1.3 Understanding distributed systems and Hadoop

Moore's law suited us well for the past decades, but building bigger and bigger servers is no longer necessarily the best solution to large-scale problems. An alternative that has gained popularity is to tie together many low-end/commodity machines together as a single functional *distributed system*.

To understand the popularity of distributed systems (scale-out) vis-à-vis huge monolithic servers (scale-up), consider the price performance of current I/O technology. A high-end machine with four I/O channels each having a throughput of 100 MB/sec will require three hours to *read* a 4 TB data set! With Hadoop, this same data set will be divided into smaller (typically 64 MB) blocks that are spread among many machines in the cluster via the Hadoop Distributed File System (HDFS). With a modest degree of replication, the cluster machines can read the data set in parallel and provide a much higher throughput. And such a cluster of commodity machines turns out to be cheaper than one high-end server!

The preceding explanation showcases the efficacy of Hadoop relative to monolithic systems. Now let's compare Hadoop to other architectures for distributed systems. SETI@home, where screensavers around the globe assist in the search for extraterrestrial life, represents one well-known approach. In SETI@home, a central server stores radio signals from space and serves them out over the internet to client desktop machines to look for anomalous signs. This approach moves the data to where computation will take place (the desktop screensavers). After the computation, the resulting data is moved back for storage.

Hadoop differs from schemes such as SETI@home in its philosophy toward data. SETI@home requires repeat transmissions of data between clients and servers. This works fine for computationally intensive work, but for data-intensive processing, the size of data becomes too large to be moved around easily. Hadoop focuses on moving code to data instead of vice versa. Referring to figure 1.1 again, we see both the data and the computation exist within the Hadoop cluster. The clients send only the MapReduce programs to be executed, and these programs are usually small (often in kilobytes). More importantly, the move-code-to-data philosophy applies within the Hadoop cluster itself. Data is broken up and distributed across the cluster, and as much as possible, computation on a piece of data takes place on the same machine where that piece of data resides.

This move-code-to-data philosophy makes sense for the type of data-intensive processing Hadoop is designed for. The programs to run ("code") are orders of magnitude smaller than the data and are easier to move around. Also, it takes more time to move data across a network than to apply the computation to it. Let the data remain where it is and move the executable code to its hosting machine.

Now that you know how Hadoop fits into the design of distributed systems, let's see how it compares to data processing systems, which usually means SQL databases.