Problem Set 7

Problem 1. Consider a text document retrieval application you wish to parallelize. Requests to the system are Boolean queries, e.g., Q= "X and Y and Z" You have built an inverted list index. To answer query Q, the system retrieves from disk the lists for X, Y, and Z, intersects them in memory, and obtains the list of documents having all three words. You are considering two options to split the index across a set of N shared-nothing computers:

(A) You partition at random the documents into N sets. For each set, you build an index on one of the computers. In this case, to execute Q you broadcast it to all computers, and each searches in parallel for the documents in its set that match Q.

(B) The words are partitioned (at random) into N sets. The postings list for the words in a set are stored in one computer. To see how to process Q, say list X is stored at computer 1, list Y at computer 2, and Z at computer 3. Say computer 1 runs the query. It sends a message to computer 2 asking it for the list Y, and another message to 3 asking for the Z list. When the lists arrive (over the network) at computer 1, they are intersected in its memory to get the final answer.

You now want to compare the two strategies based on the expected throughput, using a "back of the envelope" analysis. Assume all queries have two terms, i.e., they are all of the form "find documents with words X and Y." Assume all lists are of the same length, and each list is stored contiguously on disk. Each of the N = 5 computers has a single disk that can perform up to 20 IOs per second. Reading a list takes one IO in either scheme. The network can transfer up to 100 lists per second you estimate. All other network traffic is negligible. In particular, the cost of transmitting a result is negligible (after we intersect two lists, the number of resulting matches will be very small). Processing times at computers are also negligible. Computers have enough memory to hold the lists for all active queries in main memory.

1. What throughput (queries per second) can each strategy support? For scheme B, you may simplify things by assuming that for every query, two query terms X and Y reside in different machines. Also, assume that the machine where the query originates is different from the machines where query terms X and Y reside. Other reasonable assumptions may be made.
2. Construct a scenario with different number of computers, disk throughput, and/or network throughput where the best strategy of part 1 is no longer the best one.

Problem 2. We are trying to find a good execution plan for the query R NJ S NJ T NJ V (NJ means natural join). The join R NJ S is over attribute A, S NJ T is over B, and T NJ V is over C. Here is the information we have available:

```
Relation R is stored at site 1, and contains 20,000 tuples.
Relation S is stored at site 2, and contains 30,000 tuples.
Relation T is stored at site 3, and contains 40,000 tuples.
Relation V is stored at site 4, and contains 50,000 tuples.
```

All tuples are 100 bytes long. Attributes A, B, C are each 10 bytes long. The final result is desired at site 5.

We model transmission time for Z bytes as 0.01*Z. That is, transmitting 100 bytes takes 1 time unit. We model the computation time of join X NJ Y as 0.0001*(number of tuples in X)*(number of tuples in Y). We do not consider if the inputs to the join are stored on local disk or in local memory. We estimate the number of tuples in the result of X NJ Y to be 0.1*min number of tuples in X, number of tuples in Y. The size of each result tuple is the sum of the tuple sizes of X and Y. We model the execution time for a project P(X, D) of attribute D over relation X as 0.01*(number of tuples in X).

Our system does not start transmitting a relation (input or partial result) until all of the relation is available at the site. Similarly, a site does not start a computation until all of its inputs have been fully received.

Assume there is no contention for the network. However, assume that a site can only perform one task at a time. For instance, if sites 1 and 2 want to send data to site 3, site 3 must first receive all of the data from 1 first, and then all the data from 2 (or viceversa).

- Compute the total execution time of the following plan:
  - Sites 1, 2, 3, 4 send their relations to site 5.
  - site 5 computes the final answer.
- Compute the total execution time of the following plan:
  - Send R to site 2
  - Compute R NJ S at 2; send result to site 3.
  - Compute R NJ S NJ T at 3; send result to site 4.
  - Compute R NJ S NJ T NJ V at 4; send result to site 5.
- Compute the total execution time of the following plan:

  1a Send R to site 2

  2a Compute R NJ S at 2; send result to site 5.

  - While 1a and 2a execute:

    1b Send T to site 4

    2b Compute T NJ V at 4; send result to site 5.

  - When site 5 receives all inputs: Site 5 computes final answer.
- Devise some other query plan that beats all of the above plans. Compute its execution time. Try to make your plan as efficient as possible.

Problem 3.

- Give a simple schedule that can be produced by a two-phase locking scheduler but cannot be generated by a timestamp ordering algorithm, if there exists any.
- Give a simple schedule (history) that can be produced by a timestamp ordering algorithm but cannot be generated by a two-phase locking scheduler, if there exists any.

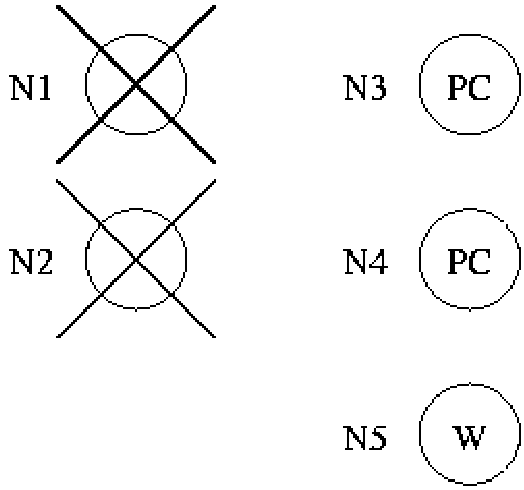

Figure 1

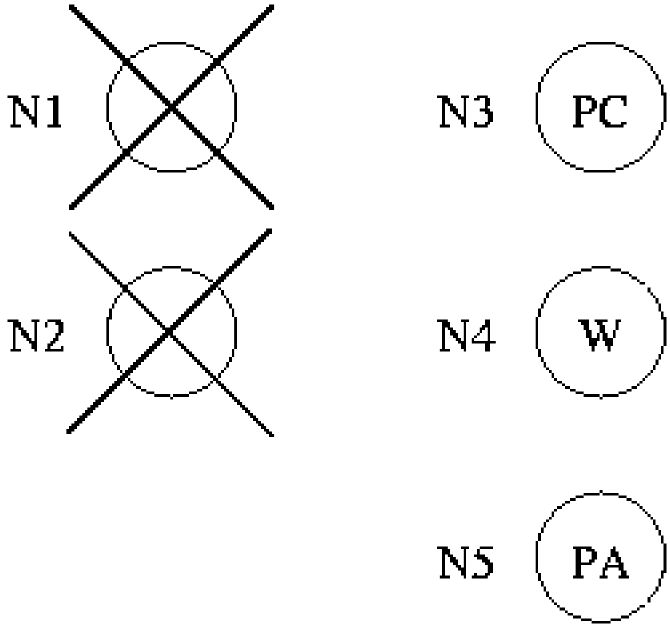

Figure 2

Problem 4. Five computers are cooperating to perform transactions, and they follow majority 3 phase commit.

- While 5 computers were performing transaction T1, two of the computers failed and the remaining three computers were in the states shown in Figure 1. List all possible states that N1 and N2 may have been Can the remaining three computers proceed with transaction T1? If they can, what should be the final outcome of T1? (Note that any of the 5 computers may have failed and recovered before N1 and N2 failed.)
- While the computers were performing transaction T2, two of the computers failed and the remaining three computers were in the states shown in Figure 2. List all possible states that N1 and N2 may have been, before the failure. Can the remaining three computers proceed with transaction T2? If they can, what should be the final outcome of T2?

Problem 5. Consider a system with nodes a, b, c, and d and the following coterie:

{ {a,b}, {a,c}, {a,d}, {b,c,d}}

Show how to implement this coterie with vote assignments. (Votes must be positive integers.)