December 2013
FINAL EXAMINATION

COMPUTER SCIENCE COMP-512

Distributed Systems

Thursday, December 5, 9am

Examiner: Prof. B. Kemme **Assoc. Examiner:** Prof. J. Kienzle

Answer in exam book.

The examination paper MUST BE RETURNED.

Student Name:

Student Number:

COMP 512 class notes (lecture notes and own handwritten comments), and COMP 512 assignments and midterm questions of the term Fall 2013 and solutions are allowed. The course book is allowed.

Other material (class notes of other courses, old exams etc.) is NOT allowed.

Dictionaries are NOT allowed (except translation dictionaries).

Calculators/Computers/Cell-phones are NOT allowed.

This exam comprises 7 pages, including the cover page.

You can achieve 130 points on this exam. Consider this in your time management.

1 Communication Potpourri (35 Points)

1. (8 points) In class we discussed three different ways to marshal data into messages: CORBA's data representation, Java's serialization, and XML (see slides 30-36 of 512-point2point-comm-2.pdf).

   For two of them, give an advantage it has over the others.

2. (8 points) The slides on point-to-point communication provide a definition for reliable delivery by defining validity and integrity properties (page 7 / slides 14 of the handout)[1]. Similarly, for multicast, the group communication slides define reliable multicast along the properties intgrity/validity/agreement (page 13, bottom slide)[2].

   Provide along similar lines a definition of "reliable remote method invocation" considering that both client and server can fail (that is, can be correct or faulty).

3. (8 points) Software clocks

   Assume a process p requests the time from a time server S. The measured round trip time is 20 ms, and p receives $t = 11:10:13:750$ (*hr:min:sec:ms*) from S.

   a.) *According to the algorithm on page 2 of the lecture notes on time, to what time does p set its local clock?*

   b.) *Assume now that p's current time, just before resetting it, shows* $11:10:14:005$. *What is the problem if p performs this reset? How can this be resolved?*

4. (11 points) Peer-to-peer communication

   In this course we discussed structured and unstructured peer-to-peer networks.

   Indicate three advantages/disadvantages that one mechanism has over the other.

---

[1] (a) Validity: any message sent is eventually delivered to the application (unless the application fails); (b) Integrity: the message received is identical to the one sent, and no messages are delivered twice.

[2] Unreliable: No guarantee that a message sent will be received.
Reliable: (a) Integrity: for any message m, every correct process receives m at most once, and only if some process multicasts m. (b) Validity: if a correct process multicasts a message m, then it eventually receives m. (c) Agreement: if a correct process receives a message m, then all correct processes receive m (atomicity of all correct processes).
Uniform reliable: (a), (b) as reliable delivery. (c) Agreement: if a process (correct or not) receives a message m, then all correct processes receive m (atomicity of all processes)

2 Group Communication over Pub/Sub (20 Points)

Assume a topic-based publish-subscribe system with a central event service (see page 2, upper slide of pub/sub and persistent queue slides). *Subscribe* and *Unsubscribe* are implemented via RMI. In order to *publish* a notification n on a topic t the publisher simply sends the message (t, n) over a TCP connection it maintains with the event service. (That is, it is not an request/reply RMI and the publisher application does not receive a confirmation that the event server has received the message). The event service notifies by simply forwarding (t, n) to all nodes that have subscribed to t. (Again, no request/reply).

Such a system can easily be used to implement the multicast semantics of a group communication system. A node can join group g by subscribing to topic g. A $multicast(g, m)$ is simply implemented as a publish of m on topic g.

1. (5 Points) Assume that the event server never fails but all other nodes might crash. Assume the underlying point-to-point communication is asynchronous and there are no network partitions. *What level of reliability does the system provide (unreliable, reliable, uniform reliable)? Provide some explanation.* For the definitions of the various levels of reliability, see handout page 13 bottom slide, or footnote on previous page.
2. (15 Points) Now assume the event server can also fail (but still no network partitions). *Depict a primary/backup based replication solution (that is, event server is replicated with a primary and a single backup) that provides reliable delivery to the group despite possible failures of the event server primary. Describe how you would enhance/implement the subscribe and the publish/notify primitives in order to provide reliable delivery to the group. Consider the actions at the primary, at the backup and at the other nodes (some "replication-awareness" needs to be integrated at the stubs at the client). Your solution can ignore the failure of the backup. You do not need to provide a detailed algorithmic description with data structures but the individual steps and semantics need to become clear.*

3 Optimistic Concurrency Control (30 Points)

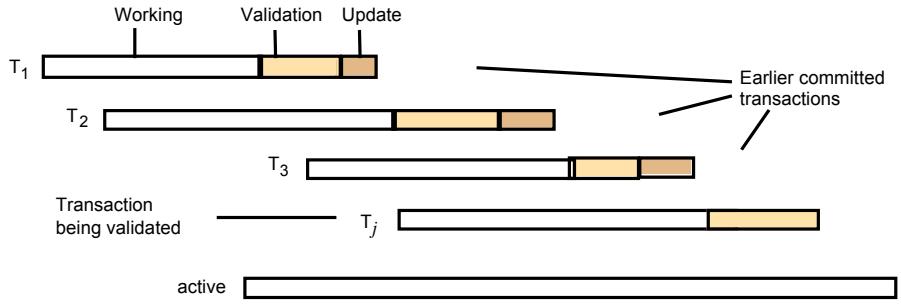
In class, we described how application servers use some form of *optimistic concurrency control* in combination with a database system that uses strict 2PL. Optimistic oncurrency control can also be directly implemented within the database system:

- Writes are performed on local copies only visible to the transaction. They build the *WriteSet* of the transaction.
- A read operation of a transaction T reads the latest committed version of a data item (or, if T has already updated this data item, its own writes). All data items read build the *ReadSet* of the transaction.
- When the client submits the commit request for T, validation takes place: if none of the data items in the *ReadSet* was updated by a concurrent transaction T', then validation succeeds, T commits, and the data items in the *WriteSet* become the latest committed versions visible to others. If one of the data items in the *ReadSet* was updated since the start of T, validation fails and T aborts (simply discarding its WriteSet). Validation and the final commit occur in a critical section.

Below an algorithmic description for database system DBS:

- At start of DBS: TC := 0 // *TC is a transaction commit counter*
- Upon begin of transaction T_i // *(could be first read/write request)*
  1. $StartTC(T_i) := TC$ // *keep track of who committed before T_i started*
  2. $CommitTS(T_i)$:= undefined // *don't know yet when T_i commits*
  3. $WriteSet(T_i) := ReadSet(T_i) := LocalC(T_i) := \{\}$
- Upon $r_i(x), w_i(x)$ request of T_i
  1. If first operation on x then
     - x_c := copy of x (latest committed version)
     - $Local(T_i) := Local(T_i) \cup \{x_c\}$
  2. If $r_i(x)$, then $ReadSet(T_i) := ReadSet(T_i) \cup \{id(x)\}$
  3. If $w_i(x)$, then $WriteSet(T_i) := WriteSet(T_i) \cup \{id(x)\}$
  4. execute operation (return/update) on local copy x_c
- Upon commit request for T_i, execute in critical section:
  1. For all $StartTC(T_i) < j <= TC$ // *validate against all concurrent transactions*
     - Let WS be the WriteSet of transaction T with $CommitTS(T) = j$
       - If $WS \cap ReadSet(T_i) \neq \emptyset$, then abort T_i
  2. If not aborted
     - $TC := TC + 1$
     - $CommitTS(T_i) = TC$
     - for each $id(x) \in WriteSet(T_i)$ // *make x_c the official last committed version.*
       - write value of $x_c \in LocalCopies(T_i)$ to x
  3. return abort resp. commit to user

Below figure shows some examples. T_1 commits before T_j starts so they are not concurrent and T_1 is not considered in the validation of T_j. However, T_2 or T_3 are concurrent to T_j. If they have written some data items that T_j read, then validation of T_j would fail. This mechanism guarantees serializability where the serialization order is the same as the validation/commit order[3].



1. Now assume assume a homogenous distributed database system. The system consists of n sites $S_1, ...S_n$. Each site maintains part of the database without replication. A transaction T_i can be submitted to any site S_j. When T_i submits an operation on data object x and S_j does not have x, S_j forwards the operation to the corresponding site S_k that maintains x and forwards S_k's response back to the client. Each site keeps track of the read- and write-operations executed on the data items it is responsible for and performs the validation for those data items. Upon commit request for T_i, S_j initiates the validation phase, and if necessary the write phase. Only when there are no conflicts at any participating site, the validation phase can succeed. That is the participating sites have to agree on whether validation succeeds or fails.

   Provide a such a distributed optimistic concurrency control protocol in algorithmic form. A site might now receive requests from clients (as in above algorithm) but also from other sites. Note that each transaction has a home site *where it submits all its requests to. For now, simply assume that validation and commit are together in a critical section without showing how this could be achieved.*

2. Problems can occur if the validation/write phase is not in a critical section and concurrent transactions could be in that phase at the same time.

   How can you guarantee that at any given time, there is at most one transaction in validation/write phase in the entire system?

[3]Aborting T_j if T_j reads data items that T_2 or T_3 wrote is necessary, because T_j might have not read the versions written by T_2 resp. T_3 but previous one, which would violate the desired serialization order.

4 Performance Analysis (20 Points)

1. (10 Points) Assume a distributed system with a hierarchical calling tree (slide 30 of commit protocol). Assume point-to-point communication. Assume a very regular tree where each node (except the leaves) has 4 children and each path from root to leave consists of 3 nodes.

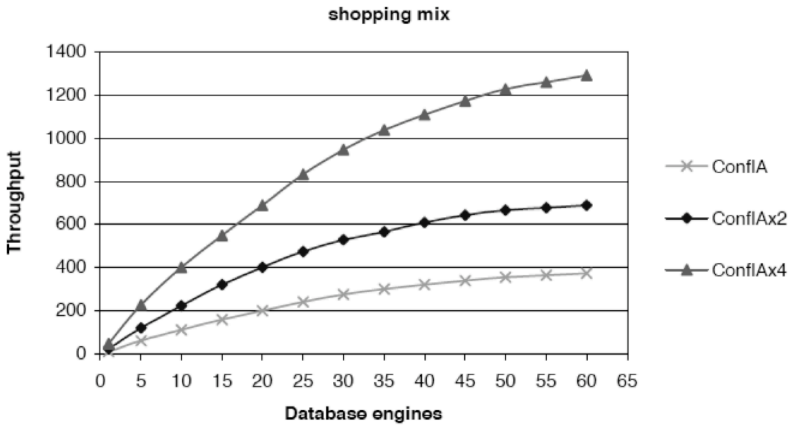
   a.) *Give the overhead in number of messages in the commit case for both the flattened and the hierarchical commit protocol for such a calling tree. The flattened protocol is the standard 2PC where the root is the coordinator and all other are participants, that is, the root knows the identity of all participants and not only its direct children, and communicates with all of them directly during the 2PC. The hierarchical commit protocol is the one that you have in detail developed in assignment 3. Consider only vote-requests, votes and decision messages but no acknowledgement (EOT) messages.*

   b.) *In case of commit, after how many message rounds can the root return the commit confirmation to the client?*

2. (10 points) Performance of a replicated system:

   The figure has at the x-axis the number of database replicas. The y-axis shows the maximum throughput that a given system can achieve (just before saturation). The three graphs show results when using machines with different power (ConflA being the least powerful, ConflAx4 being the most powerful). The workload used in this experiment had a moderate update rate of around 20% (i.e., 20% of all operations were updates).

   Describe the results that you see in the figure and give an explanation for the behavior. What does the graph tell you about the general potential for scalability of a replicated database?



5 System Development (25 Points)

The railway systems in European countries are impressive. For example, the German railway system has 5 million passengers per day, out of which around half a million are on long-distance (inter-city) travel. Around 200 trains per day connect German cities with cities of neighbor countries. The railway systems in other countries are similar.

Most countries offer online access to the train-schedules. The functionality for online clients is typically as follows:

- People can enter start and destination cities, day and/or time of departure, and receive detailed information about their travel possibilities, and the prices. Schedules are provided even if travel is international.
- People can buy train tickets online. A ticket is typically a one-way or return ticket for a connection from A to B (and return) valid for a specific time period (e.g., 7 days). Travellers can use any train on that connection during that time period. Sometimes, the ticket also includes the reservation for a seat on a specific train/time (then the ticket is only valid for this particular train departure). Clients receive the tickets by email and can print the tickets by themselves.
- Browsing of travel information does not require the user to log in, but logging in allows a user to"remember" popular connections, etc. In order to purchase a ticket, the user has to be logged in and run a session.

The path finding algorithms needed to find connections and schedules are compute intensive and access large amounts of data. Apart of this query load, ticket purchases represent typical short transactions consisting of a small number of read and write operations. They require the full ACID properties.

Furthermore, the national railway company of a given country maintains the schedule information of its own trains, i.e, updates the information whenever new connections are introduced, their times change, or they are removed, but clearly may not update the schedule information of other countries. Furthermore, in order to develop optimal new schedules, the usage of all the connections are continuously observed in order to analyze the demand. This includes large-scale analysis of both connection data and ticket purchasing data.

Design a possible architecture of the system. Some questions that need to be answered are: How and where is which data stored? Is data cached, replicated, distributed? How is data consistency handled? How are the various queries efficiently executed? Motivate your design decisions. Feel free to add to your descriptions figures if they help showing the design principles.