TU München, Fakultät für Informatik
Lehrstuhl III: Datenbanksysteme
Prof. Dr. Thomas Neumann

Exercises for *Foundations in Data Engineering*, WiSe 22/23

Alexander Beischl, Maximilian Reif (i3fde@in.tum.de)

http://db.in.tum.de/teaching/ws2223/foundationsde

Sheet Nr. 12

Exercise 1 Apache Cassandra is a recently popular no-SQL database system. It is built to be very be very scalable and available, sacrificing consistency if needbe. For the following questions, get familiar with the database system: http://docs.datastax.com/en/cassandra/3.0/

1. Which ACID guarantees does Cassandra provide?

   Solution:

   See https://docs.datastax.com/en/cassandra/3.0/cassandra/dml/dmlTransactionsDiffer.html.

   Atomicity

   In Cassandra, a write is atomic at the partition-level, meaning inserting or updating columns in a row is treated as one write operation. Cassandra does not support transactions in the sense of bundling multiple row updates into one all-or-nothing operation. Nor does it roll back when a write succeeds on one replica, but fails on other replicas. It is possible in Cassandra to have a write operation report a failure to the client, but still actually persist the write to a replica.

   For example, if using a write consistency level of QUORUM with a replication factor of 3, Cassandra will replicate the write to all nodes in the cluster and wait for acknowledgement from two nodes. If the write fails on one of the nodes but succeeds on the other, Cassandra reports a failure to replicate the write on that node. However, the replicated write that succeeds on the other node is not automatically rolled back.

   Cassandra uses timestamps to determine the most recent update to a column. The timestamp is provided by the client application. The latest timestamp always wins when requesting data, so if multiple client sessions update the same columns in a row concurrently, the most recent update is the one that readers see.

   Consistency

   Cassandra 2.0 offers two types of consistency:

   - Tunable consistency Availability and consistency can be tuned, and can be strong in the CAP sense–data is made consistent across all the nodes in a distributed database cluster.
   - Linearizable consistency In ACID terms, linearizable consistency is a serial (immediate) isolation level for lightweight (compare-and-set, CAS) transactions.

   In Cassandra, there are no locking or transactional dependencies when concurrently updating multiple rows or tables. Tuning availability and consistency always gives you partition tolerance. A user can pick and choose on a per operation basis how many nodes must receive a DML command or respond to a SELECT query.

   Linearizable consistency is used in rare cases when a strong version of tunable consistency in a distributed, masterless Cassandra with quorum reads and writes is not

enough. Such cases might be encountered when performing uninterrupted sequential operations or when producing the same results when running an operation concurrently or not. For example, an application that registers new accounts needs to ensure that only one user can claim a given account. The challenge is handling a race condition analogous to two threads attempting to make an insertion into a non-concurrent Map. Checking for the existence of the account before performing the insert in thread A does not guarantee that thread X will not insert the account between the check time and A's insert. Linearizable consistency meets these challenges.

Cassandra 2.0 uses the Paxos consensus protocol, which resembles 2-phase commit, to support linearizable consistency. All operations are quorum-based and updates will incur a performance hit, effectively a degradation to one-third of normal. For in-depth information about this new consistency level, see the article, Lightweight transactions in Cassandra.

To support linearizable consistency, a consistency level of SERIAL has been added to Cassandra. Additions to CQL have been made to support lightweight transactions.

Isolation

Full row-level isolation is in place, which means that writes to a row are isolated to the client performing the write and are not visible to any other user until they are complete.

Durability

Writes are durable. All writes to a replica node are recorded in memory and in a commit log on disk before they are acknowledged as a success. If a crash or server failure occurs before the memtables are flushed to disk, the commit log is replayed on restart to recover any lost writes. In addition to the local durability (data immediately written to disk), the replication of data on other nodes strengthens durability.

2. Which guarantees of a classic ACID database system does Cassandra not provide? What can be gained by ommiting those?
   Solution:
   Atomicity
   No rollbacks. Therefore, there is no need to sync on the undo log.
   Consistency
   Tunable. Quorum consensus can be adjusted on a per query basis. Faster because reading from less replicas. No global constraints available.
   Isolation
   Row based, but no more.
   Durability
   WAL logging, but log is only written periodically.
3. What is Cassandra's scale-out strategy?
   Solution:
   Data is partitioned. Partitions are mapped to nodes in the cluster.
   Scale out is achieved by adding more nodes to the cluster and redistributing partitions to new nodes.
4. What is Cassandra's strategy to handle node failures?
   Solution:
   Replication. Partitions are replicated over multiple machines. Gossip or quorum consensus can be used for writing and reading.

Exercise 2 Familiarize yourself with how data is represented and stored in Cassandra: documentation.

Now create a data model for a Twitter like application that has users, a friends relation and a follower relation between users. When designing the tables, make sure that these queries can be answered efficiently:

1. List all friends of one user
2. List all followers of one user

Explain why in your solution the queries can be efficiently processed in Cassandra, even when there are hundreds of thousands of users with each having many friends and followers.

Solution:
https://github.com/twissandra/twissandra/

```
CREATE TABLE users (
    username text PRIMARY KEY,
    password text
)

CREATE TABLE friends (
    username text,
    friend text,
    since timestamp,
    PRIMARY KEY (username, friend)
)

CREATE TABLE followers (
    username text,
    follower text,
    since timestamp,
    PRIMARY KEY (username, follower)
)
```

The first component of the primary key, the "'partition key"', controls how the data is spread around the cluster. All other components from the "'clustering key"'. It controls how the data is sorted on disk. In this case, the sort order isn't very interesting, but what's important is that all friends and all followers of a user will be stored contiguously on disk, making a query to lookup all friends or followers of a user very efficient.

Exercise 3 Cassandra uses tunable consistency among replications to achieve fault tolerance in the cluster. On a per-query basis, the consistency level can be chosen. See here for a list of available levels. (If needbe, refresh your memory of quorum consensus here.)

1. What is the difference between replicas and partitions in Cassandra?
   Solution:
   Replicas contain copies of the data. Partitions contain parts of the whole database. As replicas contain the same data on multiple machines, updates to the data can reach machines at different times. Therefore, consistency issues (in Cassandra, as partition crossing constraints are not allowed) are only between replicas, not partitions.
2. Which read write consistency mode combinations produce consistent answers?

Solution:
https://www.ecyrd.com/cassandracalculator/

- ONE-ALL
- QUORUM-QUORUM
- QUORUM-ALL (But to strong)

Weaker configurations are eventually consistent.

Exercise 4 Semantic Web, RDF and Sparql

Complementary to the contents of the lecture, you may want to read the following sections of "Programming the Semantic Web" by Toby Segaran, Colin Evans and Jamie Taylor: I1. Why Semantics?, I2 Expressing Meaning, II4 Just Enough RDF - The RDF Data Model It is available online in full text via the TUM library.

Equiped with this information, you are ready to explore wikidata.org. It is a project which provides structured information to Wikimedia sister projects. Find information about the data model at https://www.mediawiki.org/wiki/Wikibase/DataModel/Primer. Conveniently, it also offers a SPARQL interface for your explorations at query.wikidata.org.

Write SPARQL queries to answer the following questions:

1. List everything that uses Munich as object. Wikidata gave the URI http://www.wikidata.org/entity/Q1726 to munich. Therefore, using a prefix definition, you may refer to Munich by using wd:Q1726.
   Solution:

```
SELECT ?a ?b
WHERE
{
        ?a ?b wd:Q1726
}
```

2. Which predicate is used most?
   Solution:

```
SELECT ?b (count(?a) as ?count)
WHERE
{
        ?a ?b wd:Q1726
}
group by ?b
order by desc(?count)
```

3. Which of the cities in the database has the earliest written record?
   Solution:

```
SELECT ?a ?earliestRecord
WHERE
{
        ?a wdt:P31 wd:Q515;
       wdt:P1249 ?earliestRecord.
}
order by asc(?earliestRecord)
```

4. List the transitive subclasses of sport (Q349).

   Solution:

```
select ?x
where {
  ?x wdt:P279+ wd:Q349.
  }
```

5. List the subclasses of sport (Q349) and their labels if there exists one.

   Solution:

```
select ?x ?name
where {
  ?x wdt:P279 wd:Q349.
  OPTIONAL {
                ?x rdfs:label ?name
                filter (lang(?name) = "en")
        }
  }
```