DC TT2 QUESTION BANK

CHAPTER 3

3.1 Vector clock Algorithm.

- Vector clock is a logical clock, which is used to assign timestamps for events in a distributed system. Vector clock also gives partial ordering of the events.
- unlike lamport's clock which cannot identify concurrent events that are causally related, instead of using integers for the timestamp, vector clock uses a vector of integer values to represent the timestamp.
- If we have N-processes in the group, then each process will have a vector with N elements.

→ Initially all clocks are zero

→ Each time a process experiences an internal event, it increments its own logical clock in the vector by one.

$$VC_i[i] = VC_i[i] + 1$$

→ Each time a process sends a message, it increments its own digital clock in the vector by one and then the message piggybacks a copy of its own vector.

→ Each time a process receives a message, it increments its own logical clock by one and updates each element in the vector by taking the maximum of the value in its own vector clock and the value in the vector in the received msg.

For e.g: If P_j receives a message m from P_i,

$$VC_j = \max(VC_j[k] + 1, VC_i[k]), \forall k$$

3.2 Maekawa Algorithm.

- It is a quorum based approach to ensure mutual enclusion in distributed systems.
- Unlike other permission based algorithms where a site request permission from every other site, in a quorum based approach, a site does not request permission from every other site but from a subset of sites which is called quorum.

WORKING

In this algorithm

→ Three types of messages (REQUEST, REPLY and RELEASE) are used.

→ A site sends a REQUEST message to all other sites in its request set or quorum to get their permission to enter critical section.

→ A site sends a REPLY message to requesting site to give its permission to enter the critical section.

→ A site send a RELEASE message to all other site in its request set or quorum upon exiting the critical section

RULES FOR QUORUM

- Intersection of 2 groups should not be null
- A process should be part of any group
- only a single process cannot form a group. Minimum 2.

ALGORITHM

- To enter critical section :
  → when a site S_i wants to enter critical section, it sends a request message (i) to all other sites in the request set R_i.
  → when a site S_j receives request (i) from S_i, it check for the last RELEASE message. If yes, then it returns a REPLY to S_i otherwise queues it.

- To execute the critical section :
  → A site S_i can enter critical section if it receives a REPLY from all sites in its request set R_i.

- To release the critical section :
  → when S_i exits, it sends RELEASE (i) to R_i
  → when S_j receives RELEASE (i) from S_i, it sends REPLY message to next queued site and removes it from queue.
  → If no queue, S_j updates status to say no REPLY since last RELEASE.

3.3 Bully election algorithm.

- Election algo to choose a coordinate
- It assumes that each process has a unique priority number in the system, so highest priority = coordinator.

ALGORITHM

Lets assume P is a process that sends a message to the coordinator.

1. If no reply within T; assume coordinator has crashed.
2. Now P sends election message to every process with the highest priority number.
3. If no response within T, P elects itself as coordinator.
4. After selecting coordinator, P will send victory message to all lower priority process.
5. If P receives a response from Q:
   - It waits for time T for victory message from Q
   - No response within T; restart algo.

CHAPTER 4.

4.1 Task assignment approach

As the name implies, the task assignment approach is based on the division of the process into multiple tasks. These tasks are assigned to appropriate processors to improve performance and efficiency. This approach has a major setback in that it needs prior knowledge about the features of all the participating processes. Furthermore, it does not take into account the dynamically changing state of the system. This approach's major objective is to allocate tasks of a single process in the best possible manner as it is based on the division of tasks in a system.

ASSUMPTIONS MADE DURING WORKING

- Each task's and processors computing performance is known
- Cost of each task is known
- Inter-process-communication (IPC) cost is known,
- Task is properly split
- Reassignment is not possible.

'm' tasks & 'q' nodes = m×q combination
IRL combo < mq as some restrictions might be there

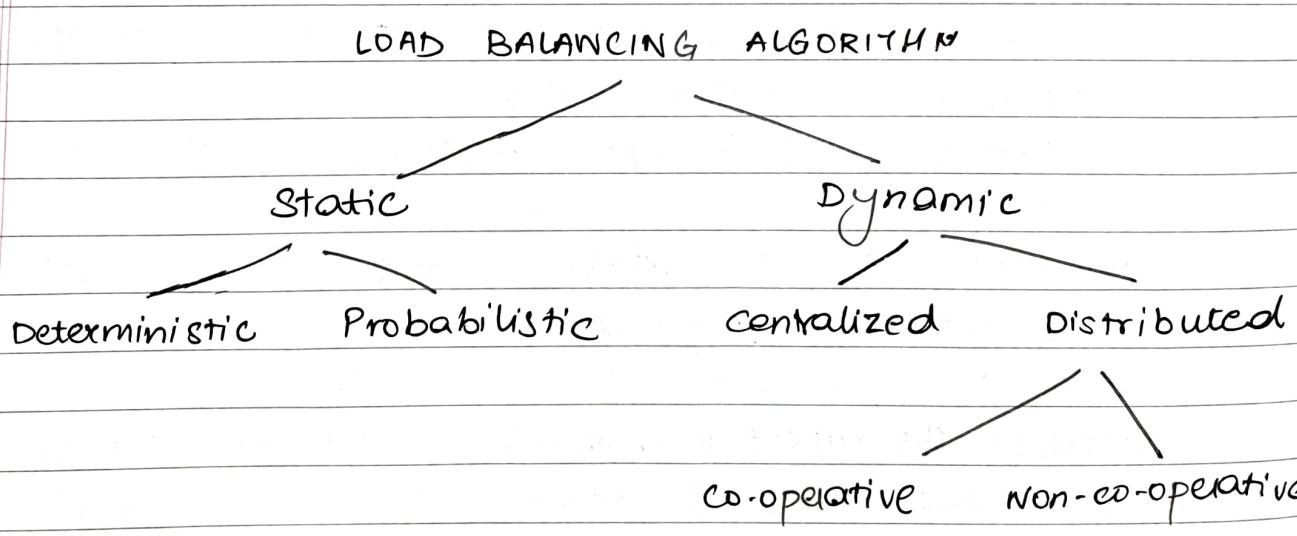
CUTSET: The cutset of a graph refers to the set of edges that when removed make the graph disconnected.

MINIMAL CUTSET: The minimal cutset of a graph refers to the cut which is minimal among all the cuts of the graph.

4.2 Load balancing approach.

In this, the processes are distributed among nodes to equalize the load among all nodes. The scheduling algorithms that use this approach are known as load balancing or load levelling Algorithms.

These algorithms are based on the intuition that for better resource utilization, it is desirable for the load in a distributed system to be balanced evenly. A load balancing algorithm tries to balance the total system load by transparently transferring the workload from heavily loaded nodes to lightly loaded nodes in an attempt to ensure good overall performance.



CHAPTER 5

5.1 Consistent ordering of operations.

Data Centric Consistency Models.

A data store may be physically distributed across multiple machines. Each process that can access data from the store is assumed to have a local or nearby copy available of the entire store.

i) STRICT CONSISTENCY MODEL

- Any read on a data item x returns a value corresponding to the result of the most recent write on x
- This is the strongest form of memory coherence which has the most stringent consistency requirement.
- Strict consistency is the ideal model but is impossible to implement in a distributed system. It is based on absolute global time or a global agreement on commitment of changes.

ii) SEQUENTIAL CONSISTENCY

- Sequential consistency is an important data-centric consistency model which is a slightly weaker consistency model than strict consistency.
- A data store is said to be sequentially consistent if the result of any execution is the same as if the (read & write) operations by all processes on the data store were executed in some sequential order and

the operations of each individual process should appear in this sequence in a specific order.

- Example : Assume 3 operations R1, W1, R2 were performed in an order on a memory address. Then (R1, W1, R2), (R1, W2, R1), (W1, R1, R2), (R2, W1, R1) are acceptable provided all processes see the same ordering.

iii) Linearizability -

- weaker than strict but stronger than sequential.
- A data store is said to be linearizable when each operation is timestamped and the result of any execution is the same as if the (read and write) operations by all processes on the data store were executed in some sequential order.
- The operations of each individual process appear in sequential order specified by its program.
- If $T \times OP_1(x) < T \times OP_2(y)$, then $OP_1(x)$ should precede $OP_2(y)$ in this sequence.

iv) Causal Consistency.

- It is a weaker model than sequential consistency.
- In causal consistency all processes see only those memory reference operations in the correct order that are potentially causally related.
- Memory reference operations which are not related may be seen by different processes in different order.

- A memory reference operation is said to be casually related to another memory reference operation if the first operation is influenced by the second operation.
- If a write (w_2) operation is casually related to another write (w_1) the acceptable order is (w_1, w_2)

v) FIFO consistency.

- It is weaker than causal consistency.
- This model ensures that all write operations performed by a single process are seen by all other processes in the order in which they were performed like a single process in a pipeline.
- This model is simple and easy to implement having good performance because processes are ready in the pipeline.
- Implementation is done by sequencing write operations performed at each node independently of the operations performed on other nodes.
- Example : If P1 does w_{11} and w_{12} and P2 does w_{21} and w_{22}. A process P3 can see them as [(w_{11}, w_{12}), (w_{21}, w_{22})] while P4 can view them as [(w_{21}, w_{22}), (w_{11}, w_{12})]

vi) Weak consistency -

- The basic idea is enforcing consistency on a group of memory reference operations rather than individual operations
- A distributed shared memory system that supports the weak consistency model uses a special variable called a synchronization variable which is used to synchronize memory.
- When a process accesses a sychronization variable, the entire memory is synchronized by making visible the changes made to the memory to all other processes.

vii) Release Consistency

- Release consistency model tells whether a process is entering or exiting from a critical section so that the system performs either of the operations when a synchronization variable is accessed by a process.
- Two synchronization variables acquire and release are used instead of single synchronization variable. Acquire is used when process enters critical section and release is when it exits a critical section
- Release consistency can be viewed as synchronization mechanism based on barriers instead of critical sections.

viii) Entry Consistency

- In entry consistency every shared data item is associated with a synchronization variable
- In order to access consistent data, each synchronization variable must be explicitly acquired.
- Release consistency affects all shared data but entry consistency affects only those shared data associated with a synchronization variable.

Q 5.2 Client-centric consistency models.

client-centric consistency models aim at providing a system wide view on a data store.

→ This model concentrates on consistency from the perspective of a single mobile client.

→ client-centric consistency models are generally used for applications that lack simultaneous updates were most operations involve reading data.

i) EVENTUAL CONSISTENCY

- In systems that tolerate high degree of inconsistency, if no updates take place for a long time all replicas will gradually and eventually become consistent. This form of consistency is called eventual consistency.
- Eventual consistency only requires those updates that guarantee propagation to all replicas.
- Eventual consistent data stores work fine as long as clients always access the same replica.
- Write conflicts are often relatively easy to solve when assuming that only a small group of processes can perform updates. This is cheap to implement.

ii) MONOTONIC READS CONSISTENCY

- A data store is said to provide monotonic-read consistency if a process reads the value of a data item x, any successive read operation on x by that process will always return the same value or a more recent value.
- A process has seen a value of x at time t, it will never see an older version of x at a later time.
- E.g: Each time user requeshes his email, he gets updates from the previously visited server.

iii) MONOTONIC WRITES

- A data store is said to be monotonic-write consistent if a write operation by a process on a data item x is completed before any successive write operation on x by the same process.
- A write operation on a copy of data item x is performed only if that copy has been brought up to date ~~item~~ x by means of any preceding write operations, which may have taken place on other copies of x.
- Enample: Monotonic-write consistency guarantees that if an update is peyormed ~~jixst~~ on a copy of server S, all preceeding updates will be peyormed jixst. The resulting server will then indeed become the most recent version and will include all updates that have led to previous versions of the server.

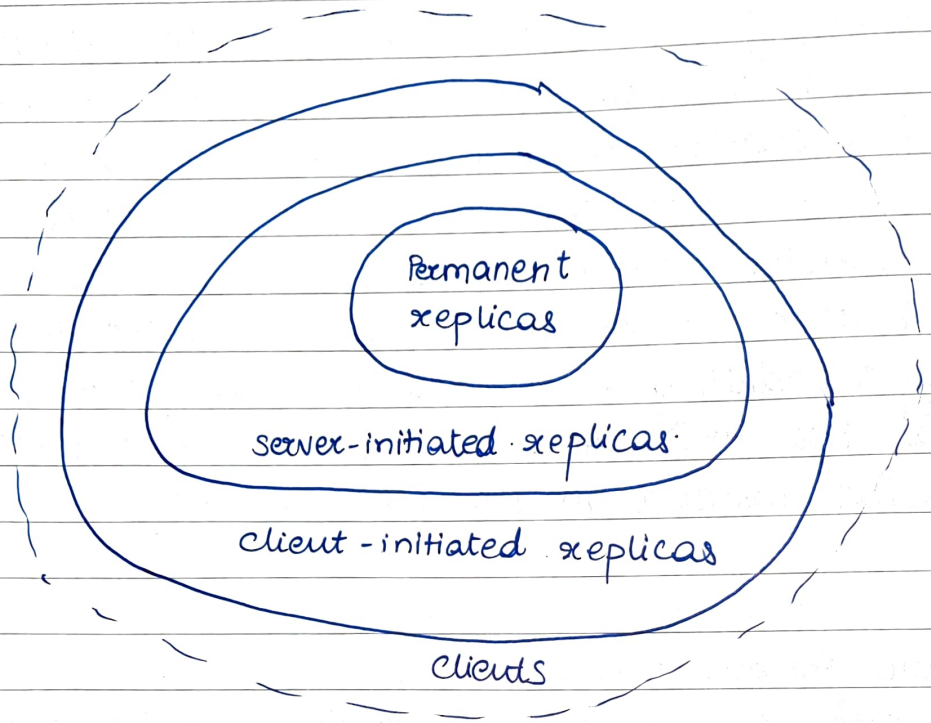
iv) READ YOUR WRITES

- A data store is said to provide read-your-writes consistency if the effect of a write operation by a process on data item x will always be a successive read operation on x by the same process.
- A write operation is always completed before a successive read operation by the same process no matter where that read operation takes place.
- Eg: updating a web page and guaranteeing that the web browser shows the newest version instead of its cached copy.

v) WRITE FOLLOWS READS

- A data store is said to provide write-follows-reads consistency if a process has write operation on a data item x following a previous read operation on x then it is guaranteed to take place on the same or a more recent value of x that was read.
- Any successive write operation by a process on a data item x will be performed on a copy of x that is up to date with the value most recently read by that process.
- E.g: user first reads an article A then posts response B. By following write-follows-read consistency, B will be written to any copy only after A has been written.

5.3 Content replication and placement.



i) PERMANENT REPLICA

Permanent replicas can be considered as the initial set of replicas that constitute a distributed data store. In many cases, the number of permanent replicas is small. Consider for e.g. a website whose distribution generally comes in two forms:

The first kind of distribution is one in which the files that constitute a site are replicated across a limited number of servers at a single location. Whenever a request comes in, it is forwarded to one of the servers using round-robin strategy.

The second form of distributed web sites is what is called mirroring where a web site is copied to a limited number of servers. called mirror sites which are geographically spread across the internet.

ii) SERVER INITIATED REPLICAS

In contrast to permanent replicas, server-initiated replicas are copies of a data store that enist to enhance peyormance and which are ceeated at the initiative of the (owner of the) data store.

consider, jor emample, a web server placed in New York. can handle normal traffic. some days, sudden high traffic comes jrom unemp ected locations from jar away place. In this case, its worth to install a temporary server.

To provide optimal jacilities such hosting services can dynamically replicate jiles to servers where those jiles are needed to enhance peyormance. that is, close to demanding clients.

The algorithm jor dynamic replication takes two issues into account. First, replication can reduce load on a server. Second, specific jiles can be migrated or replicated to servers where man requests are coming from.

iii) CLIENT INITIATED REPLICAS

Also known as client caches. In essence, a cache is a local storage jacility that is used by a client to temporarily store a copy of the data it has just requested. In principle, managing the cache is left entirely to the client.

The data store has nothing to do with keeping cached data consistent. However, as we shall see, there are many occasions in which the client can rely on particip ation jrom the data store to inform it when cached data has become stale. client caches are used only to improve access times to data.

Q.1 File-caching schemes

ANS

1. CACHE LOCATION

(i) Servers main memory : Eliminates disk access cost on a cache hit which increases performance compared to no caching

(ii) Clients disk : Eliminates network access cost but requires disk access cost : slower than cache but is simpler & larger.

(iii) Clients main memory : Eliminates both network cost and disk access cost. Permits workstations to be diskless, reliable & scalable.

2. MODIFICATION PROPAGATION

When the cache is located on client's nodes, a files data may simultaneously be cached on multiple nodes. Possibility for inconsistency.

Techniques used include :-

→ write-through scheme : New value is sent to server to update master copy of the file.

→ Delayed-write scheme : To reduce traffic. New data value is only written to the cache when an entry is modified and all updated cache entries are sent to the server at a later time.

Three approaches ⇒ write on ejection from cache
⇒ Periodic write
⇒ write on close.

3. CACHE VALIDATION SCHEMES

The modification propagation policy only specifies when the master copy of the file on a server node is updated upon modification of a cache entry. It doesn't tell about updation of other nodes. Its necessary to verify client node's data with master copy so that it doesn't become stale.

2 approaches to verify validity of cached data
i) client - initiated approach.
ii) server - initiated approach.

6.2 File - Accessing models.

The specific client's request for accessing a particular file is serviced on the basis of the file accessing model used by the distributed file system.
The file accessing model depends upon :
1) the method used for accessing remote files
2) the unit of data access.

i) ACCESSING REMOTE FILES

a) Remote service model : clients request is processed at the server.
b) Data - caching model : Reduces traffic by caching data received from server. Can have cache coherency problems.

2. UNIT OF DATA TRANSFER

a) File level transfer model

In this, the complete file is moved when a particular operation necessitates file data transfer amongst client & server. scalable and efficient.

b) Block level transfer model

In this, data amongst clients and servers is accomplished in units of file blocks. useful while using diskless workstations.

c) Byte-level transfer model

In this, data is transferred over the network in units of bytes. More flexible compared to other models since it allows retrieval and storage of an arbitrary sequential subrange of a file. Major disadvantage is cache-management because of variable-length data for different access requests.

d) Record-level transfer model

used where the file contents are structured in the form of records. The unit of data transfer in record-level transfer model is record.