Future Generation Computer Systems 27 (2011) 843–849

Contents lists available at ScienceDirect

Future Generation Computer Systems

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

Variable-sized map and locality-aware reduce on public-resource grids

Yen-Liang Su [a,1], Po-Cheng Chen [a,*,1], Jyh-Biau Chang [b], Ce-Kuen Shieh [a]

[a] Institute of Computer and Communication Engineering, Department of Electrical Engineering, National Cheng Kung University, Taiwan

[b] Department of Digital Applications, Leader University, Taiwan

ARTICLE INFO

Article history:
Received 10 May 2010
Received in revised form
24 August 2010
Accepted 1 September 2010
Available online 22 September 2010

Keywords:
MapReduce
Grid computing
Load balance
Scheduling

ABSTRACT

This paper presents a grid-enabled MapReduce framework called "Ussop". Ussop provides its users with a set of C-language based MapReduce APIs and an efficient runtime system for exploiting the computing resources available on public-resource grids. Considering the volatility nature of the grid environment, Ussop introduces two novel task scheduling algorithms, namely, Variable-Sized Map Scheduling (VSMS) and Locality-Aware Reduce Scheduling (LARS). VSMS dynamically adjusts the size of map tasks according to the computing power of grid nodes. Moreover, LARS minimizes the data transfer cost of exchanging the intermediate data over a wide-area network. The experimental results indicate that both VSMS and LARS achieved superior performance than the conventional scheduling algorithms.

1. Introduction

MapReduce [1], popularized by Google, is an emerging programming model for large-scale data-parallel applications such as web indexing, data mining, and scientific simulation [2]. Recently, there has been a dramatic proliferation of research concerned with various MapReduce framework design and implementation. For example, Apache Hadoop [3] is an open source implementation of MapReduce sponsored by Yahoo! It is used at Facebook, Amazon, and many others [2]. Besides, Phoenix [4] implemented the MapReduce model for the shared memory architecture, i.e. multi-core and multi-processor systems. Moreover, Mars [5] implemented MapReduce on the graphic processors and Rafique et al. implemented MapReduce for the Cell B.E. architecture [6]. While substantial studies have been performed on providing MapReduce frameworks for dedicated data center environment [1,3,6], virtual machine clusters [7,8] or even inside a single machine [4,5], relatively little literature has been published on leveraging MapReduce model on public-resource grids [9–11].

On the industry front, companies such as Google and its competitors may have adequate budgets for constructing large-scale data centers to provide sufficient computing and storage resource. On the other hand, non-profit organizations could not afford to build their own large-scale data centers without any financial support. Fortunately, public-resource grids, which federate donated computational power and storage to run as a cooperative system for harvesting the idle CPU cycles and unexploited disk space, may be a feasible platform for running MapReduce applications of non-profit computing projects. However, running the MapReduce model in the grid environment differs markedly from running it in the data center environment in at least two aspects.

Firstly, the grid environment has the volatility feature. Unlike data centers, which commonly consist of homogeneous and dedicated nodes, public-resource grids usually consist of heterogeneous and non-dedicated nodes. Grid nodes are probably supercomputers, single or multi-processor/multi-core PCs; the computing capability of each of them is normally different from others. Moreover, each grid node is shared between its owner and multiple grid users; and thus when a MapReduce application is run on a grid, the owner's and other grid users' job compete concurrently against the MapReduce application for the grid node resources. Such circumstances make a difficulty of the MapReduce job scheduling. According to the MapReduce programming model, all map tasks have to be finished before reduce tasks get started; Still most of conventional map task scheduling algorithms [1,3] assign equal-sized map tasks to each of map workers. Apparently, a map task of the MapReduce application running on the grid node with the poorest capability, i.e. straggler [1,7], will introduce a bottleneck effect degrading the performance of the entire MapReduce application. Therefore, a requirement exists for a grid-aware map task scheduling algorithm capable of dynamically balancing the workload distribution between volatile grid nodes.

* Corresponding address: Institute of Computer and Communication Engineering, Department of Electrical Engineering, National Cheng Kung University, No. 1, Ta-Hsueh Rd., Tainan City 701, Taiwan. Tel.: +886 6 2757575x62400 1779; fax: +886 6 234 5486.

E-mail addresses: kid@hpds.ee.ncku.edu.tw, chen.pocheng@gmail.com (P.-C. Chen).

[1] These authors contributed equally to this project and should be considered as co-first authors.

0167-739X/$ – see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2010.09.001

Secondly, the nodes in a data center are connected to each other over a local-area network (LAN), while the nodes in a public-resource grid are often distributed over a wide-area network (WAN). When a reduce worker requests associated intermediate key/value pairs produced by map workers, the cost of inter-site communication in the grid environment is obviously higher than the cost of intra-site communication in the data center environment. In this situation, the unsteadily available bandwidth of a WAN causes another difficulty of the MapReduce job scheduling. However, most of available reduce task scheduling algorithms [1,3,6,7] assumes that all data transfers are intra-site communication. To address this problem, it is desirable to develop a grid-aware reduce task scheduling algorithm to minimize inter-site communication over a WAN.

According to the above discussions, this paper presents a grid-enabled MapReduce framework called "Ussop". Ussop provides its users with a set of C-language based MapReduce APIs and an efficient runtime system for exploiting the dynamic and non-dedicated resources available on public-resource grids. Ussop hides the complexity of job parallelization, task distribution, and data partition from the users. It uses the variable-sized map scheduling (VSMS) algorithm during the map phase. VSMS algorithm achieves load balance by dynamically adjusting the size of a map task and assigning larger-sized map tasks to the grid nodes with higher capability. Moreover, it uses the locality-aware reduce scheduling (LARS) algorithm during the reduce phase. LARS minimizes the cost of data transfer by assigning a reduce task to an appropriate grid node in accordance with the data locality information. Thus, Ussop can alleviate the straggler problem caused by various availabilities of grid nodes, and enhance the performance of a MapReduce job.

The remainder of this paper is organized as follows: Section 2 briefly reviews the major MapReduce frameworks presented in the literature. Section 3 discusses the design and implementation of Ussop, while Section 4 evaluates its performance under exhaustive experiments. Finally, Section 5 presents some concluding remarks and indicates the intended direction of future research.

2. Related work

Several notable MapReduce frameworks have been proposed in the literature [6–8,11]. They are briefly compared with Ussop in this section.

Cloudlet [8] implemented MapReduce on virtual machine clusters to gain the benefits of the virtualization technique such as better performance in management and security issues. However, running a MapReduce application on such environment suffers from poor performance due to the heavy overhead of I/O virtualization. Specifically, virtual machines hosted by the same physical node have to compete for the limited network bandwidth against each other. To address this problem, Cloudlet divided the reduce phase of original MapReduce programming model into the local and the global reduce phases. The local reduce phase executes the sort and reduce functions within the same physical node, thereby minimizing the network bandwidth competition. On the other hand, the reduce workers running in a grid are distributed over more diverse network environment than that in a virtual machine cluster, consequently Ussop adopts the LARS algorithms to minimize inter-site communication over a WAN.

The conventional speculative execution mechanisms [1,3] only consider running a MapReduce application in a data center environment, which consists of homogeneous nodes. Such algorithms may misjudge any node with poorer resource availability as a straggler. Accordingly, Zaharia et al. has clearly indicated that the speculative execution strategy of Hadoop may be not robust enough for the environment more volatile than a data center environment [7]. To enhance the original speculative execution strategy, they proposed the Longest-Approximate-Time-to-End (LATE) algorithm. The LATE algorithm defines both a cap on the number of speculative tasks and a slow task threshold to prevent unnecessary speculative executions. In contrast, Ussop adopts the VSMS algorithms to reduce the adverse performance impact due to the straggler problem by dynamically balancing the workload distribution between volatile grid nodes.

MapReduce.Net [11] is somewhat similar to Ussop. It implemented a MapReduce framework based on Aneka [12]. It relies on Aneka to harvest the idle resource in an enterprise grid environment, and uses these resources to execute MapReduce applications. Due to the volatility feature of grid nodes, both MapReduce.Net and Ussop dynamically assign reduce tasks to appropriate nodes by using a locality-aware scheduling algorithm rather than a static scheduling algorithm to assign tasks. However, the task size in MapReduce.Net is equal-sized and the task size in Ussop is variable in accord with the computing power of workers.

Moreover, in the literature [6], the dynamic work unit scaling (DWUS) algorithm is also similar to the VSMS algorithm of Ussop. This literature [6] was targeted for supporting MapReduce on a cluster consisting of well-provisioned blades and computing accelerators with limited memory and I/O capacity. Therefore, when an application starts, a binary search method was used by DWUS for achieving the highest service rate. The binary search method is only executed in a period of time. Then, the final workload size determined is used for the rest of the application execution. However, the task size in Ussop is dynamic because of the volatility nature of the grid environment.

3. Ussop: system design and implementation

Many MapReduce frameworks have been designed for various platforms; however, Ussop is designed for a public-resource grid environment. Fig. 1 illustrates the general system overview of Ussop. Once a MapReduce application is submitted to the Ussop portal, the Ussop portal chooses several grid nodes to run the application. One of these gird nodes is chosen to be the master of the application and the rest are chosen to be workers. Each idle worker then requests a map or a reduce task from the master.

When a worker is assigned a map task, firstly it has to read the corresponding input data. Existing MapReduce frameworks such as Google's implementation and Hadoop run MapReduce applications inside a dedicated data center. They assume that the replicas of the input data have been distributed across nodes in the data center in advanced. On the contrary, the grid nodes exploited by Ussop are chosen on demand; thus the input data cannot be stored in these grid nodes in advance. Consequently, the map worker has to read the input data from the user's node that submits the MapReduce job or from the remote replica servers.

The grid nodes exploited by Ussop are usually from several geographically distributed sites. Moreover, they are normally heterogeneous and non-dedicated. Obviously, the homogeneity assumptions of conventional designs do not hold in Ussop anymore. Thus, Ussop introduces new scheduling algorithms, i.e. VSMS for the map tasks assignment and LARS for the reduce tasks assignment. The remainder of this section describes the concept of VSMS and LARS and the implementation of Ussop in detail.

3.1. Variable-sized map scheduling (VSMS)

Conventional MapReduce frameworks assume that workers can perform tasks at the same rate. Thus, the master of such frameworks assigns equal-sized map tasks to the idle workers. Moreover, such frameworks also assume that any detectably