Volume : 3 | Issue : 5 | May 2014 • ISSN No 2277 - 8179

Extending Hadoop to Improve Support for Multiple-input Applications

Engineering

KEYWORDS : hadoop; bi-hadoop; data locality

Sarath C S4 M.Tech, Dept of Computer Science & Engineering, N.S.S College of Engineering, Palakkad, Kerala, India.

Mrs Usha K Associate Professor, Dept of Computer Science & Engineering, N.S.S College of Engineering, Palakkad, Kerala, India.

ABSTRACT

Hadoop is a MapReduce programming model which provides a cost effective solution for many data-intensive applications. Hadoop stores data distributively and exploits data locality by assigning tasks to where data is stored. Many data-intensive applications, however, require two (or more) input data for each of their tasks. Such applications pose significant challenges for Hadoop as the inputs to one task may reside on multiple nodes, and Hadoop is unable to discover data locality in this scenario. This often leads to excessive data transfers and significant degradations in application performance. So, Bi-Hadoop was introduced as an efficient extension of Hadoop to better support binary-input applications. Bi-Hadoop integrates an easy-to-use user interface, a binary-input aware task scheduler, and a caching subsystem. Experiments show that Bi-Hadoop can significantly improve the execution of binary-input applications by reducing the data transfer overhead, and outperforms existing Hadoop by more than 3x. In this paper, we introduce a further enhancement of Bi-Hadoop by incorporating support for multiple input applications, that is, applications in which the input may reside on more than two nodes.

Introduction

Large-scale data intensive computing has become indispensable for many applications to gain insights from increasing volumes of data. The MapReduce programming model, along with its open-source implementation Hadoop, has provided a cost effective solution for such data processing needs. Hadoop is designed to support data intensive applications on clusters of hundreds or thousands of compute nodes.

However, the locality-awareness of Hadoop is based on a relatively strong assumption that *a task is expected to work on a single data split.* In practice, a split typically consists of one data block, or a part of it. After all, this is what allows Hadoop to label a compute node as *local* or *remote* for scheduling purposes. This is in accordance with the MapReduce programming model, which defines one map task over each logical data split and thus requires users to describe the mapper function as a unary operator, applicable to only one single logical data split [1].

The unary-input requirement works well for many applications such as document processing. However, many other applications require more flexible operators. For example, a task in a pattern matching application would naturally take two inputs: one record of the template data, and another record of the stored data. For such applications, the unary input oriented Hadoop system has multiple limitations. So, Bi-Hadoop was implemented as an extension of the existing Hadoop system, by making the following modifications:

- An easy-to-use interface for users to describe the association between a task and its inputs.
- A task scheduling algorithm that is able to exploit data locality for binary-input applications.
- A caching mechanism to accelerate data reads.

Extensive experiments were conducted to verify the effectiveness of Bi-Hadoop, and the results showed that Bi-Hadoop outperforms Hadoop by upto 3.3x, for binary input applications. The aim of this project is to further enhance this Bi-Hadoop to support multiple input applications. The project also aims at a performance evaluation of homogenous and heterogenous Bi-Hadoop clusters.

Bi-Hadoop: An Extension of Hadoop

As said earlier, Bi-Hadoop is an extension of Hadoop to support binary input applications [1]. Bi-Hadoop can significantly improve the execution of binary input applications by reducing the data transfer overhead.

Design Overview

Figure illustrates an overview of Bi-Hadoop, which contains the following components:

(1) the input interface,
(2) the caching subsystem, and
(3) the binary-input locality-aware scheduler.

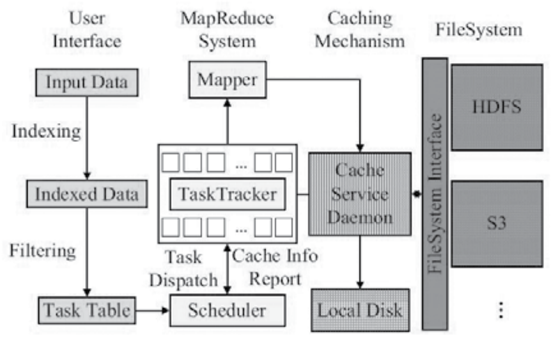

Fig 1: Bi-Hadoop Extension System Overview

Input interface: This component assigns IDs to splits. Bi-Hadoop inherits the default Hadoop output format and adds a hook so that an ID can be designated to each split. In Bi-Hadoop, tasks are generated by calling a user-defined filter function that specifies which two splits would form a valid task.

Scheduler: The scheduler first obtains the task representation by applying the filter function from the user interface, then gathers information about the locations of the input data blocks (which DataNode has which blocks), and subsequently calculates a locality-optimized execution schedule.

Caching subsystem: This component runs on each compute node and is designed to cache the input splits accessed by existing tasks (with the expectation that they will also be needed by subsequent tasks). The caching subsystem sits between MapReduce system[2] and HDFS system[3], therefore it is user transparent. It supplies a split to the requesting map task if the split is in the cache, and will seamlessly resolve to the native HDFS data read mechanism when the requested split is missing in the cache.

System Design

User Interface

The Bi-Hadoop user interface is designed to assign IDs to splits by letting the user name splits with strings. Users specify the map tasks by customizing a filter class, which returns true if a

Volume : 3 | Issue : 5 | May 2014 • ISSN No 2277 - 8179

pair of split IDs form a task, and alse otherwise.

The Locality-aware Sceduler
The scheduler weaves all the components together in Bi-Hadoop. Once a MapReduce job is submitted, the scheduler first creates an internal presentation of the tasks. The scheduler will then monitor the locality of the data splits (disk replicas and copies in the cache subsystem), exploit data sharing pattern among the tasks, and assign tasks to optimize data localities for the tasks[5]. Scheduling in Bi-Hadoop is performed in three phases:

Phase 1: Task Generation
Binary-input applications have two sets of input splits, *A* and *B*, and a task will take a split from *A* and another one from *B*. Note that *A* and *B* may overlap, either partially or completely. In this phase, we run the user-supplied filter function (discussed in the user interface) and generate an internal presentation of the tasks in the form of an incidence matrix I. Figure illustrates an example of the incidence matrix for matrix-vector multiplication. The matrix is of size 2 by 4 blocks and the vector is 4 blocks. Each value 1 in the matrix indicates a task that multiplies a matrix block with a vector block.

Phase 2: Static Task Grouping
Taking the incidence matrix as input, we partition the rows and columns into groups such that tasks within the same group share their input file splits. The static grouping phase provides an insight into the relation between tasks: tasks from the same group are likely to share (some) input splits, and if we assign them to a common compute node, we will see reduced data transfers.

Phase 3: Dynamic Task Dispatching
The dynamic task dispatching phase is executed during run time, it decides which node should execute which tasks, and the goal is to reduce data transfers while maintaining load balancing across the compute nodes. A *task pack* is the set of tasks that will be executed together by a compute node. Task pack is the unit of actual task dispatching. We use the following criteria when forming task packs:

(1) the number of tasks in a pack should not exceed the total number of tasks dividing the number of nodes;
(2) the difference between the number of row splits and the number of column splits is small;
(3) at most half of the cache will be used for row splits or column splits.

Task packs will be created such that tasks in a pack share their input splits; and the input file splits are likely to be already has a local replica or in cache. Tasks in a pack are then scheduled onto the corresponding compute nodes (represented by its TaskTracker) one by one. To take load balance into account, when no more packs can be formed for an idle TaskTracker, i.e., when all tasks have been already assigned to some pack, our scheduler falls back to the default Hadoop scheduling algorithm so that this idle TaskTracker can steal tasks from some pack that is assigned to some other TaskTracker.

Caching Subsystem
The caching subsystem has two components: a file handler object and a service daemon. The handler object is constructed when opening files. The service daemon sits on top of the file system abstraction of each compute node.

The service daemon serves handlers' requests, manages the cached blocks and reports caching status to the TaskTracker for scheduling. When it receives a caching request, it checks if the required data is in the cache. If not, the daemon uses the usual file system API(such as HDFS API) to read the data and saves blocks into local file system.

With Bi-Hadoop, users will only need to perform one extra piece of work than with existing Hadoop: specifying which two splits form a task. Other than this, they just write Hadoop programs as they currently do: focusing on the mapper and reducer functions. The three phases of our scheduler as well as the caching subsystem are transparent to the users. Bi-Hadoop can be implemented by making modifications in Hadoop configuration files[10][11]. Extensive experiments with pattern matching and Page rank algorithms show that Bi-Hadoop can significantly improve the execution of binary input applications by more than 3x.

Proposed System
Bi-Hadoop is designed for applications with exactly two inputs. The aim of this project is to further add flexibility by making Bi-Hadoop suitable to handle multiple input applications too. This can be done by making changes in the interface, task scheduler, and caching subsystem of the existing Bi-Hadoop system [1].

So, as the first step, we need to configure Hadoop on 3 nodes, of which one will be the master node, the master node itself would act as a slave also. Then we would set up single-node clusters on these individual nodes, and then would merge them to form a multi-node cluster with 3 nodes. By using virtualization software such as VMware and DropBox, it would be able to setup clusters with more nodes. Now, changes have to be made in the user interface, task scheduler, and caching subsystem of these Hadoop nodes thus making it Bi-Hadoop. These changes can be made by editing the Hadoop configuration files; this is possible since Hadoop is an open-source framework.

Now, the final step is to make further modifications to make it more compatible to handle multiple-input applications. This can also be done by editing the Hadoop configuration files.

Conclusion
So, by making changes in the input interface, task scheduler, and caching subsystem, the performance of Hadoop for binary-input applications can be enhanced. Also, this may be extended to improve the performance of multiple-input applications, that is, applications in which the input reside on more than two nodes.

REFERENCE

[1] Xiao Yu and Bo Hong, "Bi-Hadoop: Extending Hadoop To Improve Support for Binary-Input Applications", 13th IEEE/ACM International symposium on Cluster, Cloud, and Grid Computing, 2013 | [2] J. Dean, S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters", OSDI, 2004 | [3] Dhruba Borthakur, The Hadoop Distributed File System: Architecture and Design, 2007 | [4] Jeffrey Dean, Sanjay Ghemawat, "MapReduce: A Flexible Data Processing Tool", Communications of the ACM, Vol. 53, no. 1, Jan 2010 | [5] Zhenhua Guo, Geoffrey Fox, Mo Zhou, "Investigation of Data Locality in MapReduce", School of Informatics and Computing Indiana University Bloomington, February 2011 | [6] Weina Wang, Kai Zhu and Lei Ying, "Map Task Scheduling in MapReduce with Data Locality: Throughput and Heavy-Traffic Optimality", Jan 2013 | [7] Kyong-Ha Lee, Yoon-Joon Lee, "Parallel Data Processing with MapReduce: A Survey", SIGMOD Record, Vol. 40, no. 4, Dec 2011 | [8] T. White, Hadoop: The Definitive Guide, O'Reilly Media, Inc., 2009 | [9] "MapReduce", https://hadoop.apache.org/docs/mapred _tutorial.html, April 2013 | [10] "The Hadoop Distributed File System", http://developer. yahoo.com/hadoop/tutorial/module2.html | [11] "Hadoop Components", http://www.guruzon.com/6 /hadoop-cluster/architecture/hadoop-components |