8.2 Alternative Computing Paradigms

Streaming algorithms [3] represent an alternative programming model for dealing with large volumes of data with limited computational and storage resources. This model assumes that data are presented to the algorithm as one or more *streams* of inputs that are processed in order, and only once. The model is agnostic with respect to the source of these streams, which could be files in a distributed file system, but more interestingly, data from an "external" source or some other data gathering device. Stream processing is very attractive for working with time-series data (news feeds, tweets, sensor readings, etc.), which is difficult in MapReduce (once again, given its batch-oriented design). Furthermore, since streaming algorithms are comparatively simple (because there is only so much that can be done with a particular training instance), they can often take advantage of modern GPUs, which have a large number of (relatively simple) functional units [104]. In the context of text processing, streaming algorithms have been applied to language modeling [90], translation modeling [89], and detecting the first mention of news event in a stream [121].

The idea of stream processing has been generalized in the Dryad framework as arbitrary dataflow graphs [75, 159]. A Dryad job is a directed acyclic graph where each vertex represents developer-specified computations and edges rep-

resent data channels that capture dependencies. The dataflow graph is a logical computation graph that is automatically mapped onto physical resources by the framework. At runtime, channels are used to transport partial results between vertices, and can be realized using files, TCP pipes, or shared memory.

Another system worth mentioning is Pregel [98], which implements a programming model inspired by Valiant's Bulk Synchronous Parallel (BSP) model [148]. Pregel was specifically designed for large-scale graph algorithms, but unfortunately there are few published details at present. However, a longer description is anticipated in a forthcoming paper [99].

What is the significance of these developments? The power of MapReduce derives from providing an abstraction that allows developers to harness the power of large clusters. As anyone who has taken an introductory computer science course would know, abstractions manage complexity by hiding details and presenting well-defined behaviors to users of those abstractions. This process makes certain tasks easier, but others more difficult, if not impossible. MapReduce is certainly no exception to this generalization, and one of the goals of this book has been to give the reader a better understanding of what's easy to do in MapReduce and what its limitations are. But of course, this begs the obvious question: What other abstractions are available in the massively-distributed datacenter environment? Are there more appropriate computational models that would allow us to tackle classes of problems that are difficult for MapReduce?

Dryad and Pregel are alternative answers to these questions. They share in providing an abstraction for large-scale distributed computations, separating the *what* from the *how* of computation and isolating the developer from the details of concurrent programming. They differ, however, in how distributed computations are conceptualized: functional-style programming, arbitrary dataflows, or BSP. These conceptions represent different tradeoffs between simplicity and expressivity: for example, Dryad is more flexible than MapReduce, and in fact, MapReduce can be trivially implemented in Dryad. However, it remains unclear, at least at present, which approach is more appropriate for different classes of applications. Looking forward, we can certainly expect the development of new models and a better understanding of existing ones. MapReduce is not the end, and perhaps not even the best. It is merely the first of many approaches to harness large-scaled distributed computing resources.

Even within the Hadoop/MapReduce ecosystem, we have already observed the development of alternative approaches for expressing distributed computations. For example, there is a proposal to add a third *merge* phase after map and reduce to better support relational operations [36]. Pig [114], which was inspired by Google's Sawzall [122], can be described as a data analytics platform that provides a lightweight scripting language for manipulating large datasets. Although Pig scripts (in a language called *Pig Latin*) are ultimately converted into Hadoop jobs by Pig's execution engine, constructs in the language allow developers to specify data transformations (filtering, joining, grouping, etc.) at a much higher level. Similarly, Hive [68], another open-source project, provides

an abstraction on top of Hadoop that allows users to issue SQL queries against large relational datasets stored in HDFS. Hive queries (in HiveQL) "compile down" to Hadoop jobs by the Hive query engine. Therefore, the system provides a data analysis tool for users who are already comfortable with relational databases, while simultaneously taking advantage of Hadoop's data processing capabilities.