Efficient Analytics on Ordered Datasets using MapReduce

Jiangtao Yin
UMass Amherst
jyin@ecs.umass.edu

Yong Liao
Narus Inc.
yliao@narus.com

Mario Baldi
Narus Inc.
mbaldi@narus.com

Lixin Gao
UMass Amherst
lgao@ecs.umass.edu

Antonio Nucci
Narus Inc.
anucci@narus.com

ABSTRACT

Efficiently analyzing data on a large scale can be vital for data owners to gain useful business intelligence. One of the most common datasets used to gain business intelligence is event log files. Oftentimes, records in event log files that are time sorted, need to be grouped by user ID or transaction ID in order to mine user behaviors, such as click through rate, while preserving the time order. This kind of analytical workload is here referred to as RElative Order-pReserving based Grouping (RE-ORG). Using MapReduce/Hadoop, a popular big data analysis tool, in an as-is manner for executing RE-ORG tasks on ordered datasets is not efficient due to its internal sort-merge mechanism. We propose a framework that adopts an efficient group-order-merge mechanism to provide faster execution of RE-ORG tasks and implement it by extending Hadoop. Experimental results show a 2.2x speedup over executing RE-ORG tasks in plain vanilla Hadoop.

Categories and Subject Descriptors

H.3.4 [**Systems and Software**]: Distributed systems

General Terms

Design, Performance, Experimentation

Keywords

MapReduce/Hadoop; distributed framework; ordered dataset

1. PROBLEM STATEMENT

Large corporations produce and collect terabytes of data on a daily basis with the purpose of analyzing them to continually improve their services and operations. Several classes of data used to gain business intelligence have a temporal dimension, such as webpage click streams, network traffic traces, and business transaction records. Furthermore, a lot of analytic tasks over such temporal data require to group data points based on a certain feature and temporally sort them within the group. Many important analytic jobs, including user online activity sessionization, TCP/IP flow construction, and customer statement generation, treat such tasks as a vital part of their execution. The corresponding input datasets can be generally seen as event log files, in which each record is about an event occurring at a given point in time. Such datasets have the important property that since the records are placed in the dataset as they are generated, they are temporally ordered.

More generally, this kind of dataset can be represented as a list of *records* consisting of a *primary key*, a *secondary key*, and a *value*, sorted by the secondary keys. Several popular and business critical analytics tasks use such a dataset as an input to generate a set of output data points, each one being a function of a *group* of records from the input dataset that satisfy the following conditions: (1) records are grouped based on their primary keys; (2) records in a group are sorted by their secondary keys. We define the *RElative Order-pReserving based Grouping*, or RE-ORG, as the processing that creates groups satisfying the above requirements.

Efficiently executing RE-ORG over large quantity of data in a distributed environment is challenging. MapReduce [3] and its extensions [4, 6–8] have emerged as scalable frameworks for data intensive computation using a large cluster of commodity machines. However, realizing RE-ORG tasks using MapReduce in an as-is manner is not efficient. MapReduce distributes input records to mapper nodes (workers) in the cluster and then groups them by their primary keys on the reducer nodes utilizing an internal *sort-merge* scheme, which cannot take advantage of the fact that the input records are already sorted by secondary key. Sorting records by secondary key in the same group (i.e., having the same primary key) requires to either write custom code or instrument the MapReduce framework (e.g., Hadooop) to do that. Either way, the sorting operation is time consuming.

2. OUR SOLUTION

This work proposes a novel *group-order-merge* (GOM) mechanism to replace the *sort-merge* shuffle scheme of MapReduce, so as to efficiently support the execution of RE-ORG in a distributed environment. The property of the input dataset being sorted by secondary key is used to speed up the execution of RE-ORG. Figure 1 shows how the three phases of the GOM mechanism are distributed between mapper and reducer workers.

Group Phase: A worker (mapper) sequentially extracts the records from its input chunk and groups them by applying a hash function on their primary keys. The output of the group phase is a set of *segments*, each containing a set of *lists*. Each list includes all the records in the processed chunk with the same primary key. Since records are processed sequentially, each list preserves the ordering records have in the input dataset.

HPDC'13, June 17–21, 2013, New York, NY, USA.
ACM 978-1-4503-1910-2/13/06.

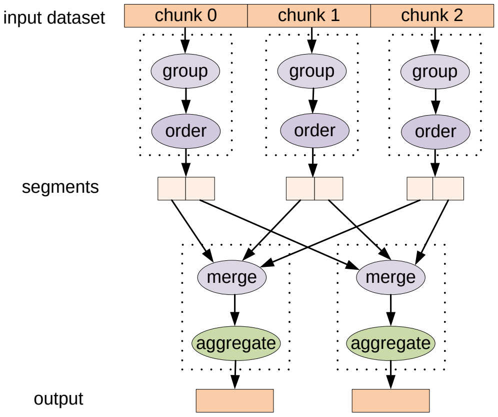

Figure 1: Group-order-merge phases.

Order Phase: Lists in each segment are sorted based on their primary keys. This is an important preparation step to then allow reducer nodes to efficiently merge segments produced by different mapper workers. Note that each list is treated as a whole in the order phase, which is much more efficient than handling each record individually. The sorted segments are then distributed to a set of reducer workers based on the primary key (i.e., one segment to one worker and guaranteeing all lists whose records have the same primary key will be received by the same reducer worker).

Merge Phase: The records contained in segments for the same primary key received from different mappers are merged into one final list. The merging algorithm must ensure that records in the resulting list of each segment preserve their relative ordering as in the input dataset. The fact that lists in the segments received from different mappers are sorted by primary key allows the merging algorithm to have minimum complexity by creating the secondary-key-sorted list corresponding to each primary key value in a single pass.

After the merge phase, data is ready to allow each reducer to perform the aggregation operation on the final secondary-key-sorted list[1].

3. IMPLEMENTATION AND EVALUATION

The group-order-merge (GOM) mechanism was implemented as an alternative shuffle scheme in Hadoop [1] to the default sort-merge one. Hadoop was selected as the basis for the implementation because a Re-Org task maps very well to the MapReduce programming model and Hadoop is the most popular open-source framework supporting such programming model. The popularity of Hadoop stems from its good performance in handling failures and capability of scaling to a large number of worker nodes. The prototype implementation that provided the results shown below is based on Hadoop version 1.0.3.

A user online activity sessionization task was run on the well known 116GB click stream dataset related to World Cup 1998 [2] using a 10-node cluster. This sessionization task first groups clicks by user and then divides the click stream of each user into browsing sessions based on a timeout threshold (e.g., 5 minutes). The performance of the sessionization task on GOM-enhanced Hadoop was compared to the one of vanilla Hadoop using the stock secondary sort implementation [5]. As it can be seen in Figure 2, the GOM-enhanced Hadoop prototype achieves a 2.2x speedup on the completion time over vanilla Hadoop.

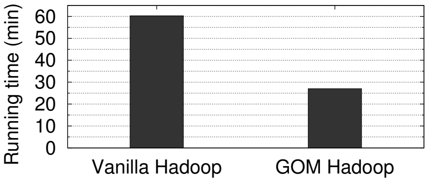

Figure 2: Running time of user sessionization.

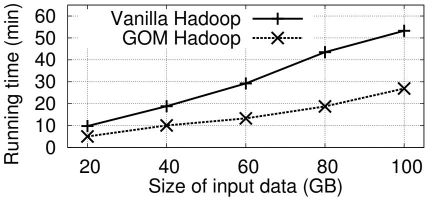

Figure 3: Running times of user sessionization on data with varying size.

Figure 3 shows how the performance of the sessionization task on a 10-node GOM-enhanced Hadoop cluster scales with increasing size of the input dataset compared to the vanilla Hadoop cluster. The 116GB click stream dataset was divided in subsets of different sizes on which the user online activity sessionization task was run. As shown in Figure 3, the running times on the GOM-enhanced framework increase linearly with the size of the input dataset outperforming vanilla Hadoop with any input data size.

Acknowledgments

This work is partially supported by NSF grant CNS-1217284.

4. REFERENCES

[1] Apache Hadoop. http://hadoop.apache.org/.

[2] World Cup 1998. http://ita.ee.lbl.gov/html/contrib/WorldCup.html.

[3] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. In *OSDI '04*, 2004.

[4] J. Ekanayake, H. Li, B. Zhang, T. Gunarathne, S.-H. Bae, J. Qiu, and G. Fox. Twister: a runtime for iterative MapReduce. In *HPDC '10*, 2010.

[5] T. White. *Hadoop: The Definitive Guide.* O'Reilly Media, Inc., 1st edition, 2009.

[6] J. Yin, Y. Zhang, and L. Gao. Accelerating expectation-maximization algorithms with frequent updates. In *CLUSTER '12*, 2012.

[7] Y. Zhang, Q. Gao, L. Gao, and C. Wang. iMapReduce: A distributed computing framework for iterative computation. In *IPDPSW '11*, 2011.

[8] Y. Zhang, Q. Gao, L. Gao, and C. Wang. PrIter: A distributed framework for prioritized iterative computations. In *SOCC '11*, 2011.

[1] The aggregation operation is not part of the proposed mechanism, but constitutes the implementation of the actual analytics task.