Future Generation Computer Systems 51 (2015) 98–110

Contents lists available at ScienceDirect

Future Generation Computer Systems

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

CloudFlow: A data-aware programming model for cloud workflow applications on modern HPC systems

Fan Zhang [a,*], Qutaibah M. Malluhi [b], Tamer Elsayed [b], Samee U. Khan [c], Keqin Li [d], Albert Y. Zomaya [e]

[a] Kavli Institute for Astrophysics and Space Research, Massachusetts Institute of Technology, Cambridge, MA 02139, USA
[b] KINDI Center for Computing Research, Qatar University, Doha, Qatar
[c] Department of Electrical and Computer Engineering, North Dakota State University, Fargo, ND 58108-6050, USA
[d] Department of Computer Science, State University of New York, New Paltz, NY 12561, USA
[e] School of Information Technologies, The University of Sydney, Sydney, NSW 2006, Australia

HIGHLIGHTS

- CloudFlow programming model is designed for cloud workflow on modern HPC systems.
- CloudFlow is not only data-aware, but also shared-data-aware.
- The programming model supports multiple Map and Reduce functions.
- Theoretical analysis proves the correctness and uniqueness of each desired output.
- Results show the speedup of CloudFlow exceeds 4X compared to traditional MapReduce.

ARTICLE INFO

Article history:
Received 14 July 2014
Received in revised form
30 September 2014
Accepted 25 October 2014
Available online 6 November 2014

Keywords:
Concurrency
Data aware
MapReduce
HPC
Programming model

ABSTRACT

Traditional High-Performance Computing (HPC) based big-data applications are usually constrained by having to move large amount of data to compute facilities for real-time processing purpose. Modern HPC systems, represented by High-Throughput Computing (HTC) and Many-Task Computing (MTC) platforms, on the other hand, intend to achieve the long-held dream of moving compute to data instead. This kind of data-aware scheduling, typically represented by Hadoop MapReduce, has been successfully implemented in its Map Phase, whereby each Map Task is sent out to the compute node where the corresponding input data chunk is located. However, Hadoop MapReduce limits itself to a one-map-to-one-reduce framework, leading to difficulties for handling complex logics, such as pipelines or workflows. Meanwhile, it lacks built-in support and optimization when the input datasets are shared among multiple applications and/or jobs. The performance can be improved significantly when the knowledge of the shared and frequently accessed data is taken into scheduling decisions.

To enhance the capability of managing workflow in modern HPC system, this paper presents CloudFlow, a Hadoop MapReduce based programming model for cloud workflow applications. CloudFlow is built on top of MapReduce, which is proposed not only being data aware, but also shared-data aware. It identifies the most frequently shared data, from both task-level and job-level, replicates them to each compute node for data locality purposes. It also supports user-defined multiple Map- and Reduce functions, allowing users to orchestrate the required data-flow logic. Mathematically, we prove the correctness of the whole scheduling framework by performing theoretical analysis. Further more, experimental evaluation also shows that the execution runtime speedup exceeds 4X compared to traditional MapReduce implementation with a manageable time overhead.

1. Introduction

1.1. Background introduction

Traditional HPC systems [1], characterized by high-frequency processor design, efficient caching system, large input/output

* Corresponding author.
E-mail addresses: f_zhang@mit.edu (F. Zhang), qmalluhi@qu.edu.qa (Q.M. Malluhi), telsayed@qu.edu.qa (T. Elsayed), samee.khan@ndsu.edu (S.U. Khan), lik@newpaltz.edu (K. Li), albert.zomaya@sydney.edu.au (A.Y. Zomaya).

http://dx.doi.org/10.1016/j.future.2014.10.028
0167-739X/

capacity and efficient cooling techniques [2], have been exclusively referred to as Supercomputing [3] or Grid Computing [4]. However, upgrading hardware to achieve high performance is not only expensive, but also requires high-performance applications to leverage the hardware capacity. Modern HPC systems, represented by High-Through Computing (HTC) [1] and Many-Task Computing (MTC) [5], suggest an otherwise direction. Instead of scaling up compute resources to meet accelerated computing demand, scaling out compute resources has gained more attention. Scaling out is typically represented by incorporating more commodity compute nodes to achieve scalability purpose.

To fit in the needs of scaling compute resource out, different programming models have been proposed to maximize the scaling capacity. Among these, one of the most successful programming models is MapReduce (MR) [6]. It has been widely accepted for large-scale and data-intensive applications [7,8]. Hadoop [9], an open source implementation of MR, gains popularity due to its reliability and scalability in providing a parallel yet simple framework [10]. Hadoop MapReduce splits input data into chunks, allocating each task to a compute node where its input data resides. In this way, it achieves data locality naturally, and the task number can be configured to utilize all compute nodes.

However, splitting input dataset to realize parallelization is mostly simple and straightforward. Due to the ever-increasing complexity of execution logic in real-life applications, more and more MR applications involve multiple correlated jobs. They are encapsulated and executed in a predefined order. For example, a PageRank job [11] involves two iterative MR sub-jobs; the first job joins a rank table and a linkage table, and the second job calculates the aggregated rank of each URL. Two non-iterative MR sub-jobs for counting out-going URLs and assigning initial ranks are also included in the PageRank job.

Though the formulation of each MR sub-job is easy, the overall solution may involve many redundant MR sub-jobs. The problem becomes even worse when the MR sub-jobs involve multiple straightforward but time-consuming tasks. A concrete example demonstrating such redundant MR sub-jobs is given in Section 1.1.

As a rule-of-thumb, MR has proven to be effective in processing large amount of dividable unshared input datasets. Different and concurrently running MR jobs may share input datasets. Linkage table in the PageRank job is an example of the shared input dataset. Performance of MR applications can be improved if the sharing of input datasets is considered. Therefore, we identify two types of data sharing: data sharing among multiple sub-jobs of the same job, and data sharing among multiple distinct jobs.

Traditionally, one MR job involves multiple Map and Reduce Tasks. Each Map Task executes the same Map Function and processes its own input data splits based on the instructions defined in the Map Function. Each Reduce Task executes the same Reduce Function and processes its own key–value pair partitions. This one-map-to-one-reduce framework, though simple enough, loses its flexibility when applications require complex logic. For such complex scenarios, users employ multiple jobs, such as the implementation of PageRank and Descendant Query in [11].

In this paper, we propose Concurrent MapReduce for cloud worklow, *CloudFlow*; a novel programming model that supports multiple and heterogeneous Map and Reduce Functions. Optimizations over shared data are offered at the function level and at the job level.

1.2. Motivation example

First, we discuss a case study of increasing salary for all employees in a multi-national research company for the coming year. Fig. 1(a) depicts a scenario that the new salary for each employee is based on either his or her current title or evaluation score. The left table (Employee Information or EI) has the information of each employee including ID, name, title, current salary, and score. The upper-right table (Title Rate or TR) depicts the titles and their corresponding salary increase rates. The lower-right table (Evaluation Rate or ER) has the evaluation scores and their corresponding rates. The company wants to compare the two methods and find out the method that costs less.

In Fig. 1(b), the salary is raised based on which branch the employee is working at. The two tables on the left (EI and NC) should be joined. There are two different rate strategies as shown in the two tables on the right (CR1 and CR2).

In traditional MR, for case 1 (in Fig. 1(a)), a total of six MR jobs are launched. The first job processes EI and outputs <ID, Title, Current Salary> for each employee. The second processes TR and outputs <Title, Rate>. The third job joins the outputs and calculates the salary for each employee. Similarly, the other three follow-up MR jobs are launched to process the EI and ER and join them. Based on this, the following points can be observed:

(1) The Map stage on both TR and ER does nothing but pass the data to the Reduce Tasks to join. This observation is not new, e.g., in Hadoop Sort benchmark [1]. However, the launching of the Map tasks, though it does nothing, causes extra overhead by allocating Map slots and creating JVMs [12].
(2) There is no reason to sequentially execute the two MR jobs illustrated by dashed-line arrows in Fig. 1(a). Once the output of EI is generated, the two MR jobs can be executed concurrently. This is useful if there are more than one table that needs to be joined, such as TR and ER in Fig. 1(a). This is also useful if multiple versions of one table are to be joined (such as CR1 and CR2 in Fig. 1(b)). We hereby refer to the two concurrent joining functions as **Reduce Functions**.
(3) In Fig. 1(b), tables EI and BC should be joined before being shuffled to the two Reduce Functions. However, traditional MR supports only one-type of map, which cannot be performed on two different tables or datasets. In this paper, we support concurrent Map Functions performed on different input datasets, such as EI, NC, CR1 and CR2. We hereby refer to them as **Map Functions**.
(4) The shared EI table is read twice by the two Map Functions, but in reality, it would be more efficient to read it only once and then supply it to the two Map Functions. This significantly reduces the I/O overhead.

In this paper, the CloudFlow programming model reduces job execution time by launching concurrent and heterogeneous Map and Reduce Functions. Each Map and Reduce Function, as in traditional MR, involves a set of homogeneous Map and Reduce Tasks.

Different from the above examples where data sharing takes place among function level, data can be shared also among multiple jobs that are submitted by different users. For example, two different managers who are unaware of each other's job submit their own jobs. The first manager submits the job of case 1 in Fig. 1(a) and the second manager submits the job of case 2 in Fig. 1(b). The input dataset EI is shared by the two different jobs.

Other typical examples of job-level concurrency include scientific simulations [13], which normally need multiple runs on a set of same input data but with different application configurations or parameters. All these jobs can run concurrently.

Traditional MR offers job switching optimization mechanisms, such as the use of fair or capacity scheduler, but no such techniques exist for managing shared data among different jobs. In this paper, we propose an optimization mechanism based on the access frequency of different datasets.