Fault-Tolerant Computer Systems

ECE 60872/CS 59000

Replication

Saurabh Bagchi
School of Electrical & Computer Engineering
Purdue University

Basic Idea

- Data is replicated to tolerate failures
- However, it introduces problems of consistency and replica management
- Goal of replica management:
  - Perform operations on the logical data items
  - Underlying system maps it to operations on data replicas
  - Mapping must ensure concurrent execution of actions on replicated data is equivalent to a serial execution of actions on non-replicated data
  - Different copies of data must be in mutually consistent state
- We will study in this topic replica control algorithms, also known as consistency control algorithms

Failure Model

- Two types of failures replica control algorithm must handle
  - Node failures
  - Communication failures
- Node failures
  - Cause some copies of data to become unavailable
  - Replica control algorithm must ensure operations on logical data can be performed, satisfying one copy serializability
- Communication failures
  - Leads to network partitions
  - Replica control algorithm has to restrict processing in the different partitions

Types of Replica Control Algorithms

- Two types of algorithms
  - Optimistic
  - Pessimistic
- Optimistic strategy
  - In case of network partition, no restriction placed on processing in the different partitions
  - Global inconsistencies, if any, are resolved after different partitions merge
- Pessimistic strategy
  - Limit access to data in the different partitions
  - Processing on merging partitions is trivial
  - Three approaches: Primary site, Active replication, and Voting

Optimistic Approach: Version Vector

- Question: How to detect inconsistencies in the partitions when they merge
- Approach: Version vectors [Parker-ToSE83]
- Assumption
  - We are dealing with units of a file
  - Copies of each file are on all nodes
- Each file has a version vector, of size *n* where *n* is the total number of nodes
- Version vector V of a copy of file f represents the number of updates that were performed on this copy
  - At node *i*, for file *f*, version vector is *V*
  - Entry *V*[*j*] (represented as v_j) is the number of updates to *f* from node *j*

Optimistic Approach: Version Vector

- A vector V of a file f is said to *dominate* another vector V′ of the file (at another node) if the following condition holds
  - $v_i \geq v_i', \forall i = 1, \ldots, n$
- When two partitions merge, the version vectors are compared one by one for each file
- For file *f*, if version vector of partition 1 (say V_1) dominates over that of partition 2 (say V_2)
- Then, copy the file with vector V_1 onto the file with vector V_2
- If the version vectors are in conflict, then manual intervention is needed

Pessimistic Approach: Primary Site

- For every data item, there is a primary site and there are multiple backup sites
- For k-resilient data, 1 primary site and k backup sites
- Requests for all operations (read or write) are sent to the primary
- If operation is read
  - Primary site performs the read and returns result to client
- If operation is update
  - Primary site sends request to at least k backups
  - When all backups have received request, then primary performs the update
  - All the backups perform the received update operation
  - FIFO reliable broadcast used by primary
  - Alternately, primary can take checkpoints periodically and send to the backups

Pessimistic Approach: Primary Site – Failure Cases

- If primary fails
  - Election happens among the backups
  - The new primary processes all update operations forwarded by the previous primary
  - Then it starts accepting new user requests
- If network partitions
  - First, a node has to be able to distinguish between node failure and network partition
  - Only the partition which contains the primary can function

Pessimistic Approach: Active Replicas

- In primary site approach, backups are passive
- Here, all replicas are active
- One approach for active replicas is state machine approach [Schneider-ACMSurveys90]
- Failure model: fail-stop failures of nodes that have the data copies
- For k-resiliency, data replicated on ______ nodes
- Request sent to all the replicas
- Any replica can service a request
- Two key requirements: *agreement* and *order*
- These can be satisfied by the atomic broadcast algorithm that we have studied

Pessimistic Approach: Voting

- Performing an operation on replicated data is determined collectively by replicas through voting
- Voting methods do not require a node to distinguish between node failures and network partitions
- Two kinds of voting methods
  - *Static methods*: The vote assignment and quorum requirements do not change with time
  - *Dynamic methods*: Vote assignment, number of copies, etc. may change with time

Static Voting Methods: Weighted Voting

- Weighted voting approach from [Gifford-SoSP79]
- Each replica of the data has a version number
  - The version number is incremented whenever a write occurs
- To read data: Acquire at least *r* votes from the nodes storing copies of the data – *Read quorum*
- To write data: Acquire at least *w* votes from the nodes storing copies of the data – *Write quorum*
- Let the total number of votes be v
- Then the following two conditions must be satisfied
  1. $r + w > v$
  2. ___________

Weighted Voting: How to Perform Read or Write

- To perform read or write, a node broadcasts a request for votes to all the nodes
- Each node which receives this request, replies with
  - Version number of its replica
  - Number of votes the node has
- The requester collects votes until it has enough votes to meet the quorum corresponding to the operation (read or write)
- The requester can then perform the operation
  - For read, it takes the value with the highest version number
  - For write, it reads the value with the highest version number, performs the update, and then writes the latest value to all the quorum members

Weighted Voting: Failure Scenarios

- What happens if the network partitions into two?
- If multiple partitions occur, then any of the following situations may arise
  - One group has read and write quorum
  - Several groups have read quorum, none has write quorum
  - No group has even a read quorum
- Two sample vote assignments:
  - r = 1, w = ?
  - $r = w = \lceil v/2 \rceil$

Hierarchical Voting: Basics

- With weighted majority voting, the number of votes that must be collected increases linearly with the number of nodes
- Hierarchical voting
  - (+) Number of votes that must be collected grows slowly
  - (–) Multiple rounds of voting are required
- Set of nodes is logically organized as a tree
- Physical copies of data placed at the leaves – level *m*
- Higher level nodes correspond to logical groups within which quorum will be established
- Number of children of a node at level *i* is l_{i+1}
  - The single root node is at level 0

Hierarchical Voting: Forming Quorums

- A quorum is associated with each level
- Read quorum at level i: How many of the l_i nodes must be included in the quorum for each level i-1 node that is included in the level i-1 quorum
- Quorum at level 1 implies quorum collection at all levels right down to the leaf level m
- A quorum consensus algorithm is shown to be correct if
  1. $r_i + w_i > l_i$, for all levels $l = 1, 2, \ldots, m$
  2. $w_i > \lceil l_i/2 \rceil$, for all levels $l = 1, 2, \ldots, m$
- For a read operation, ______ physical copies of data will be required in read quorum
- For a write operation, ______ physical copies of data will be required in write quorum

Hierarchical Voting: Improvement over Majority Voting

- Given n nodes, what is the height of the tree in terms of l_i?
- Say l_i =3. How many copies have to be read to form a read quorum if w_i = 2?
- How does this compare with the majority voting?

Dynamic Voting

- Static voting methods do not adapt to changes in the system due to failures
  – If due to repeated failures, small partitions are formed, no partition may be able to perform updates
- Dynamic voting solves the problem due to repeated partitioning
- We will study scheme by Jajodia *et al.* in SIGMOD 81
- Assumption: Each site has one vote
- Logical data *d*, with multiple replicas: replica is denoted as d_i for node *i*
- For data replica d_i
  – Version number VN_i
  – Update sites cardinality SC_i

Dynamic Voting: Update

- Current version number of data *d (VN)*: $\underset{i}{Max}(VN_i)$
- Replica d_i is current if $VN_i = VN$
- Majority partition: If the partition contains the majority of the *latest copies* of *d*
- Basic idea: A node can perform update if it belongs to the majority partition
- Steps in update:
  1. A node 1 wants to do an update and sends a request.
  2. It hears responses from nodes 2, …, *m*.
  3. Find maximum version number from among these responses, and including its own version number, say *M*.
  4. Find set of nodes with maximum version number, say *I*.
  5. Find maximum SC of nodes in *I*, say *N*.
  6. Do an update if |I| > _____ (fill in the blank)

Dynamic Voting: Update

- If node 1 can perform update
  1. Updates the data item d_1 and asks 2, …, *m* to update d_2, …, d_m.
  2. $VN_j = M+1, \forall j = 1, \ldots, m$
  3. $SC_j = m, \forall j = 1, \ldots, m$

- Dynamic voting will allow updates in partitions that do not form the majority of total nodes
- Once it allows operations in such a group, it must ensure that no other group can perform the operations without including any node from this group

Dynamic Voting: Catching Up

- After some partitions merge, a node *i* realizes it does not have the current version of the data
- It has to catch up and update its state
- Node can only update its state if it belongs to the majority partition
- Steps to update the state:
  1. Node 1 wants to catch up and sends a request.
  2. It hears responses from nodes 2, …, *m*.
  3. Find maximum version number from among these responses, and including its own version number, say *M*.
  4. Find set of nodes with maximum version number, say *I*.
  5. Find maximum SC of nodes in *I*, say *N*.

Dynamic Voting: Catching Up

- If node 1 can update its state
  1. It gets state from a node with the current copy, i.e., a node whose version number = M
  2. $VN_1 = M$
  3. $SC_1 = N$
- Example with five nodes A, B, C, D, E in the network
  - Under what condition can an update happen with majority voting?

Vote Assignment and Reassignment

- Do we believe in the rule of equal votes for each node?
- Consider the case of 4 nodes: a, b, c, d
- We are using majority voting for updates
- Case 1: All nodes have one vote
- The following scenarios will allow updates to continue

{a, b, c, d}; {a, b, c}; {b, c, d}; {c, d, a}; {a, b, d}

- Case 2: Node a has two votes, all other nodes have one vote
- The following scenarios will allow updates to continue

All the scenarios of case 1 + {a, b}; {a, c}; {a, d}

- What is the lesson about assignment of votes to nodes?

Vote Assignment and Reassignment

- Goal: Allow operations to continue as partitions become more and more fragmented
- Solution approach: Dynamically reassign votes
- Approach 1: Overthrow technique
  - One node in the majority group supplants the loss of each node x that is partitioned from the majority group
  - Example: A single node x has been partitioned from the rest
  - In the majority group, node a has been designated to take over the votes of x
  - After overthrow, a has votes v(a) + 2 v(x)
- Approach 2: Alliance technique
  - Distribute the increase 2 v(x) equally among the remaining nodes in the majority partition

Degree of Replication

- Degree of replication = Number of replicas
- As degree of replication ↑, data availability ↑ because more number of replicas are available
- But, also as degree of replication ↑, checkpointing overhead ↑ and recovery overhead ↑
- Availability = (1-unavailability due to all replicas failing) (1-unavailability due to recovery of a replica) *
  (1-unavailability due to checkpointing)

Degree of Replication: Primary Site Approach

- Notation: N: # replicas, time between failures of a replica = $1/f$, time for checkpointing = $1/h = b \times N$, service time for operation on data $1/\mu$, inter-arrival time for requests $1/\lambda$, rate of recovery r ($r >> f$)
- First term: Unavailability due to all replicas failing
- Probability p_F = $\sum_{k=0}^{N} \left(\frac{\delta}{f}\right)^k \frac{1}{k!}$
- Second term: Unavailability due to recovery
- Probability p_R = $f\left(\frac{\lambda\sqrt{Nb}}{2\mu\kappa}\right)$, where $\kappa = \sqrt{\lambda f / 2\mu}$
- Third term: Unavailability due to checkpointing
- Probability p_C = $\kappa\sqrt{Nb}$
- Availability $A = (1-p_F)(1-p_R)(1-p_C)$
- Since A is non-monotonic with N, an optimal N can be found

References

- Pankaj Jalote, “Fault Tolerance in Distributed Systems” Chapter 7: Data Replication and Resiliency

Optimistic Approach: Precedence Graph

- Version vectors cannot detect read-write conflicts
- Precedence graph approach [Davidson-ToDS84]: Both reads and writes are logged
- Within each partition, some transaction concurrency control is in place
  - Enforces, transactions within a partition are serializable
  - Let the serialization order within partition *i* be T_i1, T_i2, …, T_in
- When partitions merge, a precedence graph is formed
  - Within partition *i*, edge $T_ij \rightarrow T_ik$ if
  - *a)* T_ik reads an item produced by T_ij
  - *b)* T_ij read an item that was later modified by T_ik
  - Across partitions *i* and *l*, edge $T_ij \rightarrow T_lk$ if
  - *a)* T_ij has read a value written by T_lk
- What is the condition for no conflict between transactions in different partitions?