PART II
Distributed Data

For a successful technology, reality must take precedence over public relations, for nature cannot be fooled.

—Richard Feynman, *Rogers Commission Report* (1986)

In Part I of this book, we discussed aspects of data systems that apply when data is stored on a single machine. Now, in Part II, we move up a level and ask: what happens if multiple machines are involved in storage and retrieval of data?

There are various reasons why you might want to distribute a database across multiple machines:

Scalability
: If your data volume, read load, or write load grows bigger than a single machine can handle, you can potentially spread the load across multiple machines.

Fault tolerance/high availability
: If your application needs to continue working even if one machine (or several machines, or the network, or an entire datacenter) goes down, you can use multiple machines to give you redundancy. When one fails, another one can take over.

Latency
: If you have users around the world, you might want to have servers at various locations worldwide so that each user can be served from a datacenter that is geographically close to them. That avoids the users having to wait for network packets to travel halfway around the world.

Scaling to Higher Load

If all you need is to scale to higher load, the simplest approach is to buy a more powerful machine (sometimes called *vertical scaling* or *scaling up*). Many CPUs, many RAM chips, and many disks can be joined together under one operating system, and a fast interconnect allows any CPU to access any part of the memory or disk. In this kind of *shared-memory architecture*, all the components can be treated as a single machine [1].[i]

The problem with a shared-memory approach is that the cost grows faster than linearly: a machine with twice as many CPUs, twice as much RAM, and twice as much disk capacity as another typically costs significantly more than twice as much. And due to bottlenecks, a machine twice the size cannot necessarily handle twice the load.

A shared-memory architecture may offer limited fault tolerance—high-end machines have hot-swappable components (you can replace disks, memory modules, and even CPUs without shutting down the machines)—but it is definitely limited to a single geographic location.

Another approach is the *shared-disk architecture*, which uses several machines with independent CPUs and RAM, but stores data on an array of disks that is shared between the machines, which are connected via a fast network.[ii] This architecture is used for some data warehousing workloads, but contention and the overhead of locking limit the scalability of the shared-disk approach [2].

Shared-Nothing Architectures

By contrast, *shared-nothing architectures* [3] (sometimes called *horizontal scaling* or *scaling out*) have gained a lot of popularity. In this approach, each machine or virtual machine running the database software is called a *node*. Each node uses its CPUs, RAM, and disks independently. Any coordination between nodes is done at the software level, using a conventional network.

No special hardware is required by a shared-nothing system, so you can use whatever machines have 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

[i] In a large machine, although any CPU can access any part of memory, some banks of memory are closer to one CPU than to others (this is called *nonuniform memory access*, or NUMA [1]). To make efficient use of this architecture, the processing needs to be broken down so that each CPU mostly accesses memory that is nearby—which means that partitioning is still required, even when ostensibly running on one machine.

[ii] *Network Attached Storage* (NAS) or *Storage Area Network* (SAN).

machines, you don't need to be operating at Google scale: even for small companies, a multi-region distributed architecture is now feasible.

In this part of the book, we focus on shared-nothing architectures—not because they are necessarily the best choice for every use case, but rather because they require the most caution from you, the application developer. If your data is distributed across multiple nodes, you need to be aware 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. In some cases, a simple single-threaded program can perform significantly better than a cluster with over 100 CPU cores [4]. On the other hand, shared-nothing systems can be very powerful. The next few chapters go into details on the issues that arise when data is distributed.

Replication Versus Partitioning

There are two common ways data is distributed across multiple nodes:

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. We discuss replication in Chapter 5.

Partitioning

Splitting a big database into smaller subsets called *partitions* so that different partitions can be assigned to different nodes (also known as *sharding*). We discuss partitioning in Chapter 6.

These are separate mechanisms, but they often go hand in hand, as illustrated in Figure II-1.

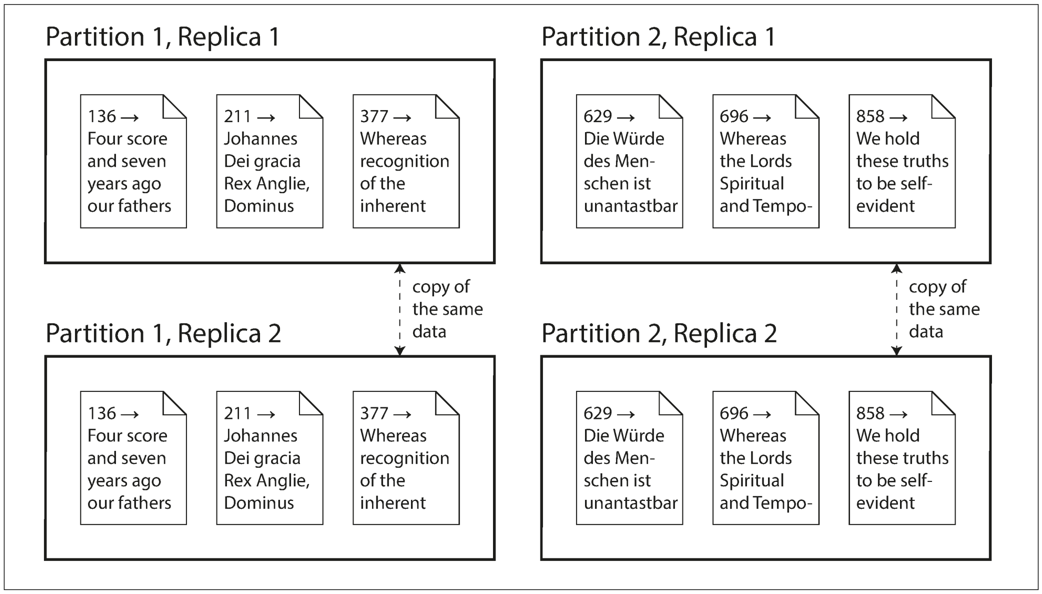
Partition 1, Replica 1
136 → Four score and seven years ago our fathers
211 → Johannes Dei gracia Rex Anglie, Dominus
377 → Whereas recognition of the inherent
Partition 2, Replica 1
629 → Die Würde des Menschen ist unantastbar
696 → Whereas the Lords Spiritual and Tempo-
858 → We hold these truths to be self-evident
copy of the same data
copy of the same data
Partition 1, Replica 2
136 → Four score and seven years ago our fathers
211 → Johannes Dei gracia Rex Anglie, Dominus
377 → Whereas recognition of the inherent
Partition 2, Replica 2
629 → Die Würde des Menschen ist unantastbar
696 → Whereas the Lords Spiritual and Tempo-
858 → We hold these truths to be self-evident

Figure II-1. A database split into two partitions, with two replicas per partition.

With an understanding of those concepts, we can discuss the difficult trade-offs that you need to make in a distributed system. We'll discuss *transactions* in Chapter 7, as that will help you understand all the many things that can go wrong in a data system, and what 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.

Later, in Part III of this book, we will discuss how you can take several (potentially distributed) datastores and integrate them into a larger system, satisfying the needs of a complex application. But first, let's talk about distributed data.

References

[1] Ulrich Drepper: "What Every Programmer Should Know About Memory," *akkadia.org*, November 21, 2007.

[2] Ben Stopford: "Shared Nothing vs. Shared Disk Architectures: An Independent View," *benstopford.com*, November 24, 2009.

[3] Michael Stonebraker: "The Case for Shared Nothing," *IEEE Database Engineering Bulletin*, volume 9, number 1, pages 4–9, March 1986.

[4] Frank McSherry, Michael Isard, and Derek G. Murray: "Scalability! But at What COST?," at *15th USENIX Workshop on Hot Topics in Operating Systems* (HotOS), May 2015.