Consistent Reads Across Heterogeneous Storage Systems

Dissertation Project Proposal

Introduction

Modern large-scale applications often employ multiple data storage systems that are optimised for different access patterns. Some of the examples are:

1. OLTP relational databases, that are used to handle frequent state updates

   Example: PostreSQL

2. Search indices, optimised for full-text search

   Example: Lucene, or derivatives such as ElasticSearch

3. In-memory caches, that store the results of expensive queries in order to improve performance

   Example: Memcached, Redis

Apart from performance, another important aspect of the "polyglot" storage infrastructure are the guarantees it provides in terms of atomicity, consistency, isolation and durability. Storing the same data in multiple forms introduces implicit dependencies across the different storage systems. Even though each one provides consistency/isolation guarantees on its own, using them together could lead to partially updated state and hence expose inconsistencies to the user.

Example: Picture a message service with a database for the messages in a conversation thread, alongside a search index. The lack of isolation might lead to a temporary state, in which the search index has been added messages that are not present in the other database yet. Hence, when the user searches for a message, they could get results which are not found in the conversation thread yet.

Ad-hoc solutions to this and related challenges might be difficult, inefficient or simply incorrect.

The "Online Event Processing" paper by Martin Kleppmann, Alastair R. Beresford, and Boerge Svingen describes a general approach which sequences the transactions through a reliable log, to which all storage systems are subscribed [1]. In this way the updates will be performed in the same order, by all systems, even in the face of transient failures, as long as the log provides those features. A simplified pipeline of the OLEP approach can be seen in figure 1.

My project will focus on providing consistent results for read-only transactions across heterogeneous storage systems. As a core goal, it will implement the "Online Event Processing" approach aiming to provide serializable transactions. It will explore the trade offs

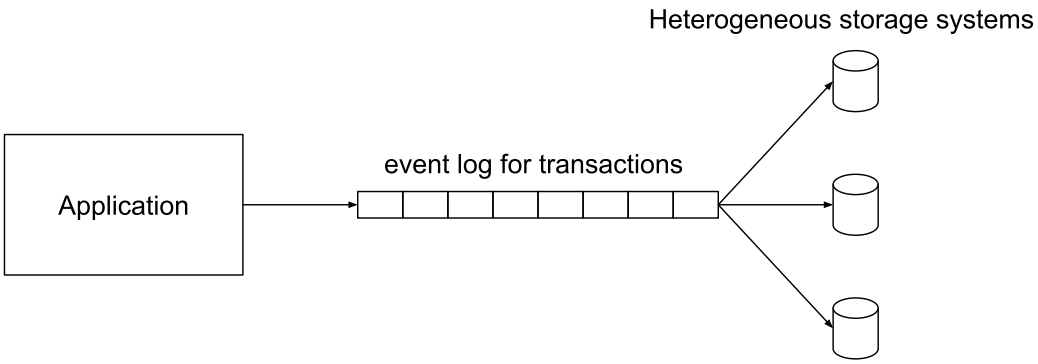

Figure 1: Simplified logical view of the OLEP approach

of this approach and evaluate its performance. As an extension, it will investigate ways to provide better performance whilst maintaining consistent read responses by guaranteeing snapshot isolation, instead of serializability, across such transactions. Snapshot isolation in general provides the following guarantees:

1. All transactions see a consistent state of the database(s), from beginning to termination
2. Readers do not slow down writers, and vice versa

Finally, another extension would be to compare all of the approaches to existing solutions for coordinating heterogeneous distributed systems, for example the X/Open XA specification [3].

Starting Point

The inspiration of my project is the "Online Event Processing" paper by Martin Kleppmann, Alastair R. Beresford, and Boerge Svingen, which describes the event-oriented approach for databases, that I aim to use [1].

Another valuable resource on the topic is Jack Wickham's 2019 part-ii dissertation which provides an implementation of the TPC-C benchmark in an "Online Event Processing" style and compares it to existing database systems.

The book "Designing Data-Intensive Applications" by Martin Kleppmann contains a detailed chapter on isolation, and in particular snapshot isolation and serializability [2]. In general, the book covers a broad range of topics in distributed storage systems, so I anticipate that it's going to be useful throughout the course of writing my dissertation.

I have a basic high level understanding of the technologies being used (Apache Kafka, PostgreSQL, Lucene). I have no hands on experience with either of them.

Resources Required

The main resources I plan to use are:

1. Apache Kafka — used for implementing the event logs used in the OLEP pipeline between the client and the heterogeneous database systems
2. PostgreSQL — as an example of ubiquitous relational database system used in the industry
3. Lucene — a full-text search index which will serve as an example of the different nature of storage systems that could be used

As all of those are open-source, no commercial licenses are required.

The project will be written in Java, which was well covered in the Computer Science Tripos. Additionally, it is a natural fit to the project since all of the aforementioned technologies provide APIs in it.

Substance and Structure of the Project

At its core, the project will aim to provide a consistent view of the data for consecutive read-only transactions, when multiple storage systems are involved. In order to demonstrate the usefulness of the guarantees it offers, I will also implement an example application which employs heterogeneous distributed storage systems.

The example application will be a simple messaging service, that provides a full-text search across conversations. In order to achieve that, it will make use of the following storage systems:

1. PostgreSQL — relational database that stores all messages sent
2. Lucene — full-text search index which will handle full-text search queries
3. *(optional)* An in-memory cache, that will maintain an unread message counter

In order to assert its correctness and evaluate its performance for various cases, the system will be designed so that it allows for:

1. Unit testing of each component
2. Integration tests from carefully chosen initial configurations

   Example1: Putting one of the storage systems some number of transactions behind the others, and then testing that reading from them still produces consistent results

   Example2: Performing long-running read or write transactions, and producing graphs of how those affect the performance of subsequent ones
3. End to end tests and integration with the example application

Success Criteria

Core The project will be deemed successful if it provides consistency in subsequent read transactions across multiple storage systems, even in the face of transient transaction delays in some of the storage systems. Those failure modes will be shown in tests which put the system in carefully chosen configurations, and then assert the consistency of the read results. This stage will be implemented by serializing all transactions through an event log, and as a result, will provide serializable transactions guarantees. It should include performance evaluation in various cases, including some of the systems falling behind with transactions, or having long-running read or write transactions.

The consistency guarantees should be utilized in an example chat application, which employs PostgreSQL relational database, together with a Lucene search index.

The quantitative results obtained in the evaluation will be used as a baseline for comparison with the performance of the extensions below.

Extension 1 A new implementation, in which read transactions are not serialized through the event log (but read requests might be). The aim is that long-running read transactions don't block write transactions. Read results across systems should still be consistent. This almost satisfies the requirements for snapshot isolation, with the exception that long-running write transactions might still block read transactions.

Extension 2 Satisfy all snapshot isolation requirements, namely:

1. All transactions see a consistent state of the database(s), from beginning to termination
2. Readers do not slow down writers, and vice versa

Extension 3 Include comparison with one of the existing approaches for coordinating multiple distributed systems - the X/Open XA specification, together with 2 phase locking [3] [4]. This should investigate the different implementations' performance in the cases of long-running read and write transactions, as well as with various number of transactions and transient delays.

Timetable and Milestones

1. Research (25 Oct 2019 — 01 Nov 2019)
   (a) Explore the technologies used (Apache Kafka, PostgreSQL, Lucene), gain some hands on experience, research their transaction semantics
   (b) Outline concrete implementation approaches for achieving consistency through serializable transactions or snapshot isolation
2. OLEP[1] & example application implementation (01 Nov 2019 — 22 Nov 2019)
   (a) Implement the basic OLEP pipeline. It should support PostgreSQL and Lucene as data storage technologies, but for now it need not provide consistent read results across storage systems.
   (b) Implement the example application and integrate it with the OLEP pipeline
   (c) Demonstrate inconsistent read results with a failing test
3. Provide consistency in read-only transactions by sequencing the reads through the event log (22 Nov 2019 — 06 Dec 2019)
   (a) Demonstrate that the failing use case in 2c is now resolved
   (b) Quantify the cost of providing the read consistency guarantees by comparing the performance of the systems with and without it

   End of Milestone 1 - Core implementation
4. Extension 1 — Research and produce a concrete design (13 Dec 2019 — 20 Dec 2019)
5. Implement Extension 1 (03 Jan 2020 — 17 Jan 2020)
   (a) The new implementation should pass the failing use case in 2c
   (b) Includes a performance comparison with the previous version, as well as with the initial system without any read consistency guarantees
6. Extension 2 — Research and produce a concrete design (17 Jan 2020 — 24 Jan 2020)
7. Implement Extension 2 (24 Jan 2020 — 7 Feb 2020)

   (a) The new implementation should pass the failing use case in 2c
   (b) Includes a performance comparison with the previous versions, as well as with the initial system without any read consistency guarantees

8. Extension 3 — Research how to integrate X/Open XA [3] with the architecture (07 Feb 2020 — 14 Feb 2020)

9. Implement Extension 3 (14 Feb 2020 — 28 Feb 2020)

   (a) The new implementation should pass the failing use case in 2c
   (b) Includes a performance comparison with the previous versions, as well as with the initial system without any read consistency guarantees

   End of Milestone 2 - Extensions implementation

10. Beginning of work on the dissertation text (28 Feb 2020 — 13 March 2020)

    (a) Finish the introduction section
    (b) Finish the preparation section
    (c) Draft version of the implementation section

11. Focus on writing the dissertation and make minor additions to the code driven by the text's needs (13 March 2020 — 10 April 2020)

    (a) Finish the implementation section
    (b) Finish the evaluation section

12. Polish the dissertation text (10 April 2020 — 01 May 2020)

    (a) Finish the conclusions section
    (b) Ask for comments and address them

    End of Milestone 3 - Write up

References

[1] Martin Kleppmann, Alastair R. Beresford, and Boerge Svingen: Online Event Processing

[2] Martin Kleppmann: Designing Data-Intensive Applications

[3] Wikipedia: X/Open XA specification for guaranteeing atomicity in distributed transactions

[4] Wikipedia: 2 Phase Locking