CMPSCI 677 Distributed Operating Systems **Spring 2003**

Homework 4 Solutions: May 11

Lecturer: Prof. Prashant Shenoy *TA: Gary Holness*

4.1 Question AST 7.16

In the two-phase commit protocol, the coordinator sends a VOTE_REQUEST message to each participant and waits for a reply (either VOTE_COMMIT or VOTE_ABORT) from each participant. State for the coordinator includes the collected votes. If the coordinator crashes and a new one is elected, the new coordinator needs to know the state of the system in order to proceed correctly. Since processes in the transaction have independent failure modes, there are various bad places where a process can crash in 2PC...

- coordinator crashes right after VOTE_REQUEST
- particpant crashes after receiving VOTE_REQUEST, but before deciding VOTE_COMMIT/VOTE_ABORT outcome.

If we elect a new coordinator, this new coordinator must somehow acquire the state of the old one. Since the old coordinator is unavailable (it crashed after all) it can only try to rebuild state by querrying each participant. Since even a single VOTE_ABORT amidst a set of VOTE_COMMIT responses means that the transaction is to be aborted, we can block. Suppose we have N participants and a coordinator sent a VOTE_REQUEST and then crashed. Suppose all participants received the VOTE_REQUEST and $N-1$ of the participants have decided upon a VOTE_COMMIT and a single participant has crashed before deciding its outcome. The newly elected coordinator can contact each participant to rebuilt the state of the transaction. By the 2PC protocol, the global outcome cannot be decided until the completion status of the transaction has been obtained for all participants. Thus, all participants will be blocked until the completion status of the transaction has settled. Once the crashed participant recovers and returns its vote, only then can the coordinator make its decision. Event if participants contact each other directly, they must wait until the crashed participant recovers and decides upon a VOTE_COMMIT or VOTE_ABORT. Just because a participant votes commit, it doesn't mean that it is to commit the transaction. It must wait for the GLOBAL_COMMIT or GLOBAL_ABORT fromt the coordinator. For a very nice alternate description of 2PC please refer to http://wwws.sun.com/software/jini/specs/core1_2.pdf and look at section TX.2 entitled "The Two-Phase Commit Protocol."

4.2 Question AST 7.20

In a stateless server, there is no information kept at the sever between client requests and all the infromation needed to service a request must be provided by the client. For example imagine a filesystem where requests make are of the form, get me block 12 of file 10 (where file 10 was identified by directory lookup). The server doesn't need to perform checkpointing because it doesn't need to perform any special recovery. It only needs to know what request it was processing. Since all request information is provided by the client, all that is required is to have the client re-do the request. In this case, the "requests in transit" are those

requests made by clients which have not been received. These can be noted down by the client and redone when the server becomes available. If write operations are being performed by the client, the server may need to perform logging.

4.3 Question AST 8.5

The KDC sends the session key for Alice to talk to Bob to the requestor claiming to be Alice. The session key is encrypted with the shared secret key $K_{A,KDC}$. If the recipient is not really Alice, the message cannot be decrypted.

4.4 Question AST 8.6

A nonce is a random number used once and chosen from a large set of numbers. Timestamps have a regularity about them and, thus, loose their randomness. Given that timestamps fall in a predictable range of numbers, this information can be used to crack the key code. Because clocks are synchronized, external processes can affect the value of the "nonce" making it easier to predict a next value.

4.5 Question AST 8.10

Public key encription is more computationally expensive than using symmentric key encription. Rather than pay the cost of public key encription for their communications, we can use the more expensive public key encription to send the session key from Alice to Bob. Henceforth, Alice and Bob use the session key. Note that it is the complexity of the encryption/decription algorithm that makes public key encryption slower.

4.6 Question AST 10.2

A global shared namespace can be mimicked by adopting a system wide consistent mounting strategy. Imagine you have host1.cs.umass.edu, host2.cs.umass.edu ...hostN.cs.umass.edu. Suppose on every host, we use the hostname as the mountpoint under a specific directory (say remote_filesystems). We would have mount points /remote_filesystems/host1, /remote_filesystems/host2, ..., /remote_filesystem/hostN. This can be done either explicitly or implicitly through a daeomon process.

4.7 Question AST 10.8

There were so many different answers for this both in the text an in other sources, reasonable answers were accepted based on your assumptions.

Caching in NFS version 3 has been left outside of the protocol. NFS version 4 implements session semantics. Note that if you adhere to the strict definitions in the text, NFS does not implement entry, nor does it implement release consistency.

If you look at what is happening, you can see it is most similar to release consistency. When a client opens a file, it can cache a local copy on which it can perform reads and writes. When the client closes the file,

if modifications have taken place, the data is flushed back to the server (AST pp596). Even after a file has been closed, data for it can be kept in the cache. If the client re-opens a file that has data which was cached, it must re-validate the cached data. Note that saying something is *similar* is not saying they are identical. Yes, there is a difference because we could have a copy of the file being modified while someone else writes it to the server.

Since the consistency is left up to the application, if you implement locking at the data/record level granularity, you can get entry consistency. If you lock at the file level, you can get release consistency.

(with credit given to Shangzu Wang) If we are considering consistency among the cached copies residing on different clients of the same file. NFS does not implement release consistency. Release consistency requires that shared data are made available on exit. When the client closes the file, NFS requires that modifications have to be flushed back to the server. But it does not require that the new version is propagated to all the clients that have local copies of the file.

NFS does not implement entry consistency. Entry consistency requires that shared data are made available on entering. In NFS, when a client requests a file from the server, and later opens it, the client may not get the latest version of the file. This is because some other client might be performing a write on a local copy of the file and has not released yet.

4.8 Question AST 10.17

Coda handles files using upload and download. Coda can solve read-write conflicts for two cases. If a single writer opens the file for writing, a copy of the file is transferred to the writer, and then all the reader's requests on the file will fail since the server grants the writer exclusive access rights to potentially modify the file. If a reader opens a file for reading, other readers can access the file until a writer opens it for writing. Once the writer opens it, no other reaeders can read the file. Once the writer finnishes, it sends its copy to the server which then sends invalidate messages to all on-going reader sessions. Since Coda treats sessions as transactions, a reader can choose to ignore the invalidate message or cancel their session.