What is Leader Election in Distributed Systems?

The goal of leader election is to give a specific entity (such as a process, host, thread, object, or person) special powers within a distributed system. These powers may include the ability to delegate tasks, the ability to modify data, or the responsibility for handling all system requests. Leader election can be a useful tool for improving efficiency, minimizing coordination, simplifying architectures, and reducing overhead, but it can also introduce additional failure modes and scaling challenges and make it more difficult to assess the validity of the system.

To achieve this goal, a leader election algorithm selects a processor to coordinate the actions of a distributed system. The leader is typically chosen based on a criterion, such as selecting the processor with the highest identifier. The other processors enter a terminated state when the leader is chosen. In a leader election algorithm, the terminated states are divided into **elected** and **non-elected** states. Once a processor enters an elected or non-elected state, it remains in that state at all times.

The safety and liveness conditions must be met for a leader election algorithm to be effective. The **liveness condition** states that every processor will eventually enter either an elected or non-elected state. The **safety condition** states that only one processor is allowed to enter the elected state and become the leader of the distributed network.

Use cases of leader election

There are three scenarios to consider when determining if a leader election is appropriate:

1. **When each node is roughly the same and there is no clear contender for a permanently assigned leader:** In this case, any node can be voted as the leader of the system, and there is no single

point of failure.

2. **When the cluster is performing a complex task that requires close collaboration:** Coordination can involve dividing the work, assigning work to specific nodes, and combining the results from different nodes. For example, in a scientific computation to determine how a protein folds, a leader node may be needed to allocate each node to a specific part of the computation and then combine the results to obtain the full folded protein configuration.
3. **When the system executes many distributed writes to data and requires good consistency:** Consistency means that no matter which node processes a request, the user will always have the most recent version of the data. In this case, a leader ensures consistency by being the source of truth about the system's current state, and the leader election algorithm must maintain this properly. For example, a bank may require strong consistency to ensure that a user's bank account balance is accurate regardless of which server responds to their online banking request and that multiple transactions on the same bank account do not conflict with each other.

Advantages and disadvantages of leader election

Leader election is a common pattern in distributed systems because it has several benefits:

- It is easier for humans to think about systems with a single leader, as it consolidates all of the system's concurrency, reduces the risk of partial failures, and provides a single location for logs and metrics.
- A single leader can be more efficient, as it can simply inform other systems about changes rather than needing to form consensus on them.
- A single leader can easily provide consistency to clients by observing and controlling all changes to the system's state.
- A single leader can improve performance or reduce costs by providing a single consistent cache of data that can be used simultaneously.

- It may be simpler to write software for a single leader than for a quorum, as the single leader does not need to consider other systems that may be working in the same state simultaneously.

There are also some significant drawbacks to leader election:

- A single leader creates a single point of failure, as the entire system may become unavailable if the system fails to detect or correct a faulty leader.
- A single leader limits the scale of the system in terms of data size and request rate, and a complete re-architecture may be required if the system needs to expand beyond a single leader.
- A leader represents a single point of trust, as a faulty leader can quickly spread problems throughout the system and have a large impact (referred to as a "blast radius").
- Leader-elected systems may be difficult to implement partial deployments, which are used in many Amazon software safety policies, such as one-box, A/B testing, blue/green deployment, and incremental deployment with automatic rollback.

Leader Election Algorithms

A leader election algorithm guides a cluster to jointly agree on one node to serve as a leader with as few back-and-forth interactions as possible. Generally, the algorithm assigns each node one of three states: Leader, Follower, or Candidate. The leader is also required to pass a "health check" or "heartbeat" regularly so that follower nodes can determine if the leader has become unavailable or failed, and a new leader can be elected.

The type of leader election mechanism used depends on whether the cluster is synchronous or asynchronous. In a synchronous cluster, nodes are synchronized to the same clock and send messages in a predictable order. In an asynchronous cluster, messages are not reliably sent within a specific timeframe or in any particular order.

Asynchronous algorithms cannot guarantee both safety (ensuring that only one leader is elected) and liveness (ensuring that every node completes the election) because any number of nodes in an asynchronous cluster can lag indefinitely. In practice, implementations prioritize safety because it has more serious consequences for the service.

Synchronous algorithms are easier to understand and may be preferable because they can guarantee both safety and liveness. However, synchronizing a cluster requires imposing additional constraints on how it operates, which may not be possible or scalable in practice.

Bully Algorithm

The Bully Algorithm is a simple synchronous leader election technique that requires each node to have a unique numeric id and for all nodes in the cluster to know each other's ids. When a node starts up or the current leader fails the health check, the election process begins.

There are two possible outcomes:

1. If a node has the highest id, it declares itself the winner and sends this message to the other nodes.
2. If a node has a lower id, it sends messages to all nodes with higher ids. If it does not receive a response, it assumes that these nodes have failed or are unavailable and declares itself the winner.

Paxos

Paxos is a general consensus protocol that can be used for asynchronous leader elections. The idea behind the Paxos algorithm is to reach consensus among the nodes in the network about which node should be the leader. In this algorithm, a node proposes itself as the leader and other nodes in the network vote for or against it. If majority of nodes vote for the node, it becomes the leader. If not, the node that failed to become the leader tries again. The process continues until a node receives a majority vote and becomes the leader.

RAFT

Raft is a popular alternative to Paxos because it is easier to understand, implement, and use. It is a non-blocking algorithm that involves each node in the Raft consensus keeping track of the current "election term." When the leader election begins, each node increments its copy of the term number and listens for messages from other nodes. If the node does not receive any messages after a random interval, it becomes a candidate leader and asks other nodes for votes.

If the candidate wins a majority of votes, it becomes the leader. If another candidate with a higher term number sends a message, the original candidate concedes. If the election is split or the timer runs out without a consensus, the algorithm restarts. Restarts are rare due to random timeouts, which prevent nodes from colliding.

Apache ZooKeeper (ZAB)

Apache Zookeeper is a centralized coordination service that is "self-distributed and highly dependable." It is designed to help distributed systems handle coordination tasks. Apache Zookeeper philosophy is that coordination is difficult, so it is better to have a shared, open-source implementation with all the necessary components so that services do not have to reinvent everything from scratch. This can be especially useful in large distributed systems.

Apache Zookeeper uses the ZAB (ZooKeeper Atomic Broadcast) protocol to manage leader election, replication order guarantees, and node recovery. ZAB is an asynchronous algorithm that ensures that writes are consistent and propagated to all nodes by "broadcasting" state changes from the leader to followers.

Conclusion

Leader election is a strong technique that may be employed in systems to help make them more fault-tolerant and easier to use. However, when we employ leader election, we pay close attention to the guarantees that each protocol gives and, more critically, does not give.

References

https://aws.amazon.com/builders-library/leader-election-in-distributed-systems/

Thanks to Navtosh for his contribution in creating the first version of this content. If you have any queries/doubts/feedback, please write us at contact@enjoyalgorithms.com. Enjoy learning, Enjoy system design, Enjoy algorithms!