CMPSCI 677 Operating Systems **Spring 2013**

Lecture 18: April 1

Lecturer: Prashant Shenoy *Scribe: Srikar Damaraju*

18.1 Failure Masking by Redundancy

One way to make the system fault tolerant is to introduce redundancy in the system, which means that the system is replicated and input is fed to all copies. Output of all the copies is taken and determine the real output. If one copy is faulty, it will produce a different output than others, and the majority output is considered as the real output. This type of policy can only address one fault in the system.

18.2 Agreement in Faulty Systems

It is always desired to have a k-fault tolerant system, than 0ne-fault tolerant system. This means the system needs at least (k+1) redundant systems to provide k-fault tolerance.

18.3 Byzantine Failures

Systems run even if they're faulty. It means faulty or sick systems run like any other normal system, except that they produce faulty results. These are hard to tolerate. One possible solution is to use "voting". In this technique systems cote on results and determine which one right result. These protocols are also known as consensus protocols, where voting takes place to determine results.

18.4 Byzantine Faults

Consider two processes, which are not faulty, but the network through which they exchange messages is faulty. Theoretically, two systems communicating over an unreliable network can never assure reliable communication.

Now if we assume that network is reliable but one of the systems is faulty, it is possible to detect faults.

18.5 Byzantine General Problem

Consider a situation where there are 4 general and one of them is a traitor, and all of them have to agree upon exchanged information. The way this can be solved is, initially each general broadcasts his information to all other generals. After this round, every general would have received information from all other generals. Then, each general broadcasts the information it has received from all other generals to each other general. After this round, every general has information received by other generals from other generals. Then each

general compares all information, i.e information it has sent, and the information others have sent to him, and the information others have received. Using this information general can determine which general is the traitor.

If we assume that there are only 3 generals and one of them is traitor, it is not possible to implement the above discussed protocol and determine the traitor, because a non-traitor general will always get conflicting information from other two generals, and he cannot determine which one is fault. One thing to notice is, general can determine that there is a traitor, but cannot determine who that traitor is.

18.6 Byzantine Fault Tolerance

Detecting a faulty process is easier, and it takes 2k+1 machines to detect k faults. Whereas to reach an agreement, system needs at least 3k+1 machines, because the system needs at least (2/3)rd majority to eliminate faulty processes. As we can observer, for achieving fault tolerance n Byzantine systems, the overhead of replication is very high, and most systems in practice will not pay such a high price.

18.7 Reliable One-One Communication

One way to achieve reliable one-one communication is to use TCP protocol or handle at the application layer.

18.8 Reliable One-many Communication

To achieve reliable multi-cast, lost messages have to be retransmitted. One way to achieve this is to make systems 'ack' messages. But this involves in exchanging many 'ack' messages, and the sender can become a bottleneck. To solve this, we can make the system send a 'negative ack' when it does not receive a message. But, this only works in sequential transfer of messages.

18.9 Atomic Multicast

The principle of atomic multicast says that the system guarantees that either all nodes receive the multicast message or no node receives the message. It is analogous to "All or Nothing" approach.

Problem One of the main problems of this approach is fault in the system. If a machine fails, then all nodes must revert back to their previous state.

Solution Each message is uniquely associated with a group of processes. Every time the system sends a message, it guarantees that all the processes in that group will receive the message. If, one process crashes, the system has to detect it, and notify all the members of the group about the crashed process, and the rest of the processes form a new group discarding the crashed process. later the system can reboot the crashed process, which can then join the group again.

18.10 Distributed Commit

In distributed commit, the policy is, either everybody commits to the transaction or nobody commits to the transaction.

18.10.1 Two Phase Commit

This policy has a coordinator process which coordinates the operation. Coordinator initiates voting, and if all processes, including the coordinator, agree with that transaction then it is permanently committed, otherwise, even if one process disagrees, that transaction is aborted. One problem with this policy is if coordinator crashes, the system still has to ensure commitment of the transaction. One way to deal with this is, make all processes decide upon the transaction, but the issue with this policy is that processes will not know what vote the coordinator has casted. Unless they taken into account of coordinator's vote, other processes cannot decide upon the transaction.