Exercise 1 (Theory) TDT4225

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

The flash translation level operates as an abstraction layer between the operating system and the NAND flash memory. It manages the mapping of addresses to physical addresses located in the flash.

Another important task for the FTL is to manage wear leveling such that write/erase cycles don’t hit their limits prematurely. This is because SSDs with NAND flash memory technology have a limited number of write/erase cycles before they wear out. There are typically three categories of wear leveling. No wear leveling (fixed mapping), dynamic wear leveling (dynamic logical mapping) and static wear leveling (dynamic logical mapping). The first one causes blocks to wear out much faster because dynamic blocks will wear out over time and the load won’t be distributed over other blocks. Dynamic wear leveling, on the other hand solves the dynamic block issue when updating a block by placing the new updated block in a new location and marking the old location as invalid. But this means that static blocks will never or very rarely be mapped to a new location. Static wear leveling solves this by doing the same thing as Dynamic wear leveling and in addition, moves static blocks periodically. Static wear leveling gives the longest lifetime but comes at a cost where performance takes a hit. Static and dynamic wear leveling both make use of a garbage collector to clean up old blocks that are considered eligible for erase.

A diagram of a data flow

Description automatically generated

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

This is because performing random writes instead generates internal fragmentation. If the SSD is new or newly erased, the performance hit is not noticeable, but over time a significant difference can be noticed. This is because on an empty SSD there is no old data for the garbage collector to worry about and therefore the performance decrease goes unnoticed for the first 15 minutes or so. When the disk begins to reach full capacity the performance of random writes drops drastically because the file translation layer is occupied with doing updates in-place, since the disk is almost at full capacity. Wear leveling, garbage collection and maintaining mapping information are all tasks that FTL must do at the same time. This becomes a very costly process and causes a huge performance drop in write operations. On the other hand, if it was sequentially written in the order of logical block addresses (LBA), this problem would be avoided because the garbage collector would have an easier time because flash blocks are invalidated block by block. This prevents the need for the garbage collector to do extra work (move valid data around) which it would have done if it was doing random writes.

A graph showing different types of data

Description automatically generated

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

Blocks that are aligned with the page size (W=P) cause no overhead when being written. But when the block being written is less than or greater than the clustered page size, the SSD controller must read the rest of the page and combine it with the new updated data. This is necessary because flash memory requires writing in whole pages or blocks.

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

When data is inserted or updated in an LSM tree-based system, it is first written to a data structure in memory called memtable. Since the memtable resides in memory it allows for fast writes. As soon as the memtable reaches a threshold, it gets flushed to disk as an immutable SSTable (Sorted string table).

The layout of the SSTable is a block based sorted table format in RocksDB as shown below.

A screenshot of a computer

Description automatically generated

Data Block 1 upto n are the key-value pairs. Meta block 1 contains Bloom filters (probabilistic data structure) to determine if a key is not present in a block. Meta block 2 holds statistics about the SSTable, such as size, number of entries etc. Meta block 3 contains the compression dictionary used to decompress and compress data blocks to save space. Meta index block contains indexes/pointers to the various meta blocks. Index block holds pointers/indexes to each data block. Footer contains the pointers to the metaindex block and index block.

1. What happens during compaction in RocksDB?

As data is written and flushed from memtables multiple SSTables accumulate on disk which can contain redundant or overlapping data (because updates on existing data creates a new SSTable, or deleted data that is still present in older SSTables). Compaction is the process of moving data between levels in the background. The compaction process merges and consolidates these SSTables for efficiency in storage space and read performance.

1. 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 optimize writes by accumulating them in memtables, and flushing them to disk in large, sequential batches. This reduces the number of random disk writes. B+ trees on the other hand immediately make the changes to disk, which leads to frequent random I/O operations while trying to maintain a sorted order of the data.

Overtime the LSM-tree’s compaction process reduces the number of files and improves read performance over time but the compaction process itself can be resource intensive. On the other hand, B+ trees rebalance nodes as soon as new data is inserted which has a greater I/O cost.

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

Hardware faults: Hardware faults are faults that are caused by physical components of a system. Examples can be disk crashes, faulty RAM, network failures, power failures, CPU fault etc.

Software faults: Are faults that are present in the instructions/program that instructs/controls computer hardware. These can be caused by software bugs, resource exhaustion (CPU, memory, disk etc being exhausted), dependency issues etc.

Human errors: Human errors are errors that are unintentionally caused by humans. Examples can be incorrect configuration changes, deploying buggy and untested code, mismanaging system resources, working under the influence etc.

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

* Microbatching: Microbatching breaks a continuous stream into small blocks and processes each block as if it were a miniature batch process. If a failure occurs, only the current microbatch is affected, and it can be retried or discarded.
* Checkpointing: Checkpointing is a technique where the state of a stream processor is periodically saved to durable storage. This way, the system can recover from a failure by restarting from the most recent checkpoint
* Atomic commit: Ensures the side effects of processing an event (e.g., writing to a database or sending messages) either take effect completely or not at all. All or nothing approach.
* Idempotence: Ensures the same operations performed multiple times produce the same results. This also means that repeated processing does not result in duplicated side effects.
* Rebuilding state: Stream processors that maintain state, such as aggregation or join tables, need to ensure that this state can be recovered after a failure.
* Failover and Recovery: n the event of a failure, the system must automatically failover to a healthy node and continue processing from the last checkpoint or state save.
* Exactly once semantics: Ensures each even in the stream is only processed once, even in the presence of failures. This is vital for real-time applications like financial transactions.

1. 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: Builds on relational models where data is stored in tables and relationships between data are defined using keys and constraints.

Advantages of SQL: Great for complex relationships, strict schemas (helps with data consistency), ACID (Atomicity, consistency, Isolation, Durability) compliance

Disadvantages of SQL: Changing schema is a complex process, not suited for deeply nested or highly hierarchical data, harder to scale horizontally.

Document model: Stores data as JSON-like documents. Documents are schema-free, which means you are free to store different data structures in the same collection. This is helpful when the data model evolves over time.

Advantages: Flexibility, Horizontal scalability, good for hierarchical or deeply nested structures, related data can be embedded within a document which reduces the need for complex queries (where relationships don’t change often).

Disadvantages: Data redundancy , generally does not provide ACID guarantees, many to many relationships is much harder and inefficient (no joins).

The problem with many to many in the case of author and papers in this example is that if an author writes many papers, their information (name and address) is duplicated in every paper they have written. If the author's address changes, it must be updated in every paper, leading to inconsistencies. Updating the author’s address is also inefficient because it has to be updated in every paper instead of just one place as with relational databases.

Furthermore, if you need to query for all papers written by a specific author, it requires it requires searching through all the documents.

1. 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 model is useful if dealing with data highly connected data, where anything is potentially related to everything. For example, social graphs where vertices are people and edges indicate which people know each other, web graph where V= web pages and E = links etc.

If we take for example a social network based data where friends have friends and we need to retrieve who knows who frequently, to make suggestions of new friends to add. There can also be complex relationships, like group memberships, likes and follows.

Relationships like people who buy this also buy these etc

1. What are the situations that decide that you should use textual encodings instead of binary encodings for sending data?

If readability is important then use textual encodings. Therefor most APIs and configuration files use textual encodings. Its also easier to log and review. Its also used in cases where schema flexibility is prioritized. On the other hand binary encoding are efficient in terms of performance and size.

1. 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

1. Create a bitmap for the values.

• 32: 0000000100000000

• 33: 0000000011110001

• 43: 1110000000000000

• 63: 0000011000000000

• 87: 0001100000000000

• 89: 0000000000001110

1. Create a runlength encoding for the values

(32, 1), (33, 4), (33, 1), (43, 3), (63, 2), (87, 2), (89, 3)

32 7,1

33 8,4,15,1

43 0,3

63 5,2

87 3,2

89 12, 3

1. 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: It is schema-less, meaning that the structure of the data is not explicitly defined in MessagePack itself. Not suitable for this schema evolution because it does not specify a schema. Since the nature of it is simply binary encoded JSON data. managing compatibility (forward or backward) is up to the application logic and not a built-in feature of the format.
* Apache and protocol buffers: both support schemas, classes can be generated from code generation tools and then used for encoding or decoding records of the schema. Adding a new field to the schema is possible given that the new field has a new tag number. Old code reading new field will simply ignore it, making it forwards compatible. As long as a new field being added is not added as a required field, since each tag number is unique, it can also be backwards compatible. For removing fields, only optional fields can be removed.
* Avro: To maintain compatibility, you may only add or remove a field that has a default value. As long as the reader schema and writer schema are compatible, and the new attribute has a default value, backwards and forwards compatibility are possible.

a) When should you use multi-leader replication, and why should you use it in these cases? When is leader-based replication better to use?

If you have a singled datacenter, it makes little sense to use multi-leader replication because the added benefits rarely outweigh the added complexity. If you have multiple datacenters scattered in different geographic locations, it would be beneficial to use this approach. This would improve latency, tolerance of datacenter outages and tolerance of network problems. If possible, multi leader replication should be avoided because of some unwanted effects it could cause. These effects can be noticed when working with autoincrementing keys, triggers, and integrity constraints.

Shortly said, use multi-leader replication if working with geographically distributed systems or when trying to maximize fault tolerance for write operations. Use leader-based replication when consistency is utmost importance or where a simple architecture is required.

b) Why should you use log shipping as a replication means instead of replicating the SQL statements?

Log shipping should be used because the same SQL statements could provide different results for a different node. RAND(), NOW(), UPDATE … WHERE … , etc can cause data to be different than on the leader node if we used statement-based replication.