A Survey on MapReduce Implementations

Amer Al-Badarneh, Jordan University of Science & Technology, Irbid, Jordan
Amr Mohammad, Jordan University of Science & Technology, Irbid, Jordan
Salah Harb, Jordan University of Science & Technology, Irbid, Jordan

ABSTRACT

A distinguished successful platform for parallel data processing MapReduce is attracting a significant momentum from both academia and industry as the volume of data to capture, transform, and analyse grows rapidly. Although MapReduce is used in many applications to analyse large scale data sets, there is still a lot of debate among scientists and researchers on its efficiency, performance, and usability to support more classes of applications. This survey presents a comprehensive review of various implementations of MapReduce framework. Initially the authors give an overview of MapReduce programming model. They then present a broad description of various technical aspects of the most successful implementations of MapReduce framework reported in the literature and discuss their main strengths and weaknesses. Finally, the authors conclude by introducing a comparison between MapReduce implementations and discuss open issues and challenges on enhancing MapReduce.

KEYWORDS:
Big Data Management, Cloud Computing, Distributed File System, Incremental Data Processing, Multiprocessing, Parallel Processing

1. INTRODUCTION

In this age of data explosion and the availability of high performance and low-cost computers and with the rapid advancement of network technologies, large-scale distributed cluster systems can be extended to process and analyze a large-scale, massively parallel data (DeWitt et al., 2008). MapReduce, which has been introduced by Google (Dean & Ghemawat, 2004), is a distributed computing programming model, which is widely used to process a massive volume of data on large clusters networks. The core idea behind MapReduce framework is to allow the users to implement their own map and reduce functions, while the system is responsible for scheduling and synchronizing the map and reduce tasks (Li, Ooi, Özsu, & Wu, 2014).

Currently, MapReduce is being used in a wide range of applications, including Machine learning (Chu et al., 2006; Wolfe, Haghghi, & Klein, 2008), Singular Value Decomposition (Reza & Gunnar, 2014), textual retrieval (Elsayed, Lin, & Oard, 2008), statistical machine translation (Dyer, Cordova, Mont, & Lin, 2008), content pattern analysis (Guo, Tan, Chen, Zhang, & Zhao, 2009), and clickstream sessionization (Friedman, Pawlowski, & Cieslewicz, 2009), among others. Furthermore, the MapReduce framework has been adapted to numerous computing fields like volunteer computing environments (Lin et al., 2010), dynamic cloud environments (Marozzo, Talia, & Trufﬁo, 2012), desktop grids (Tang, Moca, Chevalier, He, & Fedak, 2010), mobile environments (Dou, Kalogeraki, Gunopulos, Mieliikainen, & Tuulos, 2010), and multi-core and many-core systems (Chen, Chen, & Zang, 2010).
Before introducing MapReduce framework, Google used many different implementations to process and compute large data. Most of the input data was very large but the computations were relatively simple. For this, the computations needed to be distributed across hundreds of computers to complete all calculations in a reasonable time. The success of MapReduce comes from its distinguishing features, including flexibility, scalability, fault tolerance, and efficiency (Li, Ooi, Özsu, & Wu, 2014). In addition, it can be used to process large datasets. As well as it hides the difficulty of data parallelization since the application developers need only to define the parameters which control data distribution and parallelism (Lämmel, 2008).

However, MapReduce has some limitations on its functionality and its unsuitability for particular types of application (Pavlo et al., 2009; Ordonez, Song, & Garcia-Alvarado, 2010; Stonebraker et al., 2010) Hence, there have been considerable research efforts that attempt to overcome the limitations of MapReduce model (Zahria, Konwinski, Joseph, Katz, & Stoica, 2008; Jiang, Ooi, Shi, & Wu, 2010; Condie et al., 2010; Bu, Howe, Balazinska, & Ernst, 2010; Thusoo et al., 2009; Olston, Reed, Srivastava, Kumar, & Tomkins, 2008; Pike, Doward, Griesemer, & Quinlan, 2005; Chambers et al., 2010).

The goal of this paper is to provide a detailed study of the current MapReduce implementations focusing on their pros and cons. A comparison between MapReduce implementations is introduced and a set of open issues and ideas is provided to enhance and improve the MapReduce framework.

Another goal is to introduce a peer to peer fault tolerant model to improve dealing with faults in SSS MapReduce. This paper is organized as follows. Section 2 reviews the architecture and the key concepts of MapReduce computing paradigm. Section 3 presents details of current implementations of MapReduce framework and their pros and cons. In Section 4, we introduce a comparison and evaluation between MapReduce implementations. Finally, Section 5 concludes the paper and discuss open issues and challenges on enhancing MapReduce.

2. MAPREDUCE ARCHITECTURE

The programming model of MapReduce is a functional model. Each application has its inputs and outputs and are specified by the user where the output is a set of \(<key, value>\) pairs. The user defines algorithm and program by using two basic functions: map and reduce. The map function receives the input data and produces a list of intermediate output \(<key, value>\) pairs. The reduce function takes all intermediate output pairs and generates the final output. Finally, the output result pairs are sorted based on their key value.

MapReduce executes the jobs across many steps and transformation processes. Figure 1 shows the basic MapReduce execution steps. First, a master node process receives a job descriptor that specifies the MapReduce job that needs to be executed in MapReduce system. The master node starts a number of mapper and reducer processes on different many machines based on job descriptor. At the same time, it reads the input data from its location. Each mapper process executes the map function which is provided by job descriptor and generates a list of intermediate output pairs. Reducer process receives the pairs that have the same keys. Hence, each reducer executes the reduce function which merges all the values in order to generate a smaller set of values. Finally, the final output is generated by grouping all reducer outputs and delivered to a certain location, defined by the job descriptor (Vaidya, 2011).

MapReduce uses the distributed file system as a solution for accessing input/output data in MapReduce systems. All nodes communicate with others and access files in the system by using a certain file system. Particularly for standard computing environments like a data center or a cluster of computers use the distributed file system. On the other hand, distributed file systems may be ineffective in large-scale dynamic cloud environments characterized by high levels of churn.
Large-Scale Systems and Society
www.igi-global.com/chapter/large-scale-systems-society/65299?camid=4v1a

Seals on Retail Web Sites: A Signaling Theory Perspective on Third-Party Assurances
Kathryn M. Kimery and Mary McCord (2008). Web Technologies for Commerce and Services Online (pp. 111-134).
www.igi-global.com/chapter/seals-retail-web-sites/31263?camid=4v1a