After the previous two papers on GFS, BigTable, and now MapReduce all of these components are slowing showing the progression and development of what today is known as Hadoop. It is very interesting to see the progression of these technologies over time (GFS paper written in 2000 and MapReduce paper written in 2008). I had no ideas these technologies were invented at Google so the fact that I have been given this context and better understanding of how they function I am very pleased. Now to explain the paper.

The heart of the MapReduce programming model is a set of input key/value pairs that produces a set of output key/value pairs. In order to use the MapReduce function (a way to generate and compute these large analytic jobs across a distributed dataset/hardware system) a user must express their computation as two key components: a map function and a reduce function. In the paper the easiest way of expressing is to think of the problem of trying to count the number of unique words across a large quantities stored across a distributed system. A brunt force approach where the system starts on one machine and slowly works its way across all of the distributed machines would neither make sense nor be resource efficient. Instead each machine (containing its own unique documents) can determine the number of unique words across those documents and map each unique word to a word count for that machines contents. Once each machine is done the reduce function can consolidate all of those lists of unique words (and word frequencies) by summing them up and generating one final list of unique words and words counts.

This is an overly verbose explanation but the fundamental principle is there. The map invocation helps distribute a job but automatically partitioning the data into M splits - similarly the reduce partitions the intermediate key space into R pieces. The reason this part requires a bit more abstract thinking is because you have to understand the hierarchy and complexity of the reduce function. (There may be complex computations which require a hierarchical approach in the reduce function. For example: Say you want to determine a global average price of houses but also want the averages for all of the states and countries individually as well. When computing this you would sum across each hierarchical level, first states, then countries, then regions, then globally -> reducing the problem down to your final result where the intermediate results have been aggregated.)

From all of my own experience using Hadoop and MapReduce I have heard a similar story from most people. 1) The tool is exceptionally powerful in a distributed system and makes large scale computations very effective 2) The process of writing functions to operate on MapReduce (typically writing Java libraries) is time consuming and can be very complex. The system manages it effectively for you, having a single master what spawns off workers an d parses key/value pairs out of input and passes each pair to the user-defined map function. A large portion of this happens in memory and one of the reasons people have actually begun pulling out of Hadoop and moving into Spark (up to 10x faster than Hadoop) is because of its superior use of in-memory computation. All in all I would say that the conceptualization and design of MapReduce was instrumental in how we tackle large scale distributed computations (particular on data retaining to geographically separated regions being aggregated. Utilizing all of the fault

tolerance and redundancies measure on top of GFS the system has a thorough way to manage failure and anticipates it when possible.

My only compliant about MapReduce is I wish it was easier to use. (Perhaps if someone designed a graphical user interface that let people easily drag and drop and was prepared to deal with certain computational functions embedded in the system?) I know there have been a great deal of companies created around Hadoop (Hortonworks etc) but I feel there is still a bit of a barrier to entry in the sense that it is very complex to write good efficient MapReduce functions. People still rely on expensive consulting fees/services to hire someone to write these functions for them.