Apache Hadoop* Community Spotlight
Apache* MapReduce

Devaraj Das explains Apache* MapReduce, a powerful model for parallel processing large data sets—and the heart of your Apache Hadoop* system. Devaraj is a project management committee member of the Apache Hadoop project, a Committer for the Apache Hadoop Distributed File System (HDFS*), MapReduce, and HCatalog, and a contributor to the Apache HBase* database program. He’s also a cofounder and member of the technical staff at Hortonworks.

Apache MapReduce: Deceptively Simple

Apache MapReduce is a programming model that enables the massive scalability required to process large data sets in parallel. First implemented at Google, MapReduce is now a core part of the open-source Apache Hadoop framework along with HDFS and Hadoop* Common, the utilities layer. Other Apache Hadoop system components, such as the Apache Pig* data flow language and Apache Hive* data warehouse infrastructure, use MapReduce to provide front-end translation of data sets before processing.

MapReduce programs—or jobs—run on hundreds, even thousands, of servers in a Hadoop cluster and perform two basic tasks:

• **Map.** Large sets of data in HDFS are transformed into sets of key and value pairs and equally distributed on the cluster.

• **Reduce.** The results of the map task become input for the reduce task and are combined into a smaller set of key and value pairs to create the final output.

When MapReduce is paired with HDFS across clusters that provide high aggregate I/O, large data sets can be processed efficiently and with high throughput.

**Master-Slave Architecture**

MapReduce works on the basis of master-slave architecture. The master is the JobTracker, which runs on a single node or server. The slaves are TaskTrackers, which run on the remaining nodes in the system. A client submits a job to the MapReduce framework. JobTracker breaks it into tasks and sets up the data structures required to run the job in parallel across the cluster. Work is allocated to the nearest available TaskTracker, which maps the data input and “shuffles” it to a specific reduce task where output is generated. This “data locality” is a key design criterion of the MapReduce framework. It enables efficient throughput of very large data sets by storing and processing the data at the same location.

---

“This ‘data locality’ is a key design criterion of the MapReduce framework. It enables efficient throughput of very large data sets by storing and processing the data at the same location.”

—Devaraj Das

**Keeping Big Jobs Moving:**

**Fault Tolerance**

Apache* MapReduce is designed with fault-tolerance capabilities. In large clusters, nodes can be expected to fail periodically. When task processes fail on one node, MapReduce automatically re-executes them on another node. Jobs can even continue in cases where there are major cluster load losses.
Apache Hadoop includes Capacity Scheduler, a scheduling tool to enable MapReduce to run as a shared multitenant cluster, maximizing throughput as well as utilization. Capacity Scheduler enables organizations to share cluster resources, giving each tenant a minimum capacity guarantee. This makes it possible to partition available resources, with each tenant helping to fund the resources based on their specific computing needs. A tenant can access excess capacity not in use by others for cost-effective elasticity.

Sharing clusters across organizations requires strong support for multitenancy to ensure that each organization has their guaranteed capacity available to them. The Capacity Scheduler makes this possible by defining a stringent set of limits to functions, such as the maximum number of jobs initialized at any point in time. This ensures fairness and stability across the cluster so that no single job, user, or queue can consume a disproportionate amount of resources, especially the all-important JobTracker resource. Jobs are also protected by isolating them from one another in sandboxes that confine tasks to a directory structure that limits where those jobs can read and write data. This prevents one job from disrupting another.

Sharing clusters across organizations requires strong support for multitenancy to ensure that each organization has their guaranteed capacity available to them. The Capacity Scheduler makes this possible by defining a stringent set of limits to functions, such as the maximum number of jobs initialized at any point in time. This ensures fairness and stability across the cluster so that no single job, user, or queue can consume a disproportionate amount of resources, especially the all-important JobTracker resource. Jobs are also protected by isolating them from one another in sandboxes that confine tasks to a directory structure that limits where those jobs can read and write data. This prevents one job from disrupting another.

**Running Jobs in Isolation in a Multitenant Environment**

Apache Hadoop includes Capacity Scheduler, a scheduling tool to enable MapReduce to run as a shared multitenant cluster, maximizing throughput as well as utilization. Capacity Scheduler enables organizations to share cluster resources, giving each tenant a minimum capacity guarantee. This makes it possible to partition available resources, with each tenant helping to fund the resources based on their specific computing needs. A tenant can access excess capacity not in use by others for cost-effective elasticity.

Sharing clusters across organizations requires strong support for multitenancy to ensure that each organization has their guaranteed capacity available to them. The Capacity Scheduler makes this possible by defining a stringent set of limits to functions, such as the maximum number of jobs initialized at any point in time. This ensures fairness and stability across the cluster so that no single job, user, or queue can consume a disproportionate amount of resources, especially the all-important JobTracker resource. Jobs are also protected by isolating them from one another in sandboxes that confine tasks to a directory structure that limits where those jobs can read and write data. This prevents one job from disrupting another.

**How Many Jobs Is Too Many?**

Too many jobs running at once in the Apache Hadoop* system can lower overall cluster utilization and result in bottlenecks. This is a design issue with JobTracker, with known limits of 40,000 tasks in parallel over 4,000 nodes.

While this may seem like an inconceivable number of servers in a cluster, with a huge number of concurrent tasks, predictions for data growth in the near future mean MapReduce scalability limitations could become a problem. Apache Hadoop 0.23 now contains Apache Hadoop YARN to address this limitation as well as other issues.

**Running Jobs in Isolation in a Multitenant Environment**

Apache Hadoop includes Capacity Scheduler, a scheduling tool to enable MapReduce to run as a shared multitenant cluster, maximizing throughput as well as utilization. Capacity Scheduler enables organizations to share cluster resources, giving each tenant a minimum capacity guarantee. This makes it possible to partition available resources, with each tenant helping to fund the resources based on their specific computing needs. A tenant can access excess capacity not in use by others for cost-effective elasticity.

Sharing clusters across organizations requires strong support for multitenancy to ensure that each organization has their guaranteed capacity available to them. The Capacity Scheduler makes this possible by defining a stringent set of limits to functions, such as the maximum number of jobs initialized at any point in time. This ensures fairness and stability across the cluster so that no single job, user, or queue can consume a disproportionate amount of resources, especially the all-important JobTracker resource. Jobs are also protected by isolating them from one another in sandboxes that confine tasks to a directory structure that limits where those jobs can read and write data. This prevents one job from disrupting another.

**MapReduce Gets a Makeover: Meet Apache YARN**

Apache YARN is a new approach to resource management that significantly boosts the scalability in MapReduce. With YARN, Apache Hadoop systems can support computational clusters of up to 10,000 nodes and 200,000 cores. YARN also provides improved reliability and cluster utilization.

YARN provides a separation of resource management and application life-cycle management. MapReduce becomes a user space application, one of many potential computing paradigms that YARN can deploy. With this approach, you can implement any version of MapReduce safely on the same cluster with other versions. Plus, this opens up new opportunities to deploy other computing models in the future.

**Next Steps for MapReduce**

YARN unlocks new areas of potential for MapReduce and the Apache Hadoop framework, including:

- **Support for other parallel computing models.** YARN has received significant interest from other computational paradigm communities such as Open MPI, Spark, Storm, and Giraffa that can now integrate with Apache Hadoop very well.

- **Enhancements to the scheduler.** YARN currently schedules based on memory available on the cluster nodes, but the community is working to schedule at the CPU level, including preemption, gang scheduling, and better isolation of containers.

- **Implementation of more efficient algorithms.** The community is working on more efficient ways to perform multiple implementations of sort and shuffle algorithms. Because YARN enables various MapReduce versions to coexist safely, experiments with test versions are possible without breaking production applications.

- **Experimental MapReduce paradigms.** Work is being done on developing different types of MapReduce paradigms. So instead of processing data using map and reduce tasks, we are looking into models that use, for example, map, reduce, and reduce sequences.
This paper is derived from an interview with Deveraj Das on August 31, 2012. For the full interview, listen to the podcast.

For more information about the Apache MapReduce project, visit http://hadoop.apache.org.