University of Waterloo

Faculty of Engineering

Department of Electrical and Computer Engineering

ECE 454

Assignment 6

Prepared by

Chan, Carl

UW Student ID Number: 20383063

UW User ID: c73chan@uwaterloo.ca

and

Li, Debin

UW Student ID Number: 20389489

UW User ID: d73li@uwaterloo.ca

30 July 2014

1. By definition, read your write states that: A write is completed before a successive read, no matter where the read takes place. From the following example, we can see that Read your Write consistency was satisfied, as when R(x2) was performed, W(x1) was already completed by both locations.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **L1** | W(x1) |  |  | |  | W(x3) |
| **L2** |  | WS(x1;x2) |  | R(x2) | |  |

Writes follow reads, however, was not satisfied. Writes follow reads consistency states that any successive write will be performed on a copy that is at least as up-to-date with the value most recently read by the process. As seen from the counter-example, when W(x3) was performed at L1, it does not yet have the latest updates read by R(x2), as W(x2) has not being propagated to L1 from L2.

1. As seen from the example, the consistency follows Writes follow reads, as the R(x2) is picking up the latest updates at L2, and W(x3) is performed on a copy that is up-to-date with the value most recently read by the process, hence the Write follow reads condition is satisfied.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **L1** | W(x1) |  |  |  | R(x1) |
| **L2** |  | WS(x1;x2) | R(x2) | W(x3) |  |

However, in the example above, the Read your write condition was not satisfied, as both W(x2) and W(x3) has not propagated from L2 to L1 yet.

1. 1. We propose a brute force method.
      1. Every subset of the servers is calculated.
         * This has , where is the total number of servers. This is as the calculation of a combination is in factorial time, and we need to find combinations of size from 1 to .
      2. For each subset calculated above, find every mapping of all clients to the servers in the subset, with the condition that each client can only be associated with one server.
         * As all clients can be mapped to any of the (upper-bounded by ) servers, each client has choices. This means this runs in .
      3. After calculating a mapping in the previous stage, compare its total cost to a running minimum cost. If it is lower, it becomes the new running minimum mapping and its cost is the new running minimum cost.
         * The summing of the costs in the mapping and comparison to the running minimum takes place in time in terms of the number of nodes. This is as there are client-server connections to add, as well as (upper-bounded by ) server costs.
         * Assuming that all costs are either less than, or around the maximum server cost , each add and compare operation takes time. This is as they are bit-wise operations, with each add potentially producing a sum with one more bit than the operands.
         * Thus, each running of this step takes:

Thus, the overall runtime of this algorithm is:

* 1. If the server costs have an upper bound, then the part of the time complexity becomes a constant and goes away. Thus, the new time complexity becomes:

This is still not efficient.

1. Yes, there is an efficient algorithm. Let the oracle be defined as R(A, B, E, N), where “A” is the set of clients, “B” is a set of servers (with server costs), “E” is the set of allowed associations between clients and servers, and “N” is the maximum number of servers to choose.

First, we need to find the set of servers in the minimum solution. To do this, we start with the full set of servers and find the cost if we could choose from all the servers. This gives us our starting reference. Next we use the oracle to find the costs if we remove one server from consideration, for each server. The cost will always be either equal to or greater than the reference. The server who when removed caused the smallest increase in cost is then removed from consideration, and the cost without that server becomes the new reference. This process is repeated on the remaining servers, eliminating one server each time until there are “k” servers left. At that point, all the remaining servers whose removals would not result in an increase in cost are then removed. This will result in the minimum set of servers with cardinality which results in a minimum solution. This runs in . This is as at each step one server is eliminated until (at least) there are no more than “k” left. During each step, all costs after removing a potential candidate server must be compared. This results in quadratic time complexity.

After finding the subset M of servers in the minimum solution, we need to find the mappings between servers and clients. You can imagine a connected graph with some dummy vertex D in the center, directly connected to the server vertices with 0-weighted edges. This means any MST starting at D will include all the server vertices. The zero-weighted edges mean all servers will be added to the MST before any clients, so there will never be a case where a client is attached to more than one server due to the lack of loops in an MST. We want all of the M servers to be used as we know all of them are part of the minimum solution.

Define an adjacency between each server and all the clients with a weighting of . The total cost of the servers is constant regardless of the server-client mapping so it does not factor into this. By running Prim's algorithm starting at D as the root, an MST will be constructed involving all the chosen servers and clients who would result in minimum cost. Prim’s algorithm using an adjacency map runs in time, where is the number of servers chosen, and n is the total number of clients. As , this becomes .

When combining both steps, the overall time complexity is:

1. Let the fraction of accesses for F via P be and , for client 1 and client 2, respectively.

This means the number of accesses to F via P are and for client 1 and client 2, respectively.

Therefore, let be to total number of accesses to F via P from both clients.

* 1. The drop threshold should be where the total cost for P of servicing requests for F without its own replica does not exceed the total cost of doing so with a local replica. This is as even if these are equal, the resources allocated to the local replica of F could be used for something else. The condition is only met if the total number of requests to P for F is:
  2. The threshold for replication is when the total cost of accessing P without a replica exceeds the total cost of accessing a replica on P with the same number of accesses, if it had been replicated from the beginning. So the replication threshold is:
  3. The condition of enforces that there are no concurrent writes. Without this constraint, two writes which happen at the same time and contact a disjoint set of servers will be seen as having been done at the same time. With it, all sets of servers used by different writes will always overlap, meaning the servers in the overlap can decide on which write was more recent. This way, the timestamps of any two writes can be used to determine which is newer.

The constraint of means that the set of servers queried during a read will always overlap with the sets used for any writes. Due to this, the effect of the most recent write (which can be determined by comparing timestamps) will always be present in at least one of the servers being read from. Due to the previous constraint, there are never any conflicts due to concurrent writes.

Therefore, a read always gets the latest version of whatever it is reading.

* 1. The lack of propagation delay and the guarantee that enough servers will always respond to a write means that the data has strict consistency. This is as there can never be “concurrent” writes and all writes are immediately seen by any subsequent reads. Thus, operations in this system can be said to have a strict ordering. As strict consistency is stronger than causal consistency (if you have strict, you satisfy causal), you therefore have causal consistency.