CS550 Written Assignment 2 (WA#2)

Saptarshi Chatterjee(A20413922)

Chapter 3

1. Would it make sense to limit the number of threads in a server process?

Generally having more than one thread per available hardware thread running at the same time is less efficient. But at times it makes sense to have far more threads created , especially in a server environment where running threads may have to block for i/o frequently. It makes perfect sense to limit no of threads in a server process . And due to ***following limitations*** , instead of creating numerous threads , robust applications uses thread pool

A. Thread local variables stays on their own stack . And creating too many threads may eat up main memory making the system slow.
B. Having too many threads leads to Page Thrashing thus resulting in more IO. Threads might access memory in random pattern , so it often render caches useless , and causes page faults .

2. In this problem you are to compare reading a file using a single-threaded file server and a multithreaded server. It takes 5 msec to get a request for work, dispatch it, and do the rest of the necessary processing, assuming that the data needed are in a cache in main memory. If a disk operation is needed, as is the case one-third of the time, an additional 30 msec is required, during which time the thread sleeps. How many requests/sec can the server handle if it is single threaded? If it is multithreaded?

Single Threaded

Let's say x request can be processed
So, x* (2/3) * 5 + x * (1/3) * 35 = 1000
Or, 100x/3 + 35x/3 = 1000
Or, x =~ 66 (processes)
So in single threaded model server can handle 66 requests

Multi Threaded

x * 5 = 1000 (As no thread will get blocked for disk)
x = 1000/5
X = 200 Processes.
So in multi threaded model server can handle 200 requests

3. Consider a process Q that requires access to file f which is locally available on the machine where Q is currently running. When Q moves to another machine, it still requires access to f. If the file-to-machine binding is fixed, how could the system-wide reference to f be implemented?

As we can't move or copy the file, one possible solution would be creating a new process R that handles all the requests for remotely accessing F. Process Q is offered the same interface to F as before in the form of a proxy. So, new Process R masquerade as a file server, who can access F remotely.

Chapter 5

4. The root node in hierarchical location services may become a potential bottleneck. How can this problem be effectively circumvented?

If root node in a hierarchical location service becomes bottleneck , we can partition the identifier space and install separate root node for each part. We should spread the newly partitioned root node across the network so the access is load-balanced.

Also changes in the Root level is very infrequent so local caching for longer period is another way to avoid this problem.

Also this 2002 paper by Maarten van Steen and Gerco Ballintijn , proposed a Object-to-Server Mapping technique to circumvent the problem.

5. In a hierarchical location service with a depth of x, how many location records need to be updated at most when a mobile entity changes its location?

When mobile entity changes its location the maximum number of location records update happens if it moves from one operation to another and the root is shared between 2 operation . So,

A. We need to delete all the previous x+1 location records resulting x+1 operations
B. And then insert new x+1 location records resulting x+1 operations

Total 2x+1 location records need to be changed at most .

6. High-level name servers in DNS, that is, name servers implementing nodes in the DNS name space that are close to the root, generally do not support recursive name resolution. Can we expect much performance improvement if they did?

There would not be any significant improvement. The high-level name servers constitute the global layer of the DNS name space, Changes to that section of the name space is infrequent.

Consequently, caching will be highly effective, and much long-haul communication will be avoided anyway.
Recursive name resolution for low-level name servers is important, because in that case, name resolution can be kept local at the lower-level domain in which the resolution is taking place.

Chapter 6

7. Consider the behavior of two machines in a distributed system. Both have clocks that are supposed to tick 1500 times per millisecond. One of them actually does, but the other ticks only 1400 times per millisecond. If UTC updates come in once a minute, what is the maximum clock skew that will occur?

Maximum clock skew =
(click of clock A - Click of clock B) * Actual duration * Sync time

[1500-1400] * 0.001 * 60 seconds = 6 seconds.

8. To achieve totally-ordered multicasting with Lamport timestamps, is it strictly necessary that each message is acknowledged?

In case we can assume that the underlying channel does not drop or reorder packets, the successful receipt of an update request from node p to node q can be implicitly deduced when any message with a higher timestamp is received by node p from node q. Under these assumptions (no packet loss & reordering), the receipt of a message with timestamp t by node p from node q guarantees (for node p) that it has received all messages with timestamps lower than t.

Explicit acknowledgements are therefore not strictly needed in this case and it suffices for p to wait for a message with a timestamp greater than t from node q.

9. Many distributed algorithms require the use of a coordinating process. To what extent can such algorithms actually be considered distributed? Discuss.

In a distributed system different processes run on different physical machines. If we have a dedicated coordinator then the co-ordinator machine is generally fixed and it manages the system . E.g. in a hadoop system YARN or Zookeeper generally acts as coordinator.

But if we don't have a dedicated coordinator generally we use an election algorithm to choose one of the participating system as coordinator , and this election is done by other participating processes.

Having a dedicated coordinator rather than dynamically elected one doesn't make a system less distributed.

10. A distributed system may have multiple, independent resources. Imagine that process 0 wants to access resource A and process 1 wants to access resource B. Can Ricart and Agrawalas algorithm lead to deadlocks? Explain your answer.

If process 0 wants to access resource A and doesn't attempt to access any other resource same process 1 wants to access just resource B and doesn't try to access any other resource then the system will be deadlock free.
Problem arises when process 0 gets access to resource A and then it gets the lock , now without releasing the lock it tries to access Resource B , And resource B's lock already acquired by process 1 , who now wants to access resource B then the system will be in deadlock . Ricart and Agrawala's algo helps to determine who should get access to critical section , it doesn't contribute to deadlock , here deadlock occured because the resource access pattern.