HW 8

1. Two Phase Commit:

Explain the action taken in the two phase commit protocol after restart if the coordinator or subordinate nodes fail.

There are two parts to answering this question. The first section will answer what happens when the one of the subordinate nodes fails. The second part will answer what is done if one of the coordinator nodes fails.

If a subordinate node fails then there are several steps we may need to take for T depending on what is in the log. If we have an end record for T then we don’t need to do anything. However, if we have a log record for T but we don’t have an end record than we will need to undo or redo T. If we have a prepare log for T but not a commit or abort we can say that this node is a subordinate. We will then contact the coordinator to enquire about the status of T and will then accordingly redo or undo T. Additionally, if we have no prepare log for T then we need to abort T.

If we are the coordinator and we have a log for T and we have an end log for T we don’t need to worry about T because T was completed. However, if we do not have an end log for T then we need to keep sending commit/abort messages to subordinates until the acknowledge the request. Again if we have no prepare log for T we need to abort T along with sending out the abort message to all subordinates for T.

1. Multi-Version concurrency control:

Consider transactions T1, T2 and T3 with timestamps 10, 20 and 30 respectively in a database system that follows multi-version concurrency control protocol: T1 is a reader and T2 and T3 are writer transactions.

* 1. Let T1 read and T2 read and update data item D. if transaction T# updates only D, provide a schedule where the multi-version concurrency control method will not restart T3.

Shown below is schedule that will not cause T3 to be restarted.

|  |  |  |  |
| --- | --- | --- | --- |
| Time | T1 | T2 | T3 |
| 10 | Read D |  | Read D |
| 20 |  |  | Update D |
| 30 |  | Read D |  |

T3 will not be restarted in this case because when T3 goes to write there will be no transactions without standing RTS that don’t have corresponding WTS. Thus T3 will be free to write data item D.

* 1. Again, assume that T1 reads and T2 and T3 read and update only data item D provide a schedule where the multi-Version concurrency control method will restart T2.

Show below is a schedule that will cause T2 to be restarted.

|  |  |  |  |
| --- | --- | --- | --- |
| Time | T1 | T2 | T3 |
| 10 | Read D |  | Read D |
| 20 |  | Read D | Update D |
| 30 |  | Update D |  |

In this case T2 will be forced to restart because RTS(T3) < RTS(T2) and T3 has not yet written. This means that T2 must wait until T3 has written to the database.

1. Quorum Reading:

Let the number of copies of a data item in a cluster, N, be 10 and the number of nodes that participate in a successful write, W, be 5.

* 1. What is minimum number of nodes that should participate in a successful read, R?

We can say that R + W > N so given that W is 5 and N is 10 than the number of nodes that must participate In a successful read must at least be 6 nodes.

* 1. Now, consider a sloppy quorum with the same values for N and W. What is the minimum number of nodes that should participate in a successful read R?

In a sloppy quorum a read only requires one node. However, we cannot guarantee that this item will be the most up to date item.

1. Eventual consistency and vector clocks:

Each row in the following table shows vector clocks of different copies of the same data on a system with three nodes SX, SY and SZ. Explain if the copies in each row have a conflict.

Row one does not contain a conflict because the time stamp for copy one and copy two on Sx and Sy are the same. The only difference is that Sz has been added. Thus we can say with certainty that Copy 1 happened before Copy 2.

Row two does have a conflict. This is because Sy both have different values and Sz was added to Copy 2. Given that Sy for Copy 2 has a later time stamp than copy 1 on Sy and copy1 has the data item in Sz it’s not possible to tell which was done first thus they are parallel branches and must be resolved.

1. Page Rank:

Assume the Following graph depicts a part of the Web, where nodes represent pages and edges show hyper-links. Find out the pages whose PageRank values are greater than-zero and their relative PageRank values in the graph. You do not need to perform the fix point computation to determine the PageRank values. Instead, you should guess the PageRank values based on your understanding of the PageRank algorithm and explain why you think they are correct. If it is not possible to make any educated guess for some page(s), you should explain why.

For this graph every node would have a page rank that was positive. This is because the page rank is simply the probability that a random click would lead you to a given page. The only way it would be possible for a page to have a page rank of 0 would be if there were no links to that page. Thus it’s not possible to randomly click to that page. In the graph shown for problem 5 ever node has at least one edge pointed to it thus there is at least one link for every page. Therefore it is possible that via random clicks that you could end up at any one of these pages and that it therefore isn’t possible to have a page rank of 0.

In terms of relative page rank g would have the highest page rank as it has two links pointed at it and only one going outbound. E would have the lowest age rank because it has one link pointed at it and two going out bound. The rest of the pages would have the same page rank because they have one link pointed at them and one link pointing to another page.

1. MongDB
   1. Write a Query that returns cuisine type of the restaurant name The Dead Rabbit.

db.restaurants.find({"name": "The Dead Rabbit"},

{"cuisine": 1, "\_id": 0})

* 1. Write a query to create an index on the name attribute of the restaurants.

db.restaurants.createIndex({name: “text”})

* 1. Write a query that uses the index created in part b to return all the restaurants containing the term rabbit in their name.

db.restaurants.find({$text {$search: “Rabbit” }})

* 1. Write an aggregate query to show total count of restaurants in each borough.

db.restaurants.aggregate(

[

{ $group: {“\_id”: “$borough”, “count”: {$sum: 1} } }

]

)