Part II: Distributed Data

There are various reasons w hy you might w ant to distribute a database across multiple machines:

- Scalability
  If your data volume, read load, or w rite load grow s bigger than a single machine can handle, you can spread the load across multiple machines.

- Fault tolerance/high availability
  If one machine goes dow n, you can use multiple machines to give you redundancy. When one fails, another can take over.

- Latency
  You might w ant to have servers at various locations w orldw ide so that each user can be served from a datacenter that is geographically close to them.

Scaling to Higher Load

The simplest approach is to buy a more pow erful machine (scaling up / vertical scaling). Many CPUs, many RAM chips, and many disks can be joined together under one OS, and a fast interconnect allow s any CPU to access any part of the memory / disk. In this *shared-memory architecture*, all the components can be treated as a single machine.

The problem w ith this approach is that the cost grow s faster than linearly. A shared-memory architecture may offer limited fault tolerance - high end machines have hot-sw appable components, but it is definitely limited to a single geographic location.

Another approach is the *shared-disk architecture*, w hich uses several machines w ith independent CPUs and RAM, but stores data on an array of disks that is shared betw een the machines, w hich are connected via a fast netw ork. This architecture is used for some data w arehousing w orkloads, but contention and the overhead of locking limit the scalability of the shared-disk approach.

Shared-Nothing Architectures

Also know n as *horizontal scaling* or *scaling out*. In this approach, each machine or virtual machine running the database softw are is called a *node*. Each node uses its CPUs, RAM, and disks independently. Any coordination betw een nodes is done at the softw are level, using a conventional netw ork.

No special hardw are is required, so you can use w hatever machines w ith the best price/performance ratio. You can potentially distribute data across multiple geographic regions, and thus reduce latency for users and potentially be able to survive the loss of an entire datacenter. With cloud deployments of virtual machines, even for small companies, a multi-region distributed architecture is now feasible.

We w ill focus on shared-nothing architectures because they require the most caution from the application developer. If your data is distributed across multiple nodes, you need to be aw are of the constraints and trade-offs that occur in such a distributed system - the database cannot magically hide these from you.

While a distributed shared-nothing architecture has many advantages, it usually also incurs additional complexity for applications and sometimes limits the expressiveness of the data models you can use.

Replication vs. Partitioning

Tw o common w ays data is distributed across multiple nodes:

1. Replication
   Keeping a copy of the same data on several different nodes, potentially in different locations. Replication provides redundancy: if some nodes are unavailable, the data can still be served from the remaining nodes. Replication can also help improve performance. (Discussed in Chapter 5)

2. Partitioning
   Splitting a big database into smaller subsets called *partitions* so that different partitions can be assigned to different nodes (also know n as *sharding*). (Discussed in Chapter 6)

![](resources/Figure II-1.png)

With an understanding of these concepts, w e can discuss the difficult trade-offs that you need to make in a distributed system.

We'll discuss transactions in Chapter 7, as that w ill help you understand all the many things that can go w rong in a data system, and w hat you can do about them. We'll conclude this part of the book by discussing the fundamental limitations of distributed systems in Chapters 8 and 9.