Many programming models and frameworks have been introduced to abstract away the management details of running applications in distributed environments. MapReduce is regarded as a proven deliverer that solves computation problems using trivial-time resources. It provides a simple programming model and powerful runtime system for processing large datasets. The model is based on two key abstractions: “map” and “reduce,” and this runtime system automatically parallelizes input data and schedules the execution of programs across a large cluster of commodity machines. MapReduce has been applied to diverse processing problems such as distributed indexing, sorting, and clustering.

Swift is a portable programming tool for rapid and effective execution, evaluation, and management of large-scale scientific and engineering workflows, built on top of a concise scripting language called SwiftScript for specifications of complex parallel computations based on data type and iterations, and dynamic dataset mappings. The runtime system is based on Ception middleware engine for scheduling and data balancing, and integrates the Swift light-weight (task execution service) for optimized task throughput, resource provisioning, and to leverage data locality found in application access patterns.

Applications that can be implemented in MapReduce are a subset of those that can be implemented in Swift. That said to the more generic programming model found in Swift. Contrasting Swift and Hadoop are interesting as it could potentially attract new users and applications to systems which traditionally were not considered.

We compared two benchmarks, Sort and WordCount, and tested them at different scales and with different datasets. The testbed consisted of a 2.2GHz quad-core processor running Ubuntu Linux, the MapReduce implementation from Yahoo! was configured to use an on-disk File System HDFS, while Swift used Global File System GFS.

Results

We found that Swift offered comparable performance with Hadoop, a surprising finding due to the choice of benchmarks which favored the MapReduce model. In testing over a range of input in large files, Swift execution times were on average 20-30% higher when compared to Hadoop. However, for WordCount, Swift execution times were an average 60% lower. Our experience with Swift and Hadoop indicates that the file system and middleware for each needs to be considered when comparing applications. HDFS is a very scalable file system, but its out-of-box problems with small files, and a response API makes it difficult. There are current efforts in Hadoop to enable Swift to operate over local data stores rather than shared file systems and to cache data across jobs, which would be free offers comparable scalability and performance to HDFS without the added complexity of configuring applications. We do plan to do additional ex- periments on the file system with large files and data sets, as well as additional benchmarks.

Conclusions

We conclude with these observations:

1. Can MapReduce applications run on our workflow system? We believe yes, and with even better performance in some cases.
2. Is the MapReduce model an option for scientific applications?
3. What parallel programming model will be best suited for scientific applications in the coming decades? We hope future work will help answer questions 2 and 3.

Related material & Acknowledgments