Theory Assignment 2

Xuanlang Zhao(kxs806), Runzhuo Li(qmc887),
Yaokun Li(xnf483)

December 8, 2022

Contents

1 Problem Set Solution

concurrency 是一个处理器处理多个任务，parallelism是多个处理器处理

1.1 Question 1: Techniques for Performance

1
The differences between concurrency and parallelism are that concurrency focuses on dealing with more than one tasks simultaneously but for parallelism, it divides one task into many subtasks and processes them by multiple CPUs at the same time. An instance of concurrency is that one man is eating, he is also watching TV and making a phone call at the same time. An example of parallelism is that two or more men are having dinner together, their goal is to finish the dinner.
2
The impact of concurrency on throughput depends on the number of concurrency. If there's a graph, When the number of concurrency is small, the throughput is increasing with more concurrency. However, when the number of concurrency reaches an edge, it has less influence on the improvement of throughput. When the number of concurrency is quite large, the throughput decreases.
3

Caching is an example of the fast path optimization. Caching split the pipeline into two paths, the fast path for reading/writing cached data and the slow one for reading/writing non-cached data.
trade-offs:
1. it does improve performance but increases the complexity of the whole system
2. fast path is often at the expense of slower slow path.
3. fast path increase overhead

1.2 Question 2: Fundamental Abstractions

1.We can abstract the separate physical memory into a whole logical memory space, and let's set the new space's memory starting with 0. We can map a single address to a machine number by dividing the single address into k parts, so each part consists n/k addresses. An address of a single space has two parts of data, one is a logical address, and the other is a machine address. If we know a logical address x, we can get its machine number by (x/k) + 1, and this procedure costs O(1) time, and then search for data in machine memory with the second address also costs O(1) time, so it's very efficient.

One transform component converts the logical address to a machine number. If one or more machines are removed, the transform component should

record current active machines, and if a user is looking for that machine, it will tell the user it's invalid. However, the design cannot tolerate adding a machine as that will cause n/k change.
2.

```
Read(x):
        x1,x2 <- x
        machine <- getmachine(x1)
        data <- getdata(machine,x2)
        return data
```

In read pseudocode, we first split two parts of the input address into a single space address and a machine memory address. In getmachine, we calculate the space address and turn it into the actual number of the machine, then we send a request to the machine for the data in the specific address and then return the data.

```
Write(x,value):
        x1,x2 <- x
        machine <- getmachine(x1)
        res <- writevalue(machine,x2,value)
        return res
```

In Write pseudocode, we first do the same procedure to get the machine number, and then we send the value to the machine to let it write it into memory. Then the server will send back a result indicating if the write succeeded or failed for some reason; we then return the result.
3.As read and write may have some error during multi connect to a machine, these operations should be atomic and have before-or-after atomicity. To achieve that, some concurrency control should be applied; we can use CS2PL to achieve it.
4.If the machine leaves the system, we can record it in getmachine, so if it doesn't exist anymore, there's no data on that address. If a machine is added to the system, this current design won't work. We need to update correspondence in getmachine to let it be able to redirect the request from x1 to the machine number by adding a map to the get machine number component so that each address can be redirected to the right machine.

1.3 Question 3: Serializability & Locking

$$\begin{tikzcd}
 & T_2 \arrow[dl] \\
T_1 & \\
 & T_3 \arrow[uu]
\end{tikzcd}$$

(a) schedule 1

$$\begin{tikzcd}
 & T_2 \\
T_1 \arrow[ur] & \\
 & T_3 \arrow[uu] \arrow[ul]
\end{tikzcd}$$

(b) schedule 2

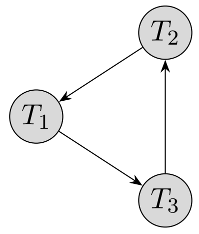
$$\begin{tikzcd}
 & T_2 \arrow[dl] \\
T_1 \arrow[dr] & \\
 & T_3 \arrow[uu]
\end{tikzcd}$$

(c) schedule 3

Figure 1: Precedence graphs

As we can see, there is no cycles in the precedence graphs of shcedule 1 and schedule 2, so both schedule 1 and schedule 2 are conflict serializable. But for shcedule 3, there is a cycle (T1 $\rightarrow$ T3 $\rightarrow$ T2) so it is not conflict serializable.

To turn schedule 1 into not conflict-serializable schedule, we can simply change the first R(X) in T1 to W(X). If we do so, the first action R(X) in T3 in schedule 1 will conflict with the W(X) we just change. The precedence graph turns to Figure 2



$$\begin{tikzcd} & T_2 \arrow[dl] \\ T_1 \arrow[dr] & \\ & T_3 \arrow[uu] \end{tikzcd}$$

Figure 2: Precedence graphs of schedule 1 if we change first R(X) in T1 to W(X)

All the schedules can not be generated by strict 2PL.

For schedule 1, T2 will get exclusive lock on Y first, and when T1 need it to execute W(Y), it is still held by T2.

For schedule 2, T3 will get shared lock on X first, and when T1 needs exclusive lock on X to execute W(X), the shared lock on X is still held by T3.

For schedule 3, T1 will get exclusive lock on X first, and when T3 needs shared lock on X to execute R(X), the exclusive lock on X is still held by T1.

1.4 Question 4: Optimistic Concurrency Control

Scenario 1
As T1 completes before T3 starts, it allows T3 to commit or roll back. For T2, it completes before T3 begins with its write phase, $T2_{WS}\{4,5\} \cap T3_{RS}\{3,4,6\} = \{4\} \neq \varnothing$. Therefore, T3 won't be allowed to commit or roll back.
Scenario 2
Because T2 completes before T3 begins with its write phase and T1 completes read phase before T3 does. Write set of T2 should be compared with Read and Write sets of T1. $T2_{WS}\{5\} \cap T1_{RS}\{5,6,7\} = \{5\} \neq \varnothing$. Conflict occures, thereofore, T3 won't be permitted to commit or roll back.
Scenario 3
Based on T2 completes before T3 begins with its write phase and T1 completes read phase before T3 does, similarly to Scenario 2, the Write set of T2 should be compared with the Read and Write sets of T1. $T2_{WS}\{6\} \cap T1_{RS}\{2,3,4,5\} = \varnothing$, $T2_{WS}\{6\} \cap T1_{WS}\{4\} = \varnothing$. Everything is fine now. Then Write set of T1 should be compared with Read and Write sets of T3. $T1_{WS}\{4\} \cap T3_{RS}\{1,2,3,5\} = \varnothing$, $T1_{WS}\{4\} \cap T3_{RS}\{7,8\} = \varnothing$. It's evident

that T3 can be allowed to commit and roll back.

2 Reference

"Q1(1): Good.
Q1(2): Good.
Q1(3): Good.

Q2: 1

Q3(1): There is a local deadlock in Node 2: T1's R(D) waits-for T4's W(D)
Q3(2): Yes but there is also a second cycle with the correct version of Node 2
Q3(3): You are on the right track, but note that transactions do not send or receive a response. It is the nodes (and coordinator) that hold the logs and send and receive responses.

Grade: 92/100"