Computers and Electrical Engineering 42 (2015) 90–106

Contents lists available at ScienceDirect

Computers and Electrical Engineering

journal homepage: www.elsevier.com/locate/compeleceng

Flexible architecture for cluster evolution in cloud computing ☆

Tzu-Chi Huang *, Sherali Zeadally

Department of Electronic Engineering, Lunghwa University of Science and Technology, Taiwan
College of Communication and Information, University of Kentucky, USA

ARTICLE INFO

Article history:
Received 18 January 2014
Received in revised form 14 August 2014
Accepted 15 August 2014
Available online 15 September 2014

Keywords:
Architecture
Bandwidth
MapReduce
Cloud computing
Cluster
Operating system

ABSTRACT

MapReduce is considered the key behind the success of cloud computing because it not only makes a cluster highly scalable but also allows applications to use resources in a cluster. However, MapReduce achieves this simplicity at the expense of flexibility for data partitioning, localization, and processing procedures by handling all issues on behalf of application developers. Unfortunately, MapReduce currently has no solution capable of giving application developers flexibility in customizing data partitioning, localization, and processing procedures. To address the aforementioned flexibility constraints of MapReduce, we propose an architecture called Flexible Architecture for Cluster Evolution (FACE) which is both language-independent and platform-independent. FACE allows a MapReduce cluster to be designed to match various application requirements by customizing data partitioning, localization, and processing procedures. We compare the performance of FACE with that of a general MapReduce system and then demonstrate performance improvements with our implemented procedures.

1. Introduction

MapReduce [1] is a programming model proposed by Google to process a large number of datasets in a cluster [2]. MapReduce is the key behind the success of cloud computing [3] today because it not only makes a cluster highly scalable but also allows applications to use resources in a cluster easily. When working for an application in a cluster, MapReduce can make computers (also known as nodes) process well-partitioned data simultaneously without interfering with each other. MapReduce relies on its runtime system to partition input data automatically and distribute intermediate results [1] over nodes in a cluster. MapReduce hides the issues of cooperatively distributing data over nodes working for applications from application developers. All MapReduce requires from application developers is the preparation of a Map function (also known as a Mapper) and a Reduce function (also known as a Reducer) to process the application data. Technically, MapReduce runs a Mapper to process input data and produce intermediate results constructed with a series of key/value pairs while running a Reducer to merge values in intermediate results associated with the same key.

MapReduce contributes to the success of cloud computing due to its simplicity, but it does so at the expense of several other potential benefits. To achieve simplicity, MapReduce handles all parallel and distributed computing issues on behalf of application developers, but as a result, it suffers from several constraints:

☆ Reviews processed and approved for publication by the Editor-in-Chief Dr. Manu Malek.

* Corresponding author at: College of Communication and Information, University of Kentucky, USA.
E-mail addresses: tzuchi@mail.lhu.edu.tw (T.-C. Huang), szeadally@uky.edu (S. Zeadally).

http://dx.doi.org/10.1016/j.compeleceng.2014.08.006
0045-7906/

- MapReduce partitions input data into a series of fixed-size blocks (e.g., 64 MB in Google and Hadoop MapReduce implementations [1,4]) as the working units for Mappers. However, a cloud is often composed of nodes with various hardware configurations along with different performances, and a fixed but appropriate block size is not easily determined to give all applications their optimal performances. Application developers of current implementations cannot dynamically adjust the granularity of a Map task at runtime to balance workloads among nodes.
- MapReduce makes use of a built-in hash function to distribute intermediate results automatically over the corresponding nodes. Consequently, application developers cannot choose nodes to perform certain location-aware computations (e.g., to transfer intermediate results among intra-rack nodes to avoid overloading links between racks). This is because MapReduce automatically selects the node with a free slot (usually indicating an available quota of CPU resources) [1,4] to execute a task. Thus, application developers cannot change the node selection policy according to their specific criteria.
- MapReduce automatically executes a Reducer to handle intermediate results produced by a Mapper, so application developers cannot process application data outside of Mappers and Reducers. Sometimes, application developers require a post-processing procedure so that they can process outputs collected from all Reducers for certain application requirements (e.g., as inputs for the next iteration in iterative applications).

To achieve simplicity, MapReduce loses many potential benefits such as data partitioning, localization, and processing procedures because it automatically cares about most issues with the procedures without leaving application developers any room and flexibility to modify the procedures. If a MapReduce system supports application developers with flexibility in the data partitioning procedure, they can dynamically adjust the sizes of partitioned data to balance task loads at an appropriate granularity. If a MapReduce system supports application developers with the flexibility in the procedure of data localization, they can choose a certain node to run a Mapper for processing a block of input data or a Reducer for processing some intermediate results. If a MapReduce system supports application developers with flexibility in the data processing procedure, the developers can program behaviors of Mappers or Reducers like current MapReduce systems and arrange certain post-processing operations for outputs collected from all Reducers at the end of application execution, e.g., for implementing iterative applications or applying more variant computing styles to data in addition to the two-phase MapReduce computations.

In this paper, we propose a Flexible Architecture for Cluster Evolution (FACE). FACE is a flexible design architecture intended to provide application developers with system primitives that allow them to develop applications based on specific application requirements. Due to the high flexibility of the system primitives, FACE allows a MapReduce cluster to be designed for various application requirements such as load balancing, location-aware computation, special node selection policies, and customization data processing. FACE allows application developers to: submit input data in files of any size to a cloud computing environment, specify the location of intermediate results to facilitate the processing of data by local Reducers, specify which node should be responsible for running a Mapper to process input data or a Reducer to process intermediate results, and arrange a post-processing operation on outputs from all Reducers at the end of application execution. In addition to processing data with a Mapper or a Reducer, FACE allows application developers to enhance the functionality of applications with other user-defined functions (e.g., by applying certain post-processing operations to outputs collected from Reducers). Above and beyond the system primitives' support designed to help the development of an application, FACE also provides application developers with node runtime information not only to monitor progress during application execution but also to facilitate the selection of a node to perform a specific function. To optimize performance, FACE implements most components in the C language. However, FACE does allow application developers to implement their applications using other languages because FACE executes user-defined functions and provides runtime node information with language-independent interfaces.

The rest of the paper is organized as follows. In Section 2, we briefly review MapReduce, discuss related works, and highlight the research contributions of this paper. We present the proposed FACE design in Section 3 and describe its implementation in Section 4. In Section 5, we present a performance evaluation of FACE, and Section 6 concludes the paper.

2. Background, related works, and research contributions of this work

2.1. Background on MapReduce

MapReduce [1] is a programming model composed of three programs: a Master, a Mapper, and a Reducer, which can be distributed over nodes in a cluster to work co-operatively on an application. MapReduce usually has only one Master that runs on a node to monitor and control the progress of application execution. However, MapReduce may have many Mappers to process different parts of input data and many Reducers to process different parts of intermediate results produced by the Mappers. MapReduce uses a runtime system (usually a library or a standalone process) to transfer input data and intermediate results between Mappers and Reducers. According to the existing prototypes, MapReduce is often implemented to deploy the Master inside the runtime system to facilitate monitoring and controlling the progress of application execution.

The runtime system of MapReduce handles most issues such as finding a suitable node to run Mappers or Reducers, loading input data to Mappers, shuffling intermediate results among nodes, and collecting results from Reducers when an