Future Generation Computer Systems 62 (2016) 40–50

Contents lists available at ScienceDirect

Future Generation Computer Systems

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

Key based data analytics across data centers considering bi-level resource provision in cloud computing[☆]

Jiangtao Zhang[a,b], Lingmin Zhang[a,c], Hejiao Huang[a,c], Zeo L. Jiang[a,d], Xuan Wang[a,d,*]

[a] School of Computer Science and Technology, Harbin Institute of Technology Shenzhen Graduate School, Shenzhen 518055, China
[b] Public Service Platform of Mobile Internet Application Security Industry, Shenzhen 518057, China
[c] Shenzhen Key Laboratory of Internet of Information Collaboration, Shenzhen 518055, China
[d] Shenzhen Applied Technology Engineering Laboratory for Internet Multimedia Application, Shenzhen 518055, China

HIGHLIGHTS

- A key based mechanism is proposed to place reducer for data analysis across data centers.
- The problem is formalized as a bi-level programming.
- A unified genetic algorithm is proposed to facilitate the placement.
- The algorithm can outperform the baseline and state-of-the-art algorithms by 49% and 40%, respectively.

ARTICLE INFO

Article history:
Received 14 January 2016
Received in revised form
24 February 2016
Accepted 14 March 2016
Available online 6 April 2016

Keywords:
Reducer placement
Resource provision
Hadoop across data centers
Distributed cloud

ABSTRACT

Due to the distribution characteristic of the data source, such as astronomy and sales, or the legal prohibition, it is not always practical to store the world-wide data in only one data center (DC). Hadoop is a commonly accepted framework for big data analytics. But it can only deal with data within one DC. The distribution of data necessitates the study of Hadoop across DCs. In this situation, though, we can place mappers in the local DCs, where to place reducers is a great challenge, since each reducer needs to process almost all *map* output across all involved DCs. In this paper, a novel architecture and a *key* based scheme are proposed which can respect the locality principle of traditional Hadoop as much as possible while realizing deployment of reducers with lower costs. Considering both the DC level and the server level resource provision, bi-level programming is used to formalize the problem and it is solved by a tailored two level group genetic algorithm (TLGGA). The final results, which may be dispersed in several DCs, can be aggregated to a designative DC or the DC with the minimum transfer and storage cost. Extensive simulations demonstrate the effectiveness of TLGGA. It can outperform both the baseline and the state-of-the-art mechanisms by 49% and 40%, respectively.

© 2016 Elsevier B.V. All rights reserved.

1. Introduction

Distributed cloud consists of multiple geo-distributed data centers (DCs) which are connected by dedicated high-speed links or expensive long distance links. It provides abundant computation and storage capacity [1] and has been widely adopted to support various services, especially for data intensive applications [2], such as socially aware services [3], astronomy [4] and internet of things [5]. Because these data have sheer size and even come from disparate geographical locations, it is impractical to move such heavy geo-spanned data together and store all data in one DC. Some countries, such as the EU, have the data security laws require some data to be stored locally. Generally, data can be stored in DCs closer to data generating sources to facilitate the more frequent local data analysis with smaller access delay. For example, the US census data are collected and stored by each state [6]. The huge remote sensing data are stored in geo-distributed data centers [4]. Although these data are managed regionally, they also

☆ Part of this paper is to appear in IoTBD 2016. This new version contains substantial revisions with new architecture designs, analysis, simulation results and discussions.

* Corresponding author at: School of Computer Science and Technology, Harbin Institute of Technology Shenzhen Graduate School, Shenzhen 518055, China. Tel.: +86 755 26033789.

E-mail addresses: jiangtaozhang@aliyun.com (J. Zhang), xbmuzlm@gmail.com (L. Zhang), hjhuang@hitsz.edu.cn (H. Huang), zoeljiang@hitsz.edu.cn (Z.L. Jiang), wangxuan@insun.hit.edu.cn (X. Wang).

http://dx.doi.org/10.1016/j.future.2016.03.008
0167-739X/© 2016 Elsevier B.V. All rights reserved.

Table 1
Differences between traditional Hadoop and Hadoop across data centers.

Items	Traditional Hadoop	Hadoop across data centers
Placement of reducers	Randomly across intra-DC racks	Across multiple candidate DCs
Reduce input copying	Intra-DC	Across DCs
Reduce input storage	Intra-DC	The input data should be copied to and stored in the DC where the reducer locates

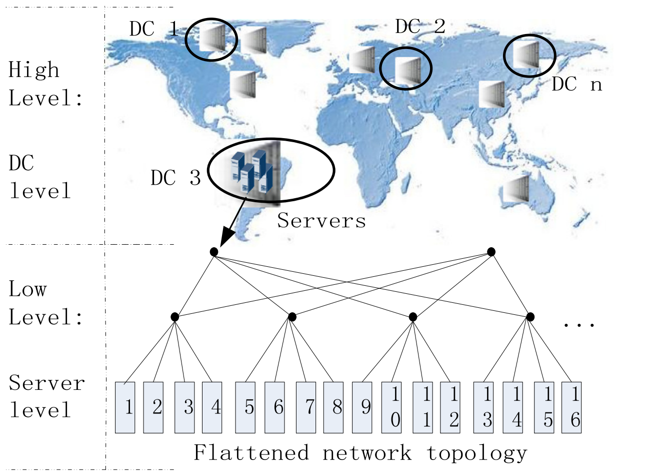

Fig. 1. Reducer placement for distributed data analysis considering both the DC and the server level resource provision in distributed cloud.

need to be processed collaboratively for a common purpose [6,4]. How to process such distributed data has attracted extensive attention of scholars [6,7] and practitioners (e.g., MapR, one of the main Hadoop suppliers, claims to support table replication across DCs in its recent version and indicates that the Hadoop framework is not necessary to be limited to one DC in the future [8]).

Hadoop, the open source version of MapReduce, which has been widely used in big data analytics, does not support data analysis across DCs in the current versions [9]. This is due to the great differences in data analytics introduced by multiple DCs. The main differences are roughly summarized in Table 1. (We used the *italic* format to indicate the general MapReduce terminologies.) For detailed information, please refer to Section 3.

To process data across DCs, a cloud service provider (CSP) should first determine which DCs are used to launch Java virtual machines (VMs) for MapReduce tasks, then the concrete servers to host the VMs. Fig. 1 illustrates the two phases. In the single-DC scenario, basically, the network within one DC is a flattened two level topology [9,10]. Considering that the intermediate *map* results are dispersed across the racks and each reducer needs to *reduce* almost all the mappers output, MapReduce does not sedulously select the location of reducers [9,10].

In the multi-DC scenario, the *map* output is stored locally in each DC. The scheme that *reduces* the data in each DC and then aggregates the intermediate results may change the final result (e.g., to calculate the global commodities sales proportion) or be inefficient [11]. So the universal case is to move all the intermediate data to the DC where the reducer locates. Generally, in each DC, the intermediate output *partitioned* to a reducer has a different size. The intuitive method is to place the reducer in the DC where more intermediate data *partitioned* to it are stored. But the premise is that the intermediate data volume should be known in advance. Fortunately, MapReduce has provided *samplers* which can be used to sample a subset of *key* space produced by mappers to approximate the distribution of *keys*, and estimate the data volume *partitioned* to one reducer. This scheme has been recommended by the definitive Hadoop guide, in chapter 8 [9] and used by other work [6]. The analysis result of data used for sampling can be reused later.

This scheme makes it possible to save the following four kinds of resources in the multi-DC scenario. First, bandwidth cost. The huge volume data copy across long distance links will incur greater bandwidth costs, increase job delay and even deteriorate availability. Second, physical machines (PMs) cost. The price of computation resources of PMs in different DCs varies widely [12]. Placing more reducers in the expensive DC will lead to more costs. Third, storage cost. Because the data transfer across DCs will take more time, the *map* output should be pre-copied to the target DC, where the reducer locates and stored in the DC in the whole *reduce* phase. This leads to additional storage costs. Since DCs also differ in the pricing of storage [13], large volume data should avoid being stored in the expensive DC. Bandwidth costs and PM costs (including computation and storage) contribute about 60% to the costs of a DC [14], another key contribution is power. Power draw accounts for 15% not including cooling and power distribution [14] and is the fourth cost to be optimized. CSP can further cut power costs by exploiting various electricity prices of distributed DCs [15] when placing reducers. The locations of reducers have a big influence on costs and services.

Each reducer can be assigned to a slot equipped with the same determined computation and memory resources in Hadoop 1.0. Hadoop 2.0 (YARN) has permitted the configuration of CPU and memory of mappers and reducers [16], so as to capture the heterogeneous capability of PMs and match the volume of data to be processed, hence improve the resource efficiency. Inside each DC, the different size VM can be consolidated so that fewer PMs are used and more power is saved.

In recent years, several MapReduce-like frameworks, such as G-Hadoop [7] and G-MR [6], have been proposed to process distributed data across DCs. G-Hadoop respects the locality principle and prefers to *map* data locally. It is flexible and can embrace any schemes in the hierarchical architecture. By default, it uses the traditional scheme to place reducers, i.e., randomly placing reducers in involved DCs. All intermediate data *partitioned* to one reducer are then copied to the DC where the reducer locates. G-MR [6] uses an advanced scheme by defining a single directional weighted graph to depict the data placement and transfer path. All input data in each DC should be partitioned to equal size partitions. The nodes of the graph are the combination of partitions in different DCs before *map* or *reduce* phase and the edge is the transfer path from one node to another. Some tactics are further introduced to avoid nodes and paths catastrophes. But just as the analysis in Section 2, it is hard to determine the partition size. The coarse-grained partition and the introduced tactics will offset the optimality of the solution. Furthermore, it cannot address power cost and can only be used when the data are associative, i.e., the iterative and hierarchical *reduce* will not change the final result.

Focusing on the distributed data analytics, this paper explores the problem of placement of the reducer VMs costs efficiently across the involved DCs. The main contributions are as follows:

1. The differences between data analytics inside one DC and across DCs are analyzed in detail. Based on the analysis, an architecture and a *key* based scheme are proposed to determine the location of reducers and optimize data transfer cost. This method can achieve more accurate data copy and further indicate the configuration of reducers based on the data volume *partitioned* to it. It is applicable to all data no matter whether the data are associative or not.
2. Considering both DC determination and the last VMs placement in the servers, as well as the costs of data transfer, storage, computation and power, the problem is formulated as a 0–1 integer linear bi-level programming.

Download English Version:

https://daneshyari.com/en/article/425536

Download Persian Version:

https://daneshyari.com/article/425536

Daneshyari.com