TDT4225 - Exercise 3

Håkon Ødegård Løvdal

October 2015

Task 1

The adaptive replacement cache (ARC) in ZFS is designed to solve the problem where a sequential file read fills the cache, and thereby removing files useful to other processes.

The main idea with ARC is to split the cache of size n into two separate lists, t_1 and t_2. This two list represent the most recently used and most frequently used block respectively. In addition to this, it also implements two lists, b_1 and b_2 (also referred to as ghost lists). All these lists are ordered in a LRU manner. The length of $t_1 + b_1$ (and $t_2 + b_2$) are equal to n. For the explanation I will focus on t_1 and b_1 (one side) but the explanation also applies to the other side. If list t_1 becomes full and a block must be referenced, the least recently used block will be removed from the list. Instead of forgetting this block, the block is referenced in b_1 (only a reference, not the actual block). This is the point where the adaptive part of the algorithm appears. If a block in b_1 is referenced, this yields that t_1 should increase its space. So, therefore the maximum size of t_1 is increased with one, and t_2 is decreased by one. This can continue until t_1's size is equal to n.

This implantation makes ZFS ARC support multiple access patterns. If the access is a sequential file read, or an access from multiple processes/threads, the algorithm adjusts the cache depending on what is most likely to happen next.

Task 2

The use of Bloom filters is appropriate under conditions where the key has a certain probability of not existing (early elimination of candidates). Typical conditions (from the compendium by Bratbergsengen):

- Validity check of credit cards
    - By having a Bloom filter of all the invalid card numbers, it will be possible to determine if a card is valid for more than 99% of the cards by only checking the filter.
- Reduce search for records in overflow storage

  - The use of a Bloom filter could reduce the search in a overflow storage. By creating a filter, one could check this before checking the overflow area.
- Duplicate check
  - By checking records to be inserted against the filter, one can easily determine if the record is a duplicate

It is important to keep in mind that a filter does not guarantee the existence of a key, but it can guarantee a non-existence. This means it generates false-positives, but never false-negatives.

Task 3

For an illustration of the following description, see figure 2.4 on page 15 in Christian Forfang: Evaluation of high performance Key-Values Stores.

Put:

The Put operation in LevelDB adds a new entry to the database. Instead of changing or removing from existing entries, LevelDB adds entries in the order they occur. First, when the command is issued, the command is written to the log (on disk/non-volatile memory). This is done to ensure that the database can recover in the event of a system crash. Thereafter, the entry is added to the MemTable with an incremented sequence number (LevelDB always increments, to distinguish between two entries with the same key). The MemTable is located in the memory (volatile). When the MemTable is full, a new MemTable is created, and the previous is set to be immutable. Then a compaction event is triggered, which converts the MemTable to a level-0 SSTable. This level SSTables are also compacted when they reach their size limits.

By doing this the consistency of the database is ensured. The log file for the MemTable in question is not updated until the MemTable is completely written to disk. The MemTable is not freed from memory before the complete disk write either.

Get:

The Get operation in LevelDB returns an entry from the database. First the MemTable in memory is checked. This is the newest location an entry can appear, and therefore it is checked first. Thereafter the immutable MemTable is checked if it exists. If no entry is found, LevelDB will continue search the levels on disk in turn. Due to the incremented sequence number it is guaranteed that the found entry is the newest when searching through the database storage in this order. It is also worth mentioning that SSTables are read-only, and therefore can be accesed with multiple threads in a thread-safe way. It does not need any mutex locks for reading.

Task 4

Get:

The LMDB uses an alternative version of B+-trees, called Copy-on-Write B+-trees. When the database is to write, a read-write transaction is initiated (even though it only writes). The Copy-on-Write B+-tree creates a copy of the previous tree when it is updates. That is, it does not create a complete copy of the tree, but it copies and modifies the root and all affected leaf nodes. The un-modified leafs are referenced to the new tree.

When all changed are applied and the transaction is committed, all changed pages are write to disk, and all pages no longer used are freed from memory.

Put:

Like the Put operation, the Get operation issues a read-write transaction. The read-transaction uses the current version of the database for lookups. These lookups are standard B+-tree lookups. During the read, the transaction is registered as an active reader.

Task 5

$$\begin{aligned} T_{HDD} &= T_R + (T_B \times T_R) \\ &= T_R \times (1 + \frac{2KB}{800KB}) \end{aligned} \tag{1}$$

$$\begin{aligned} T_{SSD} &= T_A + T_B \\ &= 0.1ms + \frac{2KB}{250MB/s} \\ &= 0.108ms \end{aligned} \tag{2}$$

$$\begin{aligned} T_{SSD} &= T_{HDD} \\ 0.108ms &= T_R \times (1 + \frac{2KB}{800KB}) \\ 0.108ms &= T_R \times (1.0025) \\ \frac{0.108ms}{1.0025} &= T_R \\ 0.107731ms &= T_R \end{aligned} \tag{3}$$

Given T_R it is easy to calculate that the rotation speed for the HDD must be $0.107731 \times 1000 * 60 = 6463.85 \approx 6464$ RPM

Task 6

$n = 2 \times 10^8$

$l = 100$ byte

$V = nl = 2 \times 10^8 \times 100 = 2 \times 10^{10}$

$b_{SSD} = 4$ KB block size

$t_{SSD} = 0.1ms = 10^{-4}$

$M = 2 \times 10^8 = 200$ MB work space

$m = \frac{M}{l} = \frac{2\times10^8}{100} = 2 \times 10^6$

$h_i = \lceil \log_2 m \rceil = \lceil \log_2 2 \times 10^6 \rceil = 21$ height initial sort

$r = 1$ microsec $= 10^{-6}$

$N = \frac{V}{2M} = \frac{2\times10^{10}}{2\times2\times10^9} = 50$

$h_m = \lceil \log_2 N \rceil = \lceil \log_2 50 \rceil = 6$

$T_{CPU,i} = n(h_i + 1)r = 2 \times 10^8 \times (21 + 1) \times 10^{-6} = 4400$ seconds (answer to subtask A)

$T_{CPU,m} = nh_m r = 2 \times 10^8 \times 6 \times 10^{-6} = 1200$ seconds

$T_{SSD,io} = \frac{4Vt_{SSD}}{b_{SSD}} = \frac{4\times2\times10^{10}\times10^{-4}}{4} = 2 \times 10^6 = 2000$ seconds

This gives the following sort time: max(4400, 1000) + max(1200, 1000) = 5600 seconds (answer to subtask B, see the list for subtask A)

Task 7

Subtask a

Records to sort in reservoir of size 4: $32, 44, 10, 3, 5, 79, 64, 43, 98, 33, 8, 3, 5, 2$
Always choosing the minimum key that fit into a run. When every key in the reservoir is less than the last key written, I create a new run. The average size of a run ("delfil") is 2M $= 2 * 4 = 8$, where M is the reservoir size.

Run 1 ("delfil"): $3, 5, 10, 32, 43, 44, 64, 79, 98$
Run 2 ("delfil"): $3, 5, 8, 33$
Run 3:("delfil"): 2

Subtask b

$$
\begin{aligned}
y &= \lceil \frac{N-1}{p-1} \rceil \\
&= \lceil \frac{370-1}{90-1} \rceil \\
&= 5
\end{aligned} \tag{4}
$$

$$
\begin{aligned}
x &= y(p-1)+1-N \\
&= 5(90-1)+1-370 \\
&= 76
\end{aligned} \tag{5}
$$

I need 76 "dummy-files" in the first merge, to get an optimal merge tree.

Subtask c

$N = 370, p = 90, y = 5, x = 76$

$$
\begin{aligned}
\text{Total I/O Volume} &= 2(p-x)+2P+2P+2P+2N \\
&= 2(14)+6*90+2*370 \\
&= 1308 \text{ delfiler}
\end{aligned} \tag{6}
$$

Task 8

	R: 250 000	A: 300 000	B: 1 200 000
Key Length	8	8	8
Record Length	200	600	300
Volume in MB	50	180	360
Bytes from operand		100	100
Nettovolume in MB		30	120

Table 1: Record details

Nested Loop

$$
\begin{aligned}
n &= \lceil \frac{V_A}{M} \rceil \\
&= \lceil \frac{30}{10} \rceil \\
&= 3
\end{aligned} \tag{7}
$$

$$\begin{aligned} V_{nl,J} &= V_A + nV_b + V_R \\ &= 180 + 3 \times 360 + 50 \\ &= 1310 \text{ MB} \end{aligned} \tag{8}$$

Nested Loop with strip file

Read A 180 MB

Read B 360 MB

Write Strip File B' 120 MB

Read Strip File B' 2 * 120 MB

Write R 50 MB

Total 950 MB

Partitioning

Read A 180 MB

Write A' Strip 30 MB

Read B 360 MB

Write B' Strip 120 MB

A' + B' 150 MB

Write R 50 MB

Total 890 MB

Task 9

- $9 * 10^6$ people
- 10^5 different names
- Rows are 150 bytes
- Projection row is 30 bytes
- WS = 50 MB

$$V_A = 150 \times 9 \times 10^6 = 1350 \text{ MB}$$

$$V_R = 10^5 \times 30 = 3 \text{ MB}$$

$$V_{nl,p} = \frac{1350(1350\times 50)}{2\times 50} + 3 = 18903 \text{ MB without Guards}$$

$$n = \lceil \frac{V_A k}{M\alpha} \rceil = \lceil \frac{1350\times 30}{50\times 150} \rceil = 6$$

$$V_{nlw,p} = 1350(1 + \frac{6-1\times 30}{150}) + 3 = 1355 \text{ MB with Guards}$$