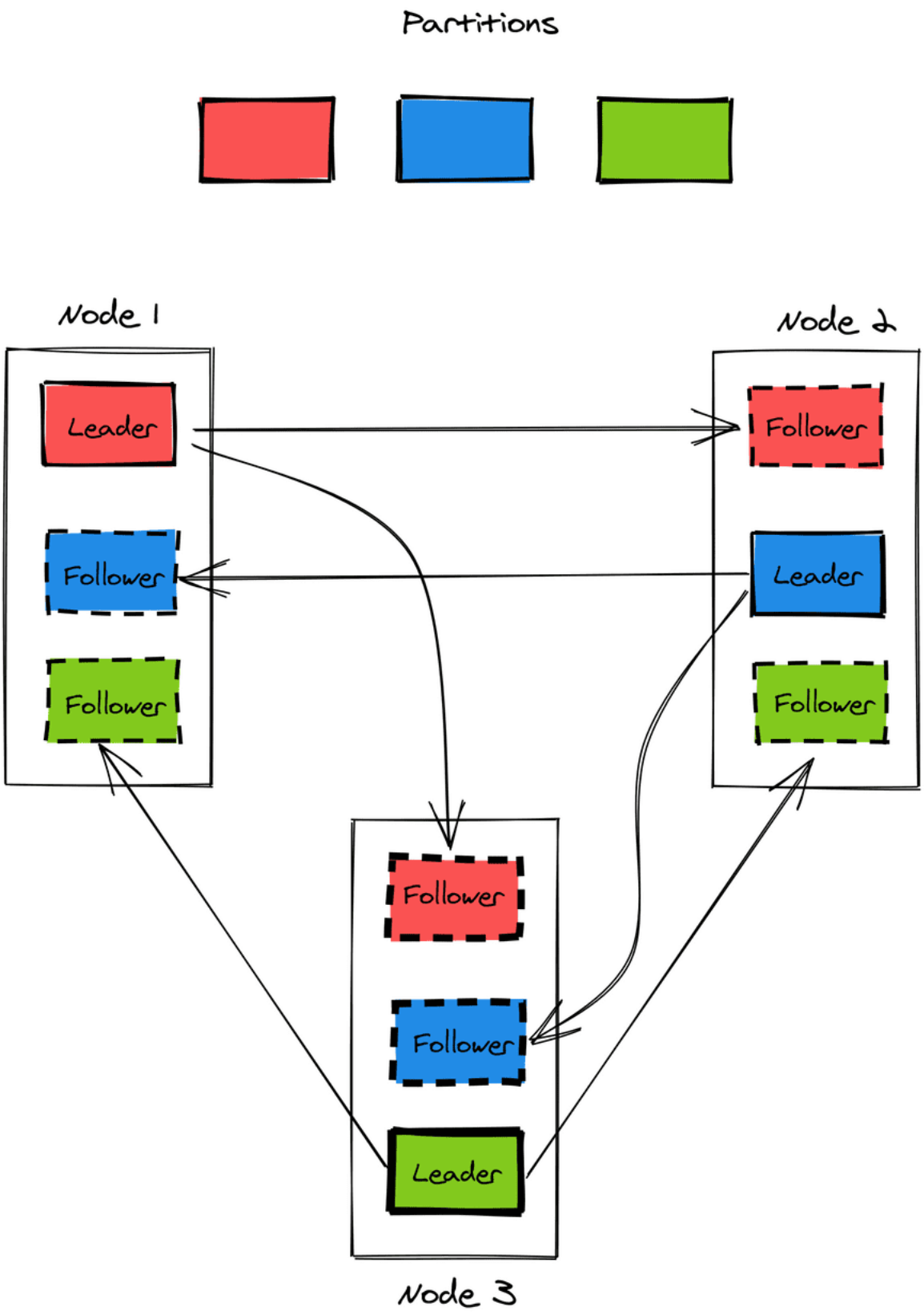
Scalability patterns of distributed systems

November 24, 2020

I have released a new chapter of Understanding Distributed Systems! It explores the different patterns at your disposal when designing horizontally scalable applications.



The simplest way to scale an application is by running it on more expensive hardware. But, that only brings you so far as the application will eventually reach a performance ceiling. The alternative to scaling up is scaling out by distributing the load over

multiple nodes.

The chapter presents three independent categories of scalability patterns that can be combined within the same application: functional decomposition, partitioning, and duplication.

Functional Decomposition

Functional decomposition is the process of taking an application and breaking it down into individual parts. Think of the last time you wrote some code - you most likely decomposed it into functions, classes, and modules. The same idea can be taken further by decomposing an application into separate services, each with its well-defined responsibility. Decomposing an application into services doesn't come for free, though, and the chapter talks about the costs of doing so.

The chapter also dives into patterns that become crucial once your application is split into multiple services, such as using an API gateway to shield external consumers from the internal APIs and leveraging asynchronous messaging to improve the overall system's availability.

Duplication

Arguably the easiest way to add more capacity to a service is to create more instances of it and have some way of routing - or balancing - requests to them. The thinking is that if one instance has a certain capacity to handle load, then two instances should have a capacity that is twice that. Creating more service instances can be a fast and cheap way to scale out a stateless service, as long as you have taken into account the impact this can have on the service's dependencies.

Scaling out a stateful service is significantly more challenging as coordination comes into play to replicate the state among the instances. The chapter introduces the concept of load balancing by discussing the implementation of L4 and L7 load

balancers. Then, various approaches to state replication are explored, like single-leader, multi-leader, and leaderless. Finally, the chapter explores how to implement caching and content distribution at scale.

Partitioning

When a dataset no longer fits a single node, it needs to be partitioned - or sharded - across multiple nodes. Sharding is a general technique that can be used in a variety of circumstances. The chapter discusses different sharding strategies and their relative trade-offs and explores how to rebalance partitions when new nodes are added to the system.

P.S.

Thank you for all the feedback you have been sending me over the past months! I have rewritten parts of previously released chapters based on it. I plan to keep updating the book over time, just like software, so that every time you go back to it, you will find an improved version of what you initially read.

Written by Roberto Vitillo

First Name

Email Address

Subscribe

I respect your privacy. Unsubscribe at any time.

← The costs of microservices

How distributed systems fail →