Assignment 1 - TDT4225

Olaf Rosendahl

September 02, 2022

1. SSD: How does the Flash Translation Level (FTL) work in SSDs?

   FTL implements wear leveling. That means that writes are distributed over the entire disk to avoid that the same set of blocks repeatedly get erase-write cycles. FLT also runs garbage collection where blocks with a lot of its pages are invalid to make the page available for later writes.

2. Why are sequential writes important for performance on SSDs?

   Random writes can generate internal fragmentation on SSDs, which leads to garbage collection when the disk starts to be filled up. Random writes basically gives the FTL a lot of work, while sequential writes effectively invalidates flash blocks block by block during writes. No garbage collection is then needed and the write amplification is near 1. One exception is when the write size is equal to or larger then the clustered page size, in these cases random writes can be equally efficient in terms of performance. This is because there is less fragmentation when pages are filled. But since it's difficult to ensure that random writes are large enough, sequential writes are much easier to use when wanting good performance on SSDs.

3. SSD: Discuss the effect of alignment of blocks to SSD pages. See e.g. Figure 2.5 of Dybvik.

   "Write requests that are aligned to clustered pages can be written to disk with no further write overhead. In contrast, write requests that are not aligned generate overhead because the SSD controller needs to read the rest of the content in the last clustered page and combine it with the updated data before all the data can be written to a new clustered page. Such read-modify-write operations increase write latency" (Dybvik, 2017)

   This means that in order to get the best performance in a SSD, one should try to make write request that are a multiple of the clustered page size in the SSD.

4. RocksDB: Describe the layout of MemTable and SSTable of RocksDB.

   The SSTable is block-based. Each block starts with key-value pairs that are partitioned across data blocks in the beginning of the file.

Then there are meta-blocks with Bloom filters, compression, statistics and some other metadata. The a metaindex block has the index of all the meta blocks, while the index-block has a the index of all data blocks. The index block is structured in order to efficiently being able to location data blocks.

The Memtable is in-memory and all updates are first inserted here. It's a skip list whit all entries stored in sorted order. This allows the memtable to be search using binary search, and it's efficient when flushed the records to the SSTable.

5. RocksDB: What happens during compaction in RocksDB?

   During a compaction in RocksDB two or more SSTables are merged into one immutable SSTable. If a key is present in multiple tables, the newest one is used. RocksDB offers both leveled, universal and FIFO compaction:

   - Leveled compaction stores SSTables in sorted runs which contains muliple SSTables that doesn't overlap in key range with a total ordering. The different sorted runs are stored on different levels, with the oldest data being in the last level. In the first level, keys may overlap, meaning that a search in L0 costs significantly more than in L4 and we do also want as few SSTables in L0 as possible. The different levels in then compactioned from Li -¿ Li+1.
   - Universal compaction does in comparison to leveled compaction not use the key range when preventing overlaps, but time range. The output is another sorted run whose time range ranges from the earliest entry in the first input SSTable to the latest entry in the last input SSTable. There are far more levels here than in leveled compaction.
   - FIFO compaction is by far the more easy compaction and functions like a time-to-live cache. Everything is stored in L0, but only kept for a certain amount of time. When the database size reaches it maximum size, the oldest SSTable is deleted.

6. LSM-trees vs B+-trees. Give some reasons for why LSM-trees are regarded as more efficient than B+-trees for large volumes of inserts.

   LSM-trees has a lower write amplification, the number of disk-writes per database-write, which can be a bottleneck for B+-trees.

   LSM-trees also write compact SSTable-files sequentially, instead of having to overwrite multiple pages like in B+-trees. This leads to a better use of memory because of how larger writes are better for compaction.

7. Regarding fault tolerance, give a description of what hardware, software and human errors may be?

   - Hardware errors could be anything from power outages, faulty RAM, the network is unplugged and fires in the data center. Hardware errors are usually independent, one faulty RAM doesn't imply that all other RAM's are faulty.

- Software errors may be bugs that causes crashes when an application recieves bad input, a process that uses up some or all shared resources like CPU, memory or disk, a dependent system becomes unresponsive or failures where one fail leads to another fail. The mentioned errors are dependent errors which means that one error imply that it may happen all other places where the same code is present.
- Human errors happen because humans are known to be unreliable. Configuration errors are for example the leading outage-cause.

8. Give an overview of tasks/techniques you may take/do to achieve fault tolerance.

    - On way to add fault tolerance for hardware errors is to add redundancy in such a way that failure doesn't directly lead to errors since components for example may have two power supplies. In addition, software fault-tolerance techniques may be used to ensure that for example virtual machines continue to be available even though the actual hardware fails.
    - There is not easy solution to software errors, but some things can help, such as testing, isolation of processes, monitoring and carefully implementing the system design.
    - To avoid human errors, one solution is to design systems in a way that minimizes error opportunities, making it easy to the right thing. Systems that doesn't fail when used wrong, but instead tells the user what to do also avoids human errors. A solution when writing code is to use automated testing which moves the responsibility of running tests away from humans and to systems.

9. Compare SQL and the document model. Give advantages and disadvantages of each approach. Give an example which shows the problem with many-to-many relationships in the document model, e.g., how would you model that a paper has many sections and words, and additionally it has many authors, and that each author with name and address has written many papers?

    SQL has the obvious advantage of relations. Relations make it easy to query and retrieve the exact data wanted quickly and together. Documents on the other hand doesn't have relations, and developers do therefore have to retrieve relations by themselves using stored ids. Since one only store ids, there is also no assurance that relations still exists, while SQL ensure this through foreign keys.

    A advantage of documents is locality. When quering a document, everything is usually in the same document which means that retrievals are fast. The disadvatega with this is that relations to other documents leads to messy and multiple sequential queries, or duplication of data.

    SQL has a defined structure of each table while documents doesn't have a structure. Having a structure can both be an advantage and a disadvantage depending on the application.

As said, relations with other documents quickly becomes messy. In order to solve the given example with documents, there is multiple solutions. One is to store an array of the author-ids in each paper. That makes it relatively easy to find all authors of a paper, but in order to find all papers of an author you would have too query all papers where its array of authors contains the given author. This can easily create performance-issues.

A different technique is to duplicate data. Here you keep an array of author-ids in each paper, but also an array of paper-ids in each author. This means that each time a paper is added, the related author-documents also have to be updated with the added paper-id in the authors array of paper-ids. This leads to duplication of information, but also easier and more performant data-retrieval.

10. When should you use a graph model instead of a document model? Explain why. Give an example of a typical problem that would benefit from using a graph model.

    A graph modal is really beneficial when wanting to represent many relations between different entities. While relational databases handles many-to-many relations fine, a graph-model simplifies the structure when the connections get more complex. Since a graph consists of vertices and edges, there is really easy to add new edges between the given vertices whenever wanting to. A obvious example a problem that benefits from a graph is social graphs. There vertices are people and edges people who know each other. It's really easy to add new friendships, and also to find all friends of a given person. There's also easy to navigate to friends of friends in order to for example find out how many friendships there is between you and the prime minister.

11. Column compression: You have the following values for a column: 43 43 43 87 87 63 63 32 33 33 33 33 89 89 89 33

    a) Create a bitmap for the values.

	43	43	87	87	63	63	32	33	33	33	33	89	89	89	33
32	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
33	0	0	0	0	0	0	0	1	1	1	1	0	0	0	1
43	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
63	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0
87	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0
89	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0

    b) Create a runlength encoding for the values

    - **32**: 6, 1 (6 zeros, 1 one, rest zeros)
    - **33**: 7, 4, 3, 1 (7 zeros, 4 ones, 3 zeros, 1 one, rest zeros)
    - **43**: 0, 2 (0 zeros, 2 ones, rest zeros)
    - **63**: 4, 2 (4 zeros, 2 ones, rest zeros)
    - **87**: 2, 2 (2 zeros, 2 ones, rest zeros)
    - **89**: 11, 3 (11 zeros, 3 ones, rest zeros)

12. We have different binary formats / techniques for sending data across the network:

    - MessagePack
    - Apache Thrift
    - Protocol Buffers
    - Avro

    In case we need to do schema evolution, e.g., we add a new attribute to a Person structure: Labour union, which is a String. How is this supported by the different systems? How is forward and backward compatibility supported?

    - *MessagePack* encodes JSON. One don't need to create data structures since there are no structure-checks. Anything can be serialized as long as it has type tags. Since there is no schema, there may be problems when adding new attributes if the systems doesn't handle the possibility of too few or too many attributes themselves.
    - *Apache Thrift* is a encoding format that require a schema. Thrift handles forward-compability by requiring that new attributes are declared as optional. And only optional fields are allowed to be deleted. Backward-compability is handled by giving each field a unique tag, new code will therefore always be able to read old data, since the tag numbers is the same.
    - *Protocol Buffers* is pretty equal to Thrift, except array is repeated types and not a list like in Thrift.
    - *Avro* doesn't use tag numbers, but it uses a schema. In order to add new fields, a default value must be provided. Only fields with a default value can be deleted. This ensures that both the writer's schema and the reader's schema work together even if different. If the reader misses a field, the default value is used, and if the reader recieves an extra field it is ignored. This ensures both backward and forward compability.