TDT4225 Assignment 1

Simen Refsland

Task 1.

A flash transition layer is a firmware layer that comes with SSDs, where wear leveling is implemented. It also handles logical-to-physical memory address mapping and garbage collection. Wear leveling is essentially distributing writes over all the blocks to avoid erase-write cycles on the same set of blocks (blocks have a limited number of erase-write cycles before going bad). The three categories of wear leveling are: **no wear leveling** , which maintains a fixed mapping from logical to physical addresses on the SSD, which causes blocks with dynamic data to wear out quicker than blocks with static data (more erase-write cycles on dynamic), **dynamic wear leveling**, which maintains a dynamical mapping between logical and physical addresses, when a block is about to be modified, this block is marked as invalid and moved to a new location (this improves the situation with dynamic blocks earlier described) and **static wear leveling**, which is the same as dynamic wear leveling, but also relocates static blocks once in a while.

Task 2.

Sequential writes are important for many reasons, one of them are that they create less internal fragmentation, making GC more efficient, as well as utilizing parallelism and pipelining. By writing sequentially in the order of logical block addresses, GC is more efficient as flash blocks are invalidated block by block as writes proceed, leading to lower write-amplification. The size of the write requests matter however, random rewrites can be justified if the write requests are larger than or equal to the clustered block size.

Task 3.

The alignment of write requests size to clustered page size, meaning that a write request is a multiple of the clustered page size, is ideal because they can be written to disk without further write operations. If they are not aligned, then rest of the last clustered page's content must be read and combined with the updated data before all the data can be written to a new clustered page. This increases latency.

Task 4.

SSTable Layout

In the beginning of the file, sorted key-value pairs are partitioned across data blocks, then comes meta blocks, which hold metadata such as compression, statistics, Bloom filters etc., the metaindex blocks holds an index for every meta block, the index block conversely holds an index for every data block.

Memtable Layout

The memtable refers to the in-memory component C_0 from the original LSM-tree. All updates are first written to this table, until it reaches a maximum size, when the RocksDB creates a new one, marks the old one as immutable and it is written to the disk immideately or at a later time as a SSTable.

Task 5.

Compaction is similar to the rolling merge process done in LSM-trees, which takes at least two SSTables and merges them together so that a total ordering is made as defined by a comparator and writes out a new

immutable SSTable, where their merged entries are located. The compaction can in RocksDB be done in three styles: leveled, universal and FIFO.

Task 6.

LSM-trees are regarded as more efficient than B^+-trees for large volumes of inserts because the data is first placed in an in-memory component, which are inserted in-place, making fast write operations possible. The updates to on-disk components are done out of place to ensure sequential writes, and batching them together in memory enables the LSM-tree to avoid small random writes (reduce I/O operations) and instead do larger sequential writes, as well as perform write requests that align with the clustered page sizes, with the effect of having greater performance than B^+-trees.

Task 7.

Regarding fault tolerance, a number of errors, be it hardware-, software- or human-related, can occur. Hardware faults may be caused by power grid blackouts, faulty/crashing hard disks, faulty RAM etc. Software errors may be software bugs that causes every instance to crash (systematic errors), runaway processes taking shared resources, cascading failures (snowballing effect where a small fault in one area leads to faults another component). Human errors may be configuration mistakes, accidental removal of critical files, incorrect commands etc.

Task 8.

To achieve fault tolerance you must think in terms of hardware, software and human errors.

For hardware errors, the natural response is to add redundancy, such as RAID setups for disks, dual power supplies, backup power in the data center etc. To mitigate for increasing computing and data volumes demands, there has been a move towards systems that can tolerate entire machines going down, by using software fault-tolerance techniques in addition to hardware redundancy.

To mitigate software errors, it is not as straight forward, but may include testing, thinking about assumptions in the system, process isolation, monitoring and analyzing system behavior in the production environment.

Human errors can be mitigated by minimizing the opportunity for error, by designing intuitive and simple APIs and admin interfaces. Fully featured non-production sandbox environments may also be used for exploration and experimentation. Fast rollbacks for config changes, rolling out new code gradually may help for easy recovery. Detailed and clear telemetry of the system makes for easier monitoring and overview.

Task 9.

SQL

Advantages

- Support for joins
- Handles many-to-one and many-to-many relationships

Disadvantages

- May lead to complicated schemas when dividing document into many tables in normalization
- Schema changes may require downtime (ALTER for example)

Document model

Advantages

- Schema flexibility (schema-on-read)
- Higher performance (locality)
- Handles one-to-many relationships well (tree-structure)

Disadvantages

- Poor support for joins
- If data is denormalized, there may be trouble in keeping the data consistent
- Becomes less appealing for many-to-many relationships

When modeling a paper, the paper, sections and words can be modeled into a single document, because it has a hierarchical tree-structure. The problem arises when adding authors, which are connected to other papers as well, creating many-to-many relationships. If you try to add authors to each paper, then the author information would be duplicated for papers with the same author, leading to possible inconsistencies. Querying an author papers would also become more complex. In SQL, an intermediary table between papers and authors could be used.

Task 10.

Whenever the data is highly connected with several many-to-many relationships, it is advisable to consider graph model. It is natural for graph models to capture relationships between entities as well as querying them. Social networks is an example where many-to-many relationships and complex interconnection are prevalent, where graph models excel.

Task 11.

Textual encodings may be used if human readability and easy debugging are desired and space concerns are not an issue.

Task 12.

a.

43 43 43 87 87 63 63 32 33 33 33 33 89 89 89 33
Bitmap for each possible value:
32: 0000000100000000

33: 0000000011110001
43: 1110000000000000
63: 0000011000000000
87: 0001100000000000
89: 0000000000001110

b.

Run-length encoding
32: 7, 1
33: 8, 4, 3, 1
43: 0, 3
63: 5, 2
87: 3, 2
89: 12, 3

Task 13.

MessagePack

MessagePack does not have at least in-built support for gracefully handling schema evolution. You may need to externally version and control it yourself.

Apache Thrift/Protocol Buffers

Apache Thrift and Protocol Buffers both come with field tags and whether the field is optional or required. This ensures forward compatibility since the old code can simply ignore the newer field tags(such as labor union for Person) that are not recognized. Backward compatibility is possible if the new field is not required, as this would not work if the newer code read older data (labor union would be a missing required field). Changing datatypes may be done gracefully or not, but may lose precision or be truncated.

Avro

Avro relies on the reader and writer having compatible schemas (not necessarily the same schema). By having an new schema as a writer and old schema (compatible) as a reader, you ensure forward compatibility. Conversely, by having an old schema as a writer and new schema as a reader (compatible), you ensure backward compatibility. To ensure compatibility, you can only add or remove fields with a default value. If you add a field with no default value, old data cannot be read by the new reader, breaking backwards compatibility. If you remove a field with no default value, new data cannot be read by old readers, breaking forward compatibility. For labor union to have a default value of null, you would need a union type (e.g. union { null, string }).