MAP REDUCE

Dreycey Albin

September 2019

The key need for the MapReduce programming model was to simplify tasks that require distributed computing, or tasks that could benefit from distributed computing. Short comings of the systems that were being used at the time vary. The paper mentions other systems that were similar, such as the Bulk Synchronous Programming system, or some MPI primitives. These all deliver parallel programming abstractions, but they do not contain a restrictive programming model ensuring fault tolerance [1].

The key need at the time MapReduces was published was an abstracted method for parallelization and distributed computing. MapReduce solves many problems occurring in other systems too, such as leaving the programmer responsible for machine failures when using other systems. MapReduce gets around this problem by implementing a restrictive programming model, thereby limiting the user to a handful actions ensuring fault tolerance, load balancing, locality optimization are all completed in the background [1].

There were several limitations to the MapReduce library, but most of these were ameliorated before the publication was released. For example, the paper introduced the idea/concept of "stragglers". Stragglers are processing units receiving commands from the master node which take abnormally long to perform computations. These stragglers were found to be incredible bottlenecks during certain runs for MapReduce, so the authors found a way around this by having more than one processor compute the last tasks. The output form the first processor to finish the task (usually reduce) were used [1].

The idea of the restrictive programming has caught my interest. It's interesting to see these levels of abstraction put in place, even though it's **restrictive**. While it rids of the complete flexibility one would have if performing the parallelization tasks on their own, it still allows *most* common distributed tasks to be performed. **Can this type of abstraction be used in bioengineering/synthetic biology?** Could an abstraction using the idea of restrictive programming allow scientists to genetically engineer organisms faster, increasing the probability a gene gets correctly inserted into a genome (fault tolerance analogy)?

This paper is related to the MapReduce paper because it utilizes the MapReduce framework using Hadoop for bioinformatics applications [2]. Hadoop is used to parralellize the computation of a sequence alignment algorithm, BLAST, and of a micro array analysis tool called Gene Set Enrichment Analysis (GSEA). This paper reiterated findings mentioned in the original MapReduce paper, such as the problem of load balancing. They tested distributing the sequences out among the nodes in an effort to use nodes as little as possible, but they found this actually had negative effects on the speedup. The results from using hadoop in conjunction with these other programs was amazing though, as they were able to, for example, BLAST 7,348,665 DNA sequences within 400 seconds! This goes to show the broad utility of distributed computing in an area such as bioinformatics, as well as demonstrates the potential use cases for the MapReduce framework in bioinformatics [2].

References

[1] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[2] Massimo Gaggero, Simone Leo, Simone Manca, Federico Santoni, Omar Schiaratura, Gianluigi Zanetti, Edificio CRS, and Sardegna Ricerche. Parallelizing bioinformatics applications with mapreduce. *Cloud Computing and Its Applications*, 12(18):22–23, 2008.