CS346 Coursework 2016/2017
One-copy Serializability in Replicated DB

Arshad Jhumka
Department of Computer Science
University of Warwick, Coventry
United Kingdom, CV4 7AL
h.a.jhumka@warwick.ac.uk, arshad@dcs.warwick.ac.uk

I. Coursework Overview

The coursework (30% of the module) will focus on distributed databases and will consist of three parts:

1) A programming part - that will account for 15%.
2) A report - that will account for 10%.
3) A presentation - that will account for 5%.

II. Introduction

Replicated databases offer increased service availability as there is no single point of failure. There is an increase in performance and scalability. However, there are challenges when replicated databases are used. For example, decisions have to be made regarding copies updates, i.e., the updates can be done in an eager fashion or lazy fashion. Another problem is to decide on the type of consistency required or that can be provided.

In concurrency control of databases (distributed or not), transaction processing (or various other transactional applications such as software transactional memory) needs to satisfy various correctness properties, typically the ACID properties. To provide the isolation property, a condition called *serializability* is used. A transaction schedule is said to be serializable if its outcome (e.g., the resulting database state) is equal to the outcome of its transactions executed serially, i.e., sequentially without interleaving in time. Serializability is the major correctness criterion for concurrent executions of transactions.

However, capturing serializability is challenging in a fully replicated database system. This can be easily observed through the following: Assume a data item X that is replicated at two sites, A and B. Transaction T_1 writes X at A, while transaction T_2 reads X at site B, and B may or may not have received the latest update by the time of the read. To ensure consistency, updates are needed.

In such cases, an analogous correctness condition called *one-copy serializability* is used to capture the serializability condition for distributed systems. One-copy serializability ensures that the effect of transactions performed by clients on replicated data items should be the same as if they had been performed serially on a single set of data items. The notion of one-copy serializability is similar to the concept of *sequential consistency* in concurrent systems. Overall, a replication protocol is required to achieve any type of consistency in a replicated system and, central to the replication protocol is the *choice of update management*: (i) where the database updates are *first* performed, and (ii) how these updates are *propagated* to the other replicas. In terms of where updates are first performed, the techniques can be characterized as *centralized* if they perform updates first on a *master copy*. On the other hand, the techniques are *distributed* if they allow updates over any replica. In terms of how updates are propagated, the techniques can be classed as *eager* or *lazy* techniques.

The combinations of various update management alternatives give rise to a family of replication protocols. The update propagation technique that is used in a protocol typically determines the mutual consistency criterion: eager protocols enforce strong mutual consistency while lazy ones enforce weak. There are various consistency models such as strict consistency vs sequential consistency vs eventual consistency etc. The model chosen should be strong enough to be useful but weak enough to be efficiently implemented. However, all of the protocols would prefer to enforce serializable histories (i.e., one-copy serializability).

III. Programming (15%)

This section will explain the requirement for this year's coursework.

One approach to ensure one-copy serializability is through the use of a *quorum-based* protocol for replica control, which ensures that no two copies of a data item are read or written by two transactions. The following paper, *"Weighted voting for replicated data"*, by David K. Gifford (Proceedings of the seventh ACM symposium on Operating systems principles, 1979) proposed one of the earliest quorum-based protocol. **You are required to read the paper to implement the protocol**.

There are two important parameters for such a quorum protocol, namely a *read quorum*, the size of which is denoted by Q_r and a *write quorum*, the size of which is denoted by Q_w. Once the quorums are set up, sites are appropriately locked whenever a client issues an operation request (i.e., Read(X) or Write(X)).

Your initial setup is as follows: you will setup 4 servers (sites), numbered $1 \dots 4$. You can assume that the system consists of homogeneous sites, i.e., all the properties of the sites are identical so that each will have equal "strength". For example, the sites will have the same availability and the same amount of resources. Each site will have a lock manager, which is responsible for locking the data items being requested locally. Each site will also have a log manager that will write the sequence of events occurring at the site to a local log file. In particular, the events to be logged are (i) the events executed by the lock manager, (ii) the operations being executed, and (iii) the value of the data item. The name of the log file needs to be prefixed by the id of the site. The log files will be part of the output of your program.

A data item, say X, will be replicated at each site. Initially, $X = 0$ at each site. There will be two transactions, executed concurrently by two different clients, that will access the data item X:

$$T_1^1 : [\text{Begin}(T_1); X := 20; Write(X); \text{Commit}(T_1)], \text{ executed at site } 1$$
$$T_2^3 : [\text{Begin}(T_2); Read(X); \text{Commit}(T_2)], \text{ executed at site } 3.$$

Note: You will assume that the database does *not* use multi-version concurrency control to ensure consistency. Rather, you are required to use some form of locking to achieve one-copy serializability. You will also assume that, whenever needed, the sites are all ready to commit (so there is no need to execute an atomic commit protocol) and that there will not be any failure in the system.

For the coursework, you will need to have the following:

1) The two parameters, Q_r and Q_w, will be specified at runtime when running your program. Your program needs to make the necessary checks on allowed values of Q_r and Q_w.
2) A number of transactions will be specified using the syntax, as above. Two transactions executing at the same site will be processed according to the total order imposed by the transactions file. On the other hand, there can be some partial order across sites. The transaction file has to be called "trans.txt", where the transactions are specified.
3) Any number of sites $4 \leq n \leq 10$ can be created and will need to be passed as a parameter.

Your solution must be written in Java and must conform to the required interfaces specified in this document. You are to use the Java sockets library. ServerSockets are used to accept connections and Sockets are used to make connections. Sockets need at least two pieces of information to make a connection: an address and a port. The address could be a hostname or an IP address, if you are running your servers and clients on the same machine you can use the loopback address localhost or 127.0.0.1 to make connections. Please be aware many ports are commonly used for existing services, so on the DCS terminal machines it is recommended to use a port number in the range of 9000 to 9060.

IV. Report (10%)

The second deliverable for your coursework is a 2-page report, which needs to contain the following:

- A 1-page report about the design, implementation and testing of your solution. Your design should include a layout of your SW architecture. **You will also need to clearly explain how to run your solution for testing**. Any specific environment or add-ons need to be made available or specified.
- A 1-page report about the paper "Spanner: Google's Globally-Distributed Database", by James C. Corbett *et al.*, published in the Proceedings of Operating Systems Design and Implementation, 2012. The paper is available for download on the module webpage. Your report needs to contain the following:
  - A short summary of the paper.
  - An explanation of what type of consistency is maintained in Spanner and how this is achieved.
  - A discussion on the possible performance impact of implementing quorum systems in Spanner and the decisions taken to either mitigate or support it.

  You are expected to read other relevant papers when addressing the above questions.

V. Presentation (5%)

Your third and final deliverable for your coursework will be a presentation, to be held in the last week of term 1. Each group will perform a 10-15 minutes presentation of the programming coursework. In the presentation, you will be expected to

- present an overview of the problem you are solving.
- present a design of your solution, explaining your design decisions.

- explain your implementation - mainly the technical details. You are not to include anything regarding project management, software engineering principles etc.
- present your testing strategy.
- conduct a short demo of your project. You are reminded that your program should run on DCS machines. However, during the demo, you will be allowed to use your own computers.
- The presentation is expected to contain roughly 10 slides.

It is expected that every member of the group will contribute to the presentation.

VI. Group Formation

You are required to form a group of 4-5 members for this project. **Once your group is formed, you will need to send me an email**, informing me of your group members. The groups will be posted on the project webpage. We will use this for the presentation part of the coursework.

VII. Requirements

Solutions need to be runnable on the DCS terminal machines. Any references to the literature, such as academic papers, industrial reports or other web resources, must be correctly cited. All code written must be your own, code influenced from any other sources should be referenced in comments in the submitted source code. Plagiarism is a serious offence and will be dealt with according to university guidelines.