igotanoffer.com

Leader election: system design interview (5 of 9)

15-20 minutes

If you want to succeed in system design interviews for a software engineering role, then you'll probably need to understand leader election and when (or if) to use it in the context of a larger system.

Leader election is an important concept for distributed systems. And it can have a significant impact on the speed, scalability, and consistency of those systems.

In the article below, we'll provide an overview of leader election, how it works, and the pros and cons of using it. Let's jump in!

1. Leader election basics
2. Leader election algorithms
3. Alternatives to leader election
4. Example leader election questions
5. System design interview preparation

1. Leader Election Basics

1.1 Definition

Sometimes horizontally scaling a system is as simple as spinning up a cluster of nodes and letting each node respond to whatever

subset of the incoming requests they receive. At other times, the task at hand requires more precise coordination between the nodes and it's helpful to have a **leader** node directing what the **follower** nodes work on.

A **leader election algorithm** describes how a cluster of nodes without a leader can communicate with each other to choose exactly one of themselves to become the leader. The algorithm is executed whenever the cluster starts or when the leader node goes down.

1.2 When to use

There are three cases to consider when deciding if leader election fits the situation.

The first case is when each node is roughly the same and there isn't a clear candidate for a permanently assigned leader. This means any node can be elected as leader, and there isn't a single point of failure required to coordinate the system.

The second case is when the cluster is doing particularly complex work that needs good coordination. Coordination can mean anything from decisions about how the work is to be divided, to assigning work to specific nodes, or to synthesizing the results of work from different nodes.

Let's consider for example a scientific computation that is trying to determine how a protein folds. Because there are so many possible solutions, this computation can take a long time and will be sped up considerably if it's distributed. The cluster will need a leader node to assign each node to work on a different part of the computation, and then add the results together to get the complete

folded protein configuration.

The third case where leader election adds value is when a system executes many distributed writes to data and requires **strong consistency**. You can read more about consistency in our article on Databases, but essentially this means it's very important that no matter what node handles a request the user will always have the most up-to-date version of the data. In this situation a leader creates consistency guarantees by being the source of truth on what the most recent state of the system is (and the leader election algorithm must preserve this properly).

Not all applications require strong consistency, but you can imagine how it might be important to a bank to ensure that no matter what server answers a user's online banking request their bank account total will be accurate, and that multiple transactions directed to the same bank account won't conflict with each other.

1.3 Drawbacks

The main downside to leader election is complexity: a bad implementation can end up with “split brain” where two leaders try to control at the same time, or no leader is elected and the cluster can’t coordinate. As such, leader election should only be used when there is a need for complex coordination or strong consistency, and none of the alternatives fit the situation.

A leader is a single node, so it can become a bottleneck or temporary single point of failure. Additionally, if the leader starts making bad decisions (whatever that means in the context of directing work for the service), the followers will just do what they're assigned, possibly derailing the entire cluster.

The leader / follower model generally makes the best practices of partial deployment and A/B testing harder by requiring the whole cluster to follow the same protocols or be able to respond uniformly to the same leader.

Now that we've gone over the benefits and downsides of leader election, and you know when it's appropriate to use, let's jump into the algorithm approaches for implementing it!

2. Leader Election Algorithms

A **leader election algorithm** guides a cluster to collectively agree on one node to act as leader with as few back and forth communications as possible.

Generally, the algorithm works by assigning one of three states to each node: Leader, Follower, or Candidate. Additionally the leader will be required to regularly pass a **"healthcheck"** or **“heartbeat”** so follower nodes can tell if the leader has become unavailable or failed and a new one needs to be elected.

The kind of leader election algorithm you want depends on whether the cluster is synchronous or asynchronous. In a **synchronous** cluster nodes are synchronized to the same clock and send messages in predictable amounts of time and ordering. In an **asynchronous** cluster messages are not reliably delivered within a certain amount of time or in any order.

In an asynchronous cluster any number of nodes can lag indefinitely so the leader election process can't guarantee both **safety** - that no more than one leader will be elected - and **liveness** - that every node will finish the election. In practice, implementations choose to guarantee safety because it has more

critical implications for the service.

Synchronous algorithms can guarantee both safety and liveness, and are therefore easier to reason about and theoretically preferable. But in practice the big drawback is synchronizing a cluster requires implementing additional constraints on how the cluster operates that aren't always feasible or scalable.

Now let's take a deeper look at the four most popular leader election algorithms.

2.1 Bully Algorithm

The **Bully Algorithm** is a simple *synchronous* leader election algorithm. This algorithm requires that each nodes has a unique numeric id, and that nodes know the ids of all other nodes in the cluster.

The election process starts when a node starts up or when the current leader fails the healthcheck. There are two cases:

1. if the node has the highest id, it declares itself the winner and sends this message to the rest of the nodes.
2. if the node has a lower id, it messages all nodes with higher ids and if it doesn't get a response, it assumes all of them have failed or are unavailable, and declares itself the winner.

The main downside of the bully algorithm is that if the highest-ranked node goes down frequently, it will re-claim leadership every time it comes back online, causing unnecessary reelections. Synchronization of messages can also be difficult to maintain, especially as the cluster gets larger and physically distributed.

2.2 Paxos

Paxos is a general consensus protocol that can be used for asynchronous leader election. Quite a lot of research has been done about the Paxos family of algorithms, which means it's both robust and there's much more to say about it than we have space for in this article.

You don't need to know all the details of Paxos for designing systems, in fact for leader election generally it's best to choose an existing implementation because of the complexity. It's unlikely that your system will have a feature constraint that isn't already covered by an existing open source library or service.

Very briefly, Paxos uses state machine replication to model the distributed system, and then chooses a leader by having some nodes propose a leader, and some nodes accept proposals. When a **quorum** of (enough of) the accepting nodes choose the same proposed leader, that proposed leader becomes the actual leader.

2.3 RAFT

Raft is an alternative to Paxos that is favored because people tend to find it simpler to understand, and therefore easier to implement and use. Raft is an asynchronous algorithm.

In Raft consensus, each node keeps track of the current "election term". When leader election starts each node increments its copy of the term number and listens for messages from other nodes. After a random interval, if the node doesn't hear anything, it will become a candidate leader and ask other nodes for votes.

If the candidate ever reaches a majority of votes, it becomes a leader, and if it ever receives a message from another candidate with a higher term number, it concedes. The algorithm restarts if

the election is split or times out without consensus. Restarts don't happen too often because the random timeouts help make it so nodes don't usually conflict.

2.4 Apache ZooKeeper (ZAB)

Apache Zookeeper is a centralized coordination service that is "itself distributed and highly reliable." The ethos behind Apache ZooKeeper is that coordination in distributed systems is difficult, and it's better to have a shared open source implementation with all the key elements so that your service doesn't have to reimplement everything from scratch. This is especially helpful in large distributed systems.

ZAB (ZooKeeper Atomic Broadcast) is the protocol used by Apache ZooKeeper to handle leader election, replication order guarantees, and node recovery. It is called this because the leader "broadcasts" state changes to followers to make sure writes are consistent and propagated to all nodes. ZAB is an asynchronous algorithm.

ZAB is focused on making sure the history of the cluster is accurate through leadership transitions. The leader is chosen such that it has the most up to date history (it has seen the most recent transaction). When enough of the nodes agree that the new leader has the most up to date history, it syncs history with the cluster and finishes the election by recording itself as leader.

3. Alternatives to leader election

Alternatives to leader election are based on the premise that coordination is possible without a dedicated leader node, thus

achieving the primary function of leader election with lower implementation complexity.

Here's a brief overview of three of the most notable alternatives:

3.1 Locking

A locking model ensures that concurrent operations on a shared resource don't conflict by only allowing changes from one node at a time. With **optimistic locking** a node will read a resource and its version id, make changes, and then before updating make sure that the version id is the same. If the id is different this means the resource has been updated since the node first read it. Going forward with the intended changes based on the old id would lose the other changes, so the node needs to try again.

In **pessimistic locking** a node locks the resource, makes changes, and then unlocks the resource. If another node tries to initiate a change while the resource is locked, it will fail and try again later. Pessimistic locking is more rigorous, but can be hard to implement and bugs can cause **deadlocks** that stop a system from functioning.

These locking patterns are named for use cases. Optimistic locking is useful when you can make the "optimistic" assumption that another node won't change the resource out from under the operation. And pessimistic locking is useful when you can make the "pessimistic" assumption that there will be contention for the resource.

3.2 Idempotent APIs

APIs can have the feature of **idempotency** to ensure consistent

interactions with a shared resource. An API is idempotent when the same request sent multiple times will not produce any inconsistent results. When reading from a resource, this means the response will always be the same value. When writing, this means the update will only happen once.

For example, idempotent *writes* can be implemented by requiring request ids so the system can tell if a request is being retried. Idempotency is also supported by other features we've talked about, like locking and database transactions.

An intuitive example of an idempotent API is bank account transfers: if a user initiates an online bank transfer and their internet goes down halfway through processing, you want to make sure the user can initiate the transfer again and your system will correctly only transfer the amount *once*.

3.3 Workflow Engines

Another way of coordinating nodes in a system is by using a **workflow engine** . A workflow engine is a centralized decision making system that contains a set of "workflows" of what work can be done, the state of data and work in the system, and the resources available to assign work to. Popular solutions are AWS Step Functions, Apache Airflow, and .NET State Machines

4. Example leader election questions

The questions asked in system design interviews tend to begin with a broad problem or goal, so it’s unlikely that you’ll get an interview question specifically about leader election.

However, you may be asked to solve a problem where leader

election will be an important part of the solution. As a result, what you really need to know is WHEN you should bring it up and how you should approach it.

To help you with this, we've compiled the below list of example system design scenarios, where leader election is relevant. These could be a helpful illustration for how leader election fits into a broader system design.

- Implement Leader Election with Kubernetes (Read the answer)
- Design a Distributed Message Queue (Read the answer)
- Implement Leader Election in Google Cloud (Read the answer)

5. System design interview preparation

Leader election can be an important part of a system. But to succeed on system design interviews, you'll also need to familiarize yourself with several other concepts. And you'll need to practice how you communicate your answers.

It's best to take a systematic approach to make the most of your preparation time, and we recommend the steps below. For extra tips, take a look at our article: 19 system design interview tips from FAANG ex-interviewers.

5.1 Learn the concepts

There is a base level of knowledge required to be able to speak intelligently about system design. To help you get this foundational knowledge (or to refresh your memory), we've published a full series of articles like this one, which cover the primary concepts that you'll need to know:

- Network protocols and proxies
- Databases
- Latency, throughput, and availability
- Load balancing
- Leader election
- Caching
- Sharding
- Polling, SSE, and WebSockets
- Queues and pub-sub

We'd encourage you to begin your preparation by reviewing the above concepts and by studying our system design interview prep guide, which covers a step-by-step method for answering system design questions. Once you're familiar with the basics, you should begin practicing with example questions.

5.2 Practice by yourself or with peers

Next, you'll want to get some practice with system design questions. You can start with the examples listed above, or with our list of 31 example questions.

We'd recommend that you start by interviewing yourself out loud. You should play both the role of the interviewer and the candidate, asking and answering questions. This will help you develop your communication skills and your process for breaking down questions.

We would also strongly recommend that you practice solving system design questions with a peer interviewing you. A great

place to start is to practice with friends or family if you can. If you don't have anyone in your network who can interview you, then you might want to check out our our system design mock interview peer group.

5.3 Practice with ex-interviewers

Practicing with peers can be a great help, and it's usually free. But, at some point you'll start noticing that the feedback you are getting from peers isn't helping you that much anymore. Once you reach that stage, we recommend practicing with ex-interviewers from top tech companies.

If you know someone who has experience running interviews at Facebook, Google, or another big tech company, then that's fantastic. But for most of us, it's tough to find the right connections to make this happen. And it might also be difficult to practice multiple hours with that person unless you know them really well.

Here's the good news. We've already made the connections for you. We've created a coaching service where you can practice system design interviews 1-on-1 with ex-interviewers from leading tech companies. Learn more and start scheduling sessions today.

Learn more about system design interviews

This is just one of 9 concept guides that we've published about system design interviews. Check out all of our system design articles on our Tech blog.