Nanis Kanana Muringuri

OS Final Project

Summary: MapReduce: Simplified Data Processing on Large Clusters

MapReduce is a programming model designed to solve the problem of processing large data sets in parallel and distributed fashion specifically the automatic parallelization and load distribution aspects of it as well as the evidence of high performance even on a commodity cluster such as that of Google [1]. The load distribution aspect helps in cases where one of the machines may fail hence the need for backup. MapReduce takes care of handling machine failures, partitioning the input, scheduling the program's execution, and managing inter-machine communication.

Some of the examples of programs that can be easily expressed as MapReduce computations include Word Frequency in a large collection of documents, Distributed grep, Count of URL access Frequency, Reverse Web-Link graph, Inverted Index, and Distributed Sort or Term-Vector per Host. The abstractions of MapReduce framework, i.e. map and reduce help in hiding the details of parallelizing your workflow, fault-tolerance, distributing data to workers, and load balancing. In using MapReduce framework, the user writes map and reduce functions, and the MapReduce library executes the program in a distributed environment.

The MapReduce Programming Model

The MapReduce programming model is built into two stages called map and reduce. Map happens first, and the input to the map stage is the input of the entire problem, and likewise, the output of the reduced stage is the output of the entire problem. The map function takes an input key, value pair and outputs an intermediate key, value pair, while the reduce function accepts the intermediate key and a set of values for that key. The reduce function then merges these values to form a possibly smaller set of values. All the data in both stages is accessed using its key; a unique key assigned in the input format and the intermediate input. In the reduce stage, an iterator is used to access values from the intermediate format, and this allows us to have intermediate data that is much larger than whatever we can fit into memory on the machine running the reduced stage.

Execution of a program using the MapReduce framework

To execute a program, the framework splits the input data set into M pieces usually 64MB pieces to be processed by different machines. They are split according to their keys, and each set of keys is sent to its corresponding map worker on a cluster of worker machines also called slave nodes and a single master node. The master is responsible for load balancing and assigning tasks to the worker nodes. Specifically, it is responsible for assigning map tasks to map workers as well as monitoring the progress of each task and helping optimize for efficiency. To start the computation, the master assigns one of the M map tasks to each worker. When a worker receives a map task, it reads the contents of the corresponding input split, parsing the key-value pairs out of the input split and outputting intermediate key-value pairs by invoking the user-defined map function The map worker then stores the output of the map task which is the intermediate data locally at its machine on its hard drive. Once it has done that, it contacts master, and the master node assigns a worker the corresponding reduced task. When a reducer is notified, it uses RPC to read the map result, the

reduce function then sorts the data by key and then iterates over this sorted data, invoking the user's reduce function for each element and once that reduced task has completed it outputs its final data appended to a final output file for this partition of the data for the entire MapReduce application. Master also stores the state (idle, in progress or completed) of each map task and reduce task and the identity of the worker machine (for non-idle tasks).

MapReduce provides fault tolerance, for instance, if no response is received from a worker in a certain amount of time, it sets the task as idle so that the task can be scheduled to other workers. Even for completed tasks, if the worker failed, the task has to be re-executed because the result is stored on a local disk. For master node failure, there's no fault tolerance currently. Some features also provided by MapReduce includes the ability to skip a bad record for the next run; having a small internal HTTP server displaying all the metrics of the cluster so that people know what is going on, and a distributed counter facility for sanity check.

The MapReduce framework error handling mechanism is restarting worker machines, and this is handled transparently. This is possible because the outputs from a map worker are stored on local disk.

MapReduce Dependencies

The two main systems that enhance the proper operation of the MapReduce framework are a distributed file system for storing input and output data from MapReduce programs and a scheduling system for managing a cluster of machines.

The paper also provides some basic performance information on the framework. MapReduce is only as fast as its slowest running worker. So, any application built on this framework needs to have nearly constant time for each segment of data run on the worker. If there are wild variances is run-times between workers, it could cause a rise in total runtime. Another issue that needs to be considered is that of data. Applications that pass small amounts of data to the reduce phase are preferred while applications that pass larger amounts of data may drastically slow computation time and limit scalability when using MapReduce. In all, MapReduce is a very popular technique for distributed computing. Google is using it for several commercial services, which proves its validity. There is also open source implementation of MapReduce as more and more applications are adapted to the framework.

Related Work

The paper also mentions other related works such as MPI and Bulk Synchronous Programming that also provide parallel programming. However, MapReduce uses a restricted programming model, and by doing that it automatically provides parallelization and other benefits. Also, MapReduce relies on in-house cluster management of Google which is similar to other systems such as Condor.

As the paper shows, MapReduce could be used to implement a large number of simple applications which involve processing a large amount of data. Besides Google using the model internally, other implementations of the model (e.g., Hadoop) exist.

References

[1] Jeffrey Dean and Sanjay Ghemawat. 2004. MapReduce: simplified data processing on large clusters. In *Osdi'04: Proceedings of the 6th Conference on Symposium on Operating Systems Design and Implementation*.