ACS Theory Assignment 1

This assignment is due via Absalon on December 8, 23:59. While individual hand-ins will be accepted, we strongly recommend that this assignment be solved in groups of maximum three students. A well-formed solution to this assignment should include a PDF file with answers to all exercises.

All homework assignments have to be submitted via Absalon in electronic format. It is your responsibility to make sure in time that the upload of your files succeeds. While it is allowed to submit scanned PDF files of your homework assignments, we strongly suggest composing the assignment using a text editor or LaTeX and creating a PDF file for submission. Email or paper submissions will not be accepted.

Learning Goals

This assignment targets the following learning goals:

- Identify and explain techniques for improving performance in computer systems.
- Discuss the design of a top-level abstraction in terms of lower level abstractions and naming. Explain strategies employed for correctness, performance, and fault tolerance.
- Analyze the serializability of transaction schedules, explaining why certain schedules are serializable and some are not.
- Predict decisions that would be made by different concurrency control protocols, e.g., variants of two-phase locking (2PL) or optimistic concurrency control.

Question 1: Techniques for Performance

Regarding techniques to improve performance of a computer system, answer the following questions:

1. Explain the difference between concurrency and parallelism. Provide one example of each. (3 sentences)
2. How does concurrency influence throughput in a computer system? Is its influence always positive, always negative, or is there a trade-off? Explain. (2-3 sentences)
3. Give a specific example of a fast path optimization and explain why it is one. Are there any trade-offs in using such optimizations? (3 sentences)

Question 2: Fundamental Abstractions

In this exercise, you will design a memory abstraction that exposes the whole aggregated memory of a cluster of K machines each with a storage system as a single address space with N entries [0..N-1]. At first, we assume that the K machines are fixed and do not change throughout the execution and N is significantly larger than K. You do not need to consider redundancy to mask failures. However, you need to consider that individual machines can fail and have your abstraction handle this in a clean way (i.e., no segmentation fault of the whole system when a machine is down).

The memory abstraction you should design is schematically shown in Figure 1.

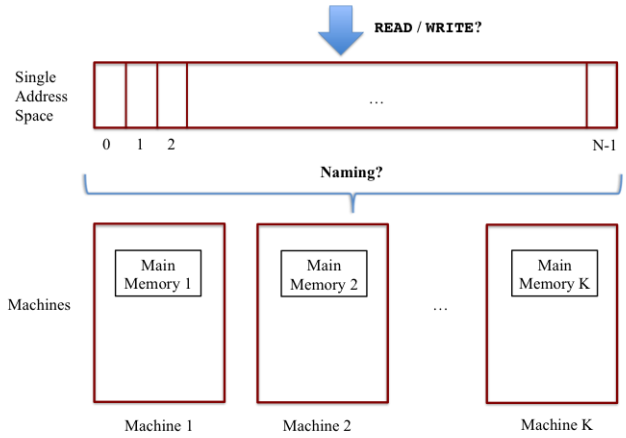

Figure 1: A top-level memory abstraction consolidating the lower-level memories of multiple machines in a cluster.

NOTE: There is no single correct answer to the proposed design of your abstraction. For example, there are many possible naming approaches to map the single address space exposed by the abstraction to the address spaces of the separate machines in the implementation. However, different methods might vary in their advantages and disadvantages. In the questions below, we will consider the quality of your arguments and the care with which you identify the multiple fundamental abstractions employed in your design proposal.

To phrase your solution, address the following:

1. Explain at a high level how you would map addresses at the single address space to addresses in each individual machine. The name mapping scheme selected should, if possible under stated conditions, have an expected constant running time. Discuss in your answer: Why is your name mapping efficient? Are there any centralized components in your design? What kinds of faults can the design tolerate? (2 paragraphs)

2. Show the pseudocode of the READ and WRITE functions of your memory abstraction. Your pseudocode should show both translation of addresses (naming) as well as communication with individual machines to obtain the actual data. In your description, clearly state which data structures and algorithms are used for name translation and resolving of addresses. (for each function, list the API plus a short description of 1-2 sentences and then present the pseudocode implementing the API)

3. Should the READ/WRITE operations against your single memory space be atomic (in terms of before-or-after atomicity)? Discuss why or why not. If you believe that it should be atomic, argue how you would achieve that property when multiple clients are accessing your abstraction concurrently. If you believe that it should not be atomic, describe what abstraction you would need to provide to clients to implement atomicity when they need it. (3 sentences)

4. Consider now that the K machines are not fixed, i.e., they may enter and leave the system, and naming must adapt to these changes. How does this change in requirements affect your design and the faults it can tolerate? How would you adjust the design to tolerate faults stemming from dynamic naming? (3 sentences)

Question 3: Serializability & Locking

Consider the following transaction schedules with the time increasing from left to right. C indicates commit.

Schedule 1

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

Schedule 2

```
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
```

Schedule 3

```
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
```

Now answer the following questions:

- Draw the precedence graph for each schedule. Are the schedules conflict-serializable? In Schedule 1, can you change a single read action into a write action (without affecting the timing) or vice versa so that your answer to the previous question changes (i.e., a conflict-serializable schedule should become not conflict-serializable or vice versa)? Explain why or why not.

- Could the schedules have been generated by a scheduler using strict two-phase locking (strict 2PL)? If so, show it by injecting read/write (also known as shared/exclusive) lock operations in accordance to strict 2PL rules. If not, explain why.

Question 4: Optimistic Concurrency Control

Consider the following scenarios, which illustrate the execution of three transactions under the Kung-Robinson optimistic concurrency control model. In each scenario, transactions T1 and T2 have successfully committed, and the concurrency control mechanism needs to determine a decision for transaction T3. The read and write sets (RS and WS, respectively) for each transaction list the identifiers of the objects accessed by the transaction.

Scenario 1

```
T1: RS(T1) = {}, WS(T1) = {3},
  T1 completes before T3 starts.
T2: RS(T2) = {2, 3, 4, 6}, WS(T2) = {4, 5},
  T2 completes before T3 begins with its write phase.
T3: RS(T3) = {3, 4, 6}, WS(T3) = {3},
  allow commit or roll back?
```

Scenario 2

```
T1: RS(T1) = {5, 6, 7}, WS(T1) = {8},
  T1 completes read phase before T3 does.
T2: RS(T2) = {1, 2, 3}, WS(T2) = {5},
  T2 completes before T3 begins with its write phase.
T3: RS(T3) = {3, 4, 5, 6}, WS(T3) = {3},
  allow commit or roll back?
```

Scenario 3

```
T1: RS(T1) = {2, 3, 4, 5}, WS(T1) = {4},
  T1 completes read phase before T3 does.
T2: RS(T2) = {6, 7, 8}, WS(T2) = {6},
  T2 completes before T3 begins with its write phase.
T3: RS(T3) = {1, 2, 3, 5}, WS(T3) = {7, 8},
  allow commit or roll back?
```

For each scenario, predict whether T3 is allowed to commit. If T3 is allowed to commit, state which validation tests were necessary to reach that conclusion. If T3 must be rolled back, state all tests which T3 fails, along with the offending objects from the read and write sets of T1 and T2.