8-hour Online Written Exam of Advanced Computer Systems – ACS, 2020-2021

Department of Computer Science, University of Copenhagen (DIKU)
Date: January 22nd, 2021

Preamble

Solution

Disclaimer: In the following, we present solution sketches for the various questions in the exam. These solution sketches are provided only as a reference, and may lack details that we would expect in a complete answer to the exam. Moreover, some of the questions may admit more than one correct solution, but even in such cases only one solution sketch is provided for brevity. Solution sketches are colored for visibility.

Note, in addition, that the evaluation of the exam takes into account our expectations regarding solutions, the actual formulations provided, the weights of various questions, but also and most importantly the overall evaluation of the exam assignment as a whole. This evaluation is performed by both internal and external examiners and grades are finally provided by discussion and consensus. As such, it is not advised to reason about final grades based on this document.

This is your final 8-hour online written exam for Advanced Computer Systems, block 2, 2020/2021. The exam will be evaluated on the 7-point grading scale with external grading, as announced in the course description.

- Hand in a **single PDF file**, in which your exam number is written on every page.
- Your answers must be provided in English.
- Hand-ins for this exam must be individual, so **cooperation with others in preparing a solution is strictly forbidden.**
- The exam is open-book and all aids are allowed. If you use **any sources other than book or reading material for the course**, they **must be cited appropriately.**
- You may also use your computer or other devices, but you may not access online resources or communicate with any other person in preparing a solution to the exam.
- Remember to write your exam number on all pages.
- You can exclude this preamble in your hand-ins if you directly write on this file.
- This exam has a total of 18 pages excluding this preamble, so please double-check that you have been given a complete exam set before getting started.

Expectations regarding formatted spaces

For each question in this exam, you will find formatted space where you can provide your answer. The spaces provided are designed to be large enough to provide satisfactory (i.e., concise and precise) answers to each of the questions.

Exam number: ____________

Expectations regarding question importance

Questions carry indicative weights. The weights will be used during evaluation to prioritize question answers towards grading; however, recall that the exam is still evaluated as a whole. In other words, we provide weights only as an indication so that you can prioritize your time, should you need to during the exam. You cannot assume that weights will be divided equally among subquestions.

Exam number: ____________

The following table summarizes the questions in this exam and their weights.

Question	Weight
Q1 Atomicity	7%
Q2.1 Bottlenecks	4%
Q2.2 Performance Evaluation	3%
Q3.1 Schedules	13%
Q3.2 Deadlocks	7%
Q4.1 The Missing Log	6%
Q4.2 The Recovery	10%
Q5.1 Lamport Logical Clock and Vector Clock	10%
Q5.2 Replication	10%
Q6.1 Basic Concepts of Distributed Transactions	2%
Q6.2 Two-Phase Commit	8%
Q7.1 Selection	4%
Q7.2 Join and Aggregate	16%

Errors and Ambiguities

If you find any errors or ambiguities in the exam text, you should clearly state your assumptions in answering the corresponding questions. Some of the questions may not have a single correct answer, so recall that ambiguities could be intentional.

8-hour Online Written Exam of Advanced Computer Systems – ACS, 2020-2021
Department of Computer Science, University of Copenhagen
Date: January 22nd, 2021

Exam number: ________

1 Atomicity (7%)

Give three flavors of atomicity considered in the lecture and explain what each of them aims to achieve. For each flavor, give and briefly explain one main challenge that makes achieving this flavor a non-trivial task and give one example approach towards achieving it.

Atomicity: strong modularity mechanism that requires that implementations do not expose the fact that a complex high-level action is composed of several sub-actions. Difficulty: leaky abstractions, i.e., systems tend to expose too much of their internals; failures of one component might crash the entire system. Approach: modular system design (clients & services), clean interfaces.

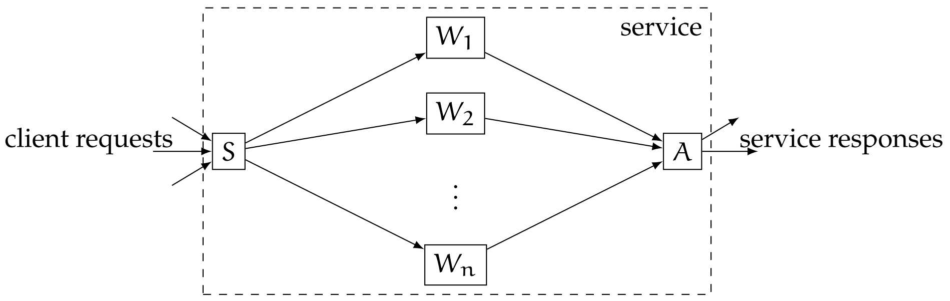
Before-or-after atomicity: I(solation) from ACID: transaction should run as if they were the only one in the system. Difficulty: want to use concurrency for performance; due to concurrency the transactions are not running on their own. Approach: Concurrency Control, e.g., S2PL or OCC.

All-or-nothing atomicity: A(tomicity)+D(urability) from ACID: transaction should be either executed completely and its effects must be durably persisted or not at all. Difficulty: Want No Force (not every write is forced to disk) + Steal (transaction will be granted some space in main memory upon request, which will cause a page modified by another, potentially not-yet-committed transaction to be written to disk yet) for performance, No Force implies that parts of a committed transaction may not have been persisted (danger for D if system crashes—need to redo). Steal implies that parts of a not-yet-committed transaction may have been persisted (danger for A if transaction aborts—need to undo).

Approach: Recovery, e.g., ARIES (redo + undo).

2 Performance (7% in total)

Consider a service that internally uses the following architecture to process the incoming client requests:



The service consists of several distributed components, each of which runs on a separate compute node (i.e., on physically separate hardware):

- a *scheduler* S that distributes the workload between the workers
- n *workers* W_i that perform the actual computation
- an *aggregator* A that combines the computation results into a single output.

The communication between the different compute nodes proceeds as shown in the picture. In particular the workers do not communicate with each other.

Consider two possible scenarios:

1. All workers perform the same computation on different disjoint parts on the input. For example, consider a word count service that inputs words. The words are uniformly assigned to workers by S. Each worker outputs word count histograms, which A periodically combines into a single histogram.
2. Each worker performs a different task on a part of the input. The parts assigned to different workers may (need to) overlap. For example, the service might periodically compute different aggregates upon inputing words and numbers, e.g., the sum of all seen numbers, the set of longest seen words, or the number of words of length greater than the largest number seen so far. The scheduler is responsible for giving each worker the required information to perform its assigned task.

2.1 Bottlenecks (4%)

What are the main performance bottlenecks of the service in both given scenarios? In particular, which bottlenecks are common to both scenarios, and which are specific to only one of them? Which architectural alternatives could alleviate some of the bottlenecks. Briefly justify your answers.

Common bottlenecks are the throughput of S and A—these are centralized components (hopefully their computation is not the expensive part of the overall computation). Both could be parallelized in principle. The workers' throughput can also be a bottleneck. In the first scenario all workers are equal, in the second the one with the most difficult task could become a bottleneck. In the second scenario: data duplication can also be a bottleneck. Combine first and second scenario to alleviate bottlenecks (data parallelism + task parallelism).

2.2 Performance Evaluation (3%)

How would you evaluate the performance of the above architecture in both scenarios? Which performance metrics would you measure? How would you measure them? Would you setup the experiments differently for the two scenarios?

Measure latency and throughput. End to end measurements + individual measurements for components. For latency: use latency markers in the input. See when they arrive at the output (works for both scenarios). For throughput: fix latency see how many events can be processed while maintaining sub-second (or some other threshold) latency.

3 Concurrency Control (20% in total)

3.1 Schedules (13%)

We consider three transactions `T1`, `T2`, and `T3` consisting of the following actions.

```
T1: R(X) W(X) R(Y) W(Y) C        T2: R(Y) W(Y) W(Z) C        T3: R(X) R(Z) W(Z) C
```

IMPORTANT: We ask you to come up with five **different** transaction schedules of `T1`, `T2`, and `T3` satisfying different constraints given below. For each schedule **explain** why it satisfies the constraints.

Schedules, 3.1.1: Give a serializable transaction schedule of `T1`, `T2`, and `T3`.

Solution

```
T1: R(X) W(X) R(Y) W(Y) C
T2:                          R(Y) W(Y) W(Z) C
T3:                                             R(X) R(Z) W(Z) C
```

The given schedule is serial, thus also serializable. (2%)

Schedules, 3.1.2: Give a conflict-serializable transaction schedule of `T1`, `T2`, and `T3`.

Solution

```
T1:                    R(X) W(X) R(Y) W(Y) C
T2:                                             R(Y) W(Y) W(Z) C
T3: R(X) R(Z) W(Z) C
```

The given schedule is serial, thus also conflict-serializable. (2%)

Schedules, 3.1.3: Give transaction schedule of `T1`, `T2`, and `T3` that is **not** conflict-serializable.

Solution

```
T1: R(X) W(X)                R(Y) W(Y) C
T2:                R(Y) W(Y)                          W(Z) C
T3:           R(X)                         R(Z) W(Z) C
```

The precedence graph has a cycle `T1 -WR(X)-> T3 -RW(Z)-> T2 -RW(Y)-> T1`. (Smaller cycles are possible, too.) (2%)

Schedules, 3.1.4: Give a conflict-serializable transaction schedule of `T1`, `T2`, and `T3` that could **not** have been generated by a scheduler using strict two-phase locking (S2PL) with lock upgrades (from shared to exclusive).

Solution

```
T1:      R(X) W(X)              R(Y) W(Y) C
T2:                                          R(Y) W(Y) W(Z) C
T3: R(X)           R(Z) W(Z) C
```

The given schedule conflict-equivalent to the one from 4.1.2, thus conflict-serializable (alternative: draw precedence graph). S2PL acquires locks gradually, but releases them at once at the end: at the beginning T1 and T3 successfully acquire shared locks on X, but then T1 needs to upgrade the lock, which fails, because T3 is not ready to release its shared lock. (3%)

Schedules, 3.1.5: Give a conflict-serializable transaction schedule of T1, T2, and T3 that could have been generated by a scheduler using strict two-phase locking (S2PL) with lock upgrades (from shared to exclusive), but **not** by a scheduler using conservative two-phase locking (C2PL) with lock downgrades (from exclusive to shared).

Solution

```
T1: R(X)                   W(X) R(Y) W(Y) C
T2:                                          R(Y) W(Y) W(Z) C
T3:      R(X) R(Z) W(Z) C
```

The given schedule conflict-equivalent to the one from 4.1.2, thus conflict-serializable (alternative: draw precedence graph). S2PL could have generated this schedule by getting shared locks for X in T1 and T3. T3 proceeds by getting a shared lock for Z and upgrading it to an exclusive lock. Then T3 releases all its locks, which allows T1 to get an exclusive lock for X. From that moment on the execution is serial. C2PL acquires all locks necessary for a transaction at the beginning of its execution (and releases them gradually). T1 requires an exclusive lock for X. After that T3 cannot obtain a shared lock for X, making this schedule impossible for C2PL. (4%)

3.2 Deadlocks (7%)

Deadlocks, 3.2.1: Which of the two-phase locking variants can result in deadlocks? Explain why by giving concrete schedules with deadlocks. Also explain why other variants cannot generate deadlocks.

S2PL and 2PL (i.e., any variant that takes locks gradually).

Example schedule T1X(A) T1W(A) T2X(B) T2W(B) T1S(B) T1R(B) T2S(A) T2R(A)

Other schedules take all required locks at once at the beginning. If this fails, no lock is taken (no way for others to wait for this schedule). (3%)

Deadlocks, 3.2.2: Consider the following two strategies aiming to prevent deadlocks among a fixed set of transactions $\{T_1 \dots T_n\}$ being scheduled by strict two-phase locking (S2PL):

1. If a transaction T_i requests a lock that is currently held by another transaction T_j, then T_j is aborted if $i < j$; otherwise T_i waits.
2. If a transaction T_i requests a lock that is currently held by another transaction T_j, then T_i waits if T_i holds strictly more (other) locks than T_j does; otherwise T_i aborts.

Which of these strategies prevent deadlocks? For those that do, explain why. For those that do not, give a counterexample schedule that exhibits a deadlock.

Strategy 1 prevents deadlocks. It is a variant of wound-wait; the transaction index breaks the cycle (transaction with larger index wait for the ones with smaller index). The indices are assigned statically and do not change.

The second is a “faulty” variant of wait-die. It can lead to deadlocks as follows:

`T1X(A) T2X(B) T2X(C) T2X(A)[waits] T1X(D) T1X(E) T1X(B)[waits]`

The underlying problem is that the set of locks can grow over time under S2PL. (4%)

4 Recovery (16% in total)

We consider a transactional database that uses the ARIES algorithm as its recovery mechanism. At the beginning of time, there are no transactions active in the system and no dirty pages. A checkpoint is taken. After that, three transactions, T1, T2, and T3, enter the system and perform various operations, **including at least one page update per transaction**. Eventually, a crash happens, the system restarts

Exam number: ____________

and proceeds with the recovery. As the result of the analysis phase, the algorithm arrives at the following transaction table (sorted by lastLSN) and dirty page table (sorted by recLSN):

XACT_ID	status	lastLSN
T3	committed	4
T2	aborted	7
T1	running	8

PID	recLSN
P42	3
P<EN>	5 + (<EN> modulo 2)
P99	8

Here, <EN> refers to your personal Exam Number. For example, if your Exam Number is 154 then the last line of the dirty page table reads “P154 | 5” (because 154 modulo 2 is 0).

4.1 The Missing Log (6%)

Complete the below log, in the usual format, to be a log at the moment of the crash, which could have lead to the shown situation. Assume that the log sequence numbers are the (consecutive) natural numbers starting with 1.

```
LSN     PREV_LSN     XACT_ID     TYPE         PAGE_ID UNDONEXTLSN
---     --------     -------     ----         ------- -----------
1       -            -           begin CKPT   -       -
2       -            -           end CKPT     -       -
...
```

Solution

```
LSN     PREV_LSN     XACT_ID     TYPE         PAGE_ID UNDONEXTLSN
---     --------     -------     ----         ------- -----------
1       -            -           begin CKPT   -             -
2       -            -           end CKPT     -             -
3       -            T3          update       P42           -
4       3            T3          commit       -             -
5       -            T1          update       P154          -
6       -            T2          update       P42           -
7       6            T2          abort        -             -
8       5            T1          update       P99           -
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX CRASH XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

OK to undo T2’s update before the crash (but the end log records for T2 and T3 should not be there)

4.2 The Recovery (10%)

Based on your log, complete the recovery procedure. Show:

1. the sets of winner and loser transactions;
2. the values for the LSNs where the redo phase starts and where the undo phase ends;
3. the set of log records that may cause pages to be rewritten during the redo phase;
4. the set of log records undone during the undo phase;
5. the contents of the log after the recovery procedure completes.

For each of the items above, briefly justify your answer.

Solution

1. Winner T3. Losers T1, T2. (1 %)
2. Redo starts at 3 (min. recLSN in DPT), undo ends at 5 (oldest LSN belonging to a loser) (2 %)

3. Redo 3, 5, 6, 8 (all pages may have been in memory at the moment of the crash). (1 %)
4. Undo 5, 6, 7, 8 (all belong to loser transactions) (1%)
5.
```
LSN     PREV_LSN     XACT_ID     TYPE         PAGE_ID UNDONEXTLSN
---     --------     -------     ----         ------- -----------
1       -            -           begin CKPT   -               -
2       -            -           end CKPT     -               -
3       -            T3          update       P42             -
4       3            T3          commit       -               -
5       -            T1          update       P154            -
6       -            T2          update       P42             -
7       6            T2          abort        -               -
8       5            T1          update       P99             -
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX CRASH XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
9       4            T3          end          -               -
10      8            T1          abort        -               -
11      10           T1          CLR          P99             5
12      7            T2          CLR          P42             -
13      12           T2          end          -               -
14      11           T1          CLR          P154            -
15      14           T1          end          -               -
```
(5%)
1 can be solved without having solved the log exercise
For others, correctness relative to the solution of previous exercise.

Exam number: ____________

5 Reliability (20% in total)

The architect of a E-Commerce service, DanskeShop, has adopted an architecture that decomposes the system into several fine-grained independent services, including Payment, Order, and Shipment.

Clients only interact with the Order service, which in turn work with Payment and Shipment to complete the client orders. In particular, Order accept two types of client requests, submit an order or cancel an on-going order. All the communication is *asynchronous*.

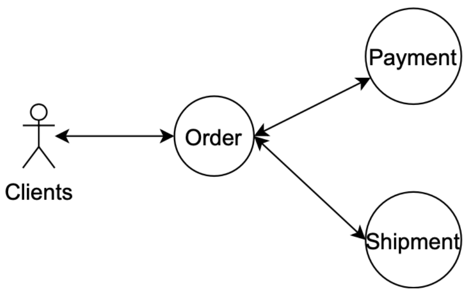


Figure 1: System Architecture of DanskeShop

When Order receives a client's order, it *asynchronously* sends the following two messages:

1. an NewOrder message with the order ID and the order amount to Payment;
2. an NewOrder message with the order ID, the list of ordered products and the shipping address to Shipment.

Payment and Shipment will, repsectively, send a message AckOrder back to Order upon the completion of processing the order. When Order receives both the AckOrder messages from Payment and Shipment, it acknowledges the client.

A client can send a request to the Order service to cancel an on-going order. Once Order receives a cancellation request from a client, it sends the following messages

1. a Cancel message with the order ID to Payment;
2. a Cancel message with the order ID to Shipment.

Payment and Shipment send, respectively, an AckCancel message back to Order as the response. Upon receiving the AckCancel messages from both Payment and Shipment, it acknowledges the client of the order cancellation.

A client can submit the same order request multiple times. But the system should guarantee that only one of the requests is carried out.

Figure 2 shows the message exchanges of a particular excecution, where a client sends a new order request and then shortly after, cancel the same order while it is still on-going. For brevity, we omit the messages sent between the client and Order. At event $e1$, Order receives a new order request from a Client, and at $e3$, it receives an order cancellation request from the same client regarding the same order. The same client regrets the cancellation, and resend the same order request with the same order ID again at $e7$.

Furthermore, some events (including e2, e4 and e6) represent both receiving a message and sending back the responding message. For example, event $e2$ represents Payment receives the NewOrder message and sends back the AckOrder message. Each one of these events is considered a single atomic event.

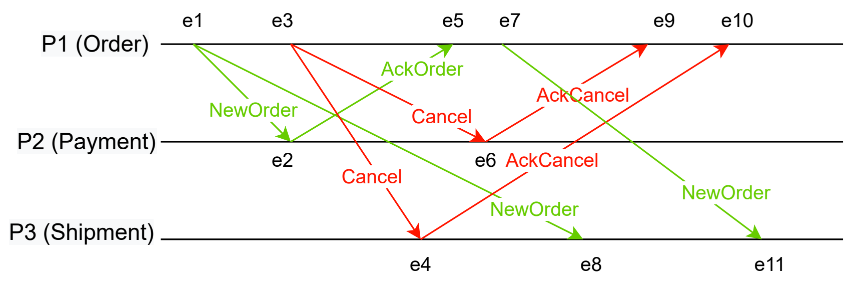


Figure 2: Exchange of messages between services

5.1 Lamport and Vector Clock (10%)

Based on the above scenario and the messages exchanges depicted in Figure 2. Anwser the following questions.

Lamport and Vector Clock, 5.1.1: Write the Lamport logical clock of each event, assuming each process starts with a clock at 0.

e1: 1, e2: 2, e3: 2, e4: 3, e5:3, e6:3, e7:4, e8:4, e9:5, e10:6, e11:5

Lamport and Vector Clock, 5.1.2: Write the vector clock of each event, assuming each process starts with clock at $(0, 0, 0)$.

e1:(1,0,0), e2:(1,1,0), e3:(2,0,0), e4:(2,0,1), e5:(3,1,0), e6:(2,2,0), e7:(4,1,0), e8:(2,0,2), e9:(5,2,0), e10:(6,2,1), e11(4,1,3)

Lamport and Vector Clock, 5.1.3: P1 received two messages at e9 and e10, respectively. By using Lamport logical clock or vector clock, can P1 tell the message received at e10 was actually sent before the one received at e9? If so, how? Othersie, why? Briefly explain the reasoning of your answer.

No, the vector clocks associated with the messages are not comparable.

Exam number: ____________

Lamport and Vector Clock, 5.1.4: At e4, P3 received a Cancel message, and then immediately responded with an AckCancel message. At e8 and e11, P3 received, respectively, a NewOrder message with the same order ID. In the figure, the Shipment's responses to these two messages were ommitted. Recall that, when a client submit the same order quest multiple times, the system should guarantee that only one of them is executed. What are the correct responses of Shipment to these two NewOrder messages? How Shipment can enforce the correct responses? Brief explain your reasoning.

Shipment should ignore the message at e8, but process the order recieved at e11 and send back an AckOrder message. By comparing the vector clocks of messages received at e4, e8 and e11, it knows that the message at e8 was sent before the message at e4, but the one at e11 was sent after. Therefore, it can enforce the same effect as the messages are processed in their sendign order.

5.2 Replication (10%)

To enhance the reliability of DanskeShop system, the states of the services have to be replicated. Suppose each service should be replicated into 5 copies. Answer the following questions.

Replication, 5.2.1: The Shipment service maintains the inventory of products available at DanskeShop. After an order is processed, the stocks of the products involved in the order would be deducted accordingly. Compare the pros and cons of synchronous replication and asynchronous replication for replicating Shipment.

In synchronous replicaiton, update can return only after all copies (or the write quorum) have completed the update. Therefore, it can take a longer time to complete a client request. However all the copies are guaranteed to be consistent at anytime. Upon failure, a replica can take over without data consistency problems. In asynchronous replication, latency is lower. But it may suffer data consistency problems.

Exam number: ____________

Replication, 5.2.2: Suppose synchronous replication is adopted to replicate the state of Shipment into 5 copies. The write quorom is 3 ($Q_w = 3$) while the read quorum is 2 (Q_r=2). Would this setup be problematic? Why or why not? Briefly explain the reasoning of your answer.

$Q_w + Q_r$ has to be greater than N. This is to ensure Read can access the up-to-date version.

Replication, 5.2.3: To replicate the Order service, which of the following replication scheme can also enhance the scalabliy of the service in terms of I/O throughput: a) Asynchronous Primary-Site, b) Asynchronous Peer-to-Peer, and c) Synchronous Replication? How is the state of Order replicated under the chosen scheme? Briefly explain the reasoning of your answer.

Asynchronous Peer-to-Peer. All the peers can act as a master site. Each master site can be the owner of a disjoint fragment of the state. Conflicts can be avoided. Primary-site can only have one master copy, therefore less scalable.

Exam number: ____________

6 Distributed Transaction (10% in total)

Consider again the scenario of the e-commerce site DanskeShop described in Question 5 and answer the following questions.

6.1 Basic Concepts of Distributed Transactions (2%)

Basic Concepts of Distributed Transactions, 6.1.1: Does the processing of a client's new order request fullfill the atomicity property in the case of node failures? Use an example scenario to briefly explain why or why not.

No. If there is a node failure, a request could be blocked and half done. For example, if the Shipment service fails while Order and Payment are working. Then the order request will be half done (and blocked) and will not be aborted.

Exam number: ___________

6.2 Two-Phase Commit (8%)

DasnkeShop decides to implement another version of the system, where Two-Phase Commit (2PC) is used to carry out a new order request. Suppose the three services are deployed on separate machines, and the transaction coordinator runs on the machine of the Order service, while all the three services take part in each transaction. In this system version, the client cannot cancel an order once it is submitted. Answer the following questions.

Two-Phase Commit, 6.2.1: Briefly describe the procedure of 2PC to commit a New Order transaction over the three services when there is no failure occured. Write down the coordinator's log records regarding the 2PC protocol in this case. For each log record, write down its type, such as "prepare", and the necessary information to tolerate node failures.

Describe 2PC with one coordinator and three subordinates. A log record for each message to be sent: "prepare" (or "canCommit") and "commit" (or "doCommit"). Each log record should contain the XactID, and the address of the subordinates, e.g. (T1, "prepare", Order, Shipment, Payment).

Exam number: ____________

Two-Phase Commit, 6.2.2: Suppose the site of Order fails after the coordinator has decided to abort a transaction and has written the abort log record to the disk, but before the abort messages were sent to any subordinates. Assume Shipment has voted yes (i.e. commit) and Payment has voted no (i.e. abort), describe a scenario where Shipment and Payment can proceed to complete this trasaction. Expalin the scenario step by step.

Two possible answers: 1) Shipment and Payment will be blocked until Order recovered. Describe the recovery process at the coordinator and the subsequent process of the 2PC protocol. 2) The coordinator can send the participants of the trasactions in the prepare message. Shipment and Payment can communicate with each other. As long as one of the votes is a "no", then the transaction ca nbe abort in both Shipment and Payment.

Exam number:

Two-Phase Commit, 6.2.3: Suppose 2PC with presumed abort is implemented, and the site of Order fails after it has decided to abort and has written the abort log record to the disk. Desribe what could happen to Shipment if it has not received the abort message, and what could happen to Order after it restarts from the failure.

If Shipment has voted no, it should just abort. Otherwise, it blocks until the site of Order recovers, and ask for the outcome of the transaction. After Order restarts, it will look into transaction table for all the active transactions, and then it looks into the log file for the status of these active transaction. If it finds an abort log record of an incomplete transaction. It will then remove the corresponding transaction from the transaction table, and then will resend the abort message to all the subordinates. If it receives an inquiry about the outcome of a transaction that is not in the transaction table, it will simply reply abort.

7 Data Processing (20% in total)

Consider again the DanskeShop scenario described in Question 5. The data analyst at DanskeShop is required to analyze some sales statistics. The following tables are provided to the data analyst:

- Orders(orderID, productID, unitPrice, quantity), which stores the unit price, and the quantity of each product in each order.
- Shipments(orderID, addr, city, country), which stores the shipping address of each order.

The Orders table has 25,000 pages, and Shipments has 1,500 pages.

7.1 Selection (4%)

The first task is to select all the records in Orders, where unitPrice is greater than 500 and the quantity is greater than 10.

Selection, 7.1.1: Suppose there is no index on the attribute of unitPrice or quantity. What is the most efficient algorithm to evaluate this query? What is the I/O cost of the algorithm? Birefly explain your answer.

a sequential scan. 25,000 I/O.

Selection, 7.1.2: Suppose there is a non-clustered B+ tree index on unitPrice, and there are 50,000 records that can meet the selection conditions. What is the worst-case I/O cost of using this index to answer this query? Is it always a good idea to use this index than the algorithm that you come up with in the previous question? Briefly explain your answer?

The worst case is one I/O per output record, therefore 50,000 I/O. This is because the data is not sorted in the same order as the index. No, this could be more expensive in the worst case.

Exam number: ___________

7.2 Join and Aggregate (16%)

Another task of the data analyst is to produce the regional sales statistics: "for each product and each city, return the total income from the sales of this product in all the orders delivered to this city". The schema of the results should be Sales(productID, city, salesIncome). Suppose the database server used by the data analyst has 150 buffer pages. Answer the following questions.

Join and Aggregate, 7.2.1: The first step is joining the tables Orders and Shipments on oderID. What is the most efficient algorithm to perform this step? What is the I/O cost of this algorithm? For simplicity, you can ommit the I/O of writing out the join results. Briefly explain your answer and justify your choice of algorithm.

Choose grace hash join. $1500 > 150 > \sqrt{1500}$, so it needs 2 passes. I/O= $3 * (25,000 + 1,500)$. SMJ cannot be completed in 2 passes, because it requires $B > \sqrt{1500} + \sqrt{25000} = 197$. But we have only $B = 150$

Join and Aggregate, 7.2.2: If the server can be upgraded with more memory, is there a case where your choice of algorithm in Question 7.2 be different? If yes, give the precise condition (in terms of memory size), under which you would make a different algorithm choice, and explain your answer. Otherwise, briefly explain the reasoning.

Yes. Given the memory is x pages, where $1500 > x > \sqrt{1500}$, the cost of BNLJ is $1500 + 1500/(x-2) * 25000$ and the cost of grace hash join is 3*(25000+1500). If the memory size $x \geq \lceil (1500 * 25000)/(3 * (25000 + 1500) - 1500) \rceil + 2 = 483$ pages and less than 1,500 pags, then BNLJ could be a more efficient algorithm.

Exam number: ___________

Join and Aggregate, 7.2.3: The second step is grouping the records of the results from the previous step based on the values of productID and city and then perform aggregation on each group. Assume the output size of the previous step is 100,000 pages and the server is upgraded to having 500 buffer pages. Moreover, suppose the final output should be sorted by productID. Describe the most efficient algorithm to perform this step. What is the I/O cost of this algorithm? Again, you can ommit the I/O of writing out the results. Briefly explain you anser and justify your choice of algorithm.

Sort the data primarily on productID and secondarily on city. The data can be sorted in 2 passes. Generate the aggregate while merging the sorted runs in the 2nd pass. IO=$3 * 100,000$. The hash-based algorithm can generate the groupby aggregate with the same I/O. But it cannot generate sorted resutls, hence a subsequent sorting is needed, which makes it more expensive.

Join and Aggregate, 7.2.4: Suppose the data analyst is provided with a clutser of 5 servers, rather than 1 server. Describe the most efficient parallel algorithm to peform the processing in Question 7.2. Assume the input data of this step is randomly stored on the 5 servers. What is the expected amount of data to be transfered over the network? Briefly explained and justify youre answer.

Range paritition on $\langle$productID, city$\rangle$. Sort each partition and perform aggregation locally using the same non-parallel algorithm described above. The amount of data transfer would be $4/5 * N$.