Assignment (Lec. 15 & 16): Distributed Operating System Principles (COP5615)

Name: Vikas Chaubey**, UFID:** 3511 5826**, Email:** vikas.chaubey@ufl.edu

1) Agreements in Fault Tolerant System: In distributed systems where different processes work together in order to reach a common goal. It is important that all processes reach a common agreement. Processes can easily reach agreement if the system is not faulty in order to do that sites need to know about the values related to other sites. However, when the system is faulty reaching an agreement among the processes is not that easy because the faulty processes can send conflicting values to other processes which can prevent the system to come to an agreement. Non faulty processes might not know about the faulty processes hence in agreement problem the non-faulty processes need to come to an agreement even if certain components are faulty in the system. This could be achieved by certain agreement protocols by regular exchange of messages among processes.

a) **Byzantine Agreement:** Byzantine agreement term is defined from the "Byzantine Generals Problem", which states that in a distributed operating system when a failure occurs then the actors must agree on a strategy in order to keep the system running even if some processes are unreliable. In this agreement all non-faulty processes agree on the same value. In order to do this a source process sends its initial value to all other processes. If the source process is non faulty then the common agreed upon value in all the processes will be the initial value of the source process. All non-faulty processes can agree upon a common value and it's not important that faulty processes agree on a common value or not.
b) **Consensus Agreement:** In the consensus problem all processes transmit their initial value. Initial value of the processes might be different. Eventually all non-faulty processes must agree on a single value. If the non-faulty processors have an initial value, then all processor must have the same agreed upon value. In case if the initial values are different then all processes should come to an agreement on common value. In this case as well it does not matter whether faulty processes come to an agreement or not.
c) **The Interactive Consistency Agreement:** all processes transmit their initial value. Initial value of the processes might be different. All non-faulty processes agree upon a common vector which is a set of initial values of different processes ex. {v-1, v-2, v-3 ….}. So, for example if any process k has initial value as (v-k) then all processes agree on the kth value as (v-k). If the kth process is faulty then all processes can agree on a commo value. In this case as well it does not matter whether faulty processes come to an agreement or not.

2) K- fault Tolerance: if a system handle k faults then it is called k-fault tolerant. In order to achieve a K – fault tolerance in case of silent failures, a system should have k+1 component. Hence in that case if K components fail then the extra component can still keep the system running. In case of Byzantine faults in the system should have a minimum of (2K+1) components.so if k components in the system fail (k+1) components can still work together.to keep the system running.

3) Reliable Multicasting Schemes: Multicasting is a special type of broadcasting in which a sender node send message to multiple nodes in a group. The nodes which want to receive message can join the group and can receive all messages from the sender. Message transmission and reporting feedback are the reliable multicasting schemes used in a distributed system. A message ordering protocol guarantees that when two semantically similar messages are sent to a group one after another, the receiver will deliver the message in the system in same order.

a) **unordered multicast**: There is no guarantee on order of transmission of messages in the system.
b) **FIFO – ordered multicast:** Messages are delivered in first come first serve fashion and in order.
c) **causally ordered multicast:** In this multicast potential causality is preserved. So, if two messages are sent one after another and first message causality precedes the second one then the receiver will always deliver second message after receiving and delivering first message.
d) **Total ordered multicast:** It is always important to deliver the message in the same order as they were received in without concerning about mode of message delivery.

4) Distributed Commit protocols: Commit protocols in distributed systems ensure atomicity that is either all the effects of the transaction executed in the system persist or none of the transactions persist, whether any failures occur or not.

a) **Two Phase Commit Protocol:** In this protocol the commit process is divvied in two phases. One of the process is designated as coordinator and rest processes are called cohorts. Initially the coordinator sends a query for commit to the servers in the system and wait for servers to respond back. In phase 1 all servers write the data records to the logs if the operation is successful then the cohort servers respond with SUCCESS message to coordinator otherwise FAILURE message is sent.in second phase if all processes respond with Success to coordinator, then it writes a commit record into its log and sends a commit message to all the processors. If coordinator does not receive success messages from all processes, then it sends message to all servers to roll back the transaction.

b) **Three Phase Commit Protocol:** This protocol can resolve the blocking problem of the system in two phase-commit.in general two-phase commit protocol make assumption that either no failure occurs or at most k nodes fail in the system. It is an extension of two-phase protocol where in third phase multiple sites are involved to decide a commit. In this protocol if the coordinator intends to commit it first ensures that at least k nodes in the system know about the commit.in case the coordinator fails new coordinator is elected.

5) Recovery After Failure: when a failure occurs in distributed system then the system has to be brought back to error free state. The system could be made error free by bringing the system back to previous error free state which is called backward error recovery. Another way of doing it would be finding a new state in the system from where system can continue. This is called forward error recovery.

6) Checkpointing: Checkpointing in the distributed system helps in handling the system failures. Checkpointing is the process of saving the state of system time to time. In a distributed system different processes save their state by copying all important data in reliable storage time to time. In case, if any system failure occurs then the system process could be rolled back to the most recent checkpoints, that way system does not has to restart from the scratch and lose all the work progress.

References:

1) http://lass.cs.umass.edu/~shenoy/courses/spring08/lectures/Lec17.pdf
2) https://www.cs.uic.edu/~ajayk/Chapter14.pdf
3) https://engineering.purdue.edu/ee695b/public-web/handouts/References/
4) https://medium.com/cosmosgaminghub/fault-tolerance-1800d3093657
5) https://www.cs.rutgers.edu/~pxk/rutgers/notes/content/fault-tolerance-slides.pdf
6) https://www.cis.upenn.edu/~lee/07cis505/Lec/lec-ch8b-mcast-v5.pdf
7) https://www.inf.ed.ac.uk/teaching/courses/ds/slides1516/agreement.pdf
8) https://medium.com/@macworks/message-ordering-16011fce95e1
9) http://courses.cs.vt.edu/~cs5204/fall00/distributedDBMS/duckett/tpcp.html