justification. Other reasonable assumptions with reasonable justifications (e.g. decrease queue delay) may also receive partial credits.