Hadoop Data Cube In Python: TSCube

Song Ziwenhan 12330279
Ruan Shihai 12330272
Si Jiyu 12330278
Shi Tianzuo 12330275
Wang Hongji 12330302

May 30, 2014

Abstract

Data Cube is a multi-dimensions Group by, and our goal is to make this process efficiently and fast. Hadoop is an imitated distributed file system from Google Mapreduce, which is an open source system. To cube a large amount of data, the HDFS is a nice choice, and this project combines the batch cube method and terasort method into an aggregate in Python.The final program will be a powerful tool to cube multi-dimensions data.

1 Introduction

1.1 Hadoop Distributed File System (HDFS)

- During the **Map** phase, the input data is distributed across the mapper machines, where each machine then processes a subset of the data in parallel and produces one or more <key, value> pairs for each data record.

- During the **Shuffle** phase, those <key, value> pairs are repartitioned (and sorted within each partition) so that values corresponding to the same key are grouped together into values {v1, v2, ...}.

- During the **Reduce** phase, each reducer machine processes a subset of the <key, {v1, v2, ...}> pairs in parallel and writes the final results to the distributed file system.

- The map and reduce tasks are defined by the user while the shuffle is accomplished by the system. Fault-tolerance is inherent to a MapReduce system, which detects failed map or reduce tasks and reschedules the tasks to other nodes in the cluster.

1.2 Data cube

```
SELECT Model, Year, Color, SUM(sales) AS Sales
FROM Sales
WHERE Model in {'Ford', 'Chevy'}
  AND Year BETWEEN 1990 AND 1992
GROUP BY CUBE Model, Year, Color;
```

SALES

Model	Year	Color	Sales
Chevy	1990	red	5
Chevy	1990	white	87
Chevy	1990	blue	62
Chevy	1991	red	54
Chevy	1991	white	95
Chevy	1991	blue	49
Chevy	1992	red	31
Chevy	1992	white	54
Chevy	1992	blue	71
Ford	1990	red	64
Ford	1990	white	62
Ford	1990	blue	63
Ford	1991	red	52
Ford	1991	white	9
Ford	1991	blue	55
Ford	1992	red	27
Ford	1992	white	62
Ford	1992	blue	39

CUBE

DATA CUBE

Model	Year	Color	Sales
Chevy	1990	blue	62
Chevy	1990	red	5
Chevy	1990	white	95
Chevy	1990	ALL	154
Chevy	1991	blue	49
Chevy	1991	red	54
Chevy	1991	white	95
Chevy	1991	ALL	198
Chevy	1992	blue	71
Chevy	1992	red	31
Chevy	1992	white	54
Chevy	1992	ALL	156
Chevy	ALL	blue	182
Chevy	ALL	red	90
Chevy	ALL	white	236
Chevy	ALL	ALL	508
Ford	1990	blue	63
Ford	1990	red	64
Ford	1990	white	62
Ford	1990	ALL	189
Ford	1991	blue	55
Ford	1991	red	52
Ford	1991	white	9
Ford	1991	ALL	116
Ford	1992	blue	39
Ford	1992	red	27
Ford	1992	white	62
Ford	1992	ALL	128
Ford	ALL	blue	157
Ford	ALL	red	143
Ford	ALL	white	133
Ford	ALL	ALL	433
ALL	1990	blue	125
ALL	1990	red	69
ALL	1990	white	149
ALL	1990	ALL	343
ALL	1991	blue	106
ALL	1991	red	104
ALL	1991	white	110
ALL	1991	ALL	314
ALL	1992	blue	110
ALL	1992	red	58
ALL	1992	white	116
ALL	1992	ALL	284
ALL	ALL	blue	339
ALL	ALL	red	233
ALL	ALL	white	369
ALL	ALL	ALL	941

Figure 1: A cube of SUM measure.

Figure 1 shows a clearly process to cube a three-dimension table with the method of SUM. Every attributes are grouping by hierarchy, which makes the data more readable and significant.

1.3 Batch Cube

Because of the multi-dimensions, many attributes will be splited into a varity of combinations. And it is followed with a new issue, exponential data. Every piece will transmit the whole data, N attributes will generate even 2^N times' data.

In order to contrive to reduce the data size of transmission, **Batch Cube** is a nice choice. Batch cube is a stratgy to split into a part of combinations in **Mapper**, and then, postprocess those data in **Reducer**

1.4 Top-down Approach

Top-down and bottom-up are both strategies of information processing and knowledge ordering, used in a variety of fields including software, humanistic and scientific theories, and management and organization. In practice, they can be seen as a style of thinking and teaching.

A top-down approach (also known as stepwise design and in some cases used as a synonym of decomposition) is essentially the breaking down of a system to gain insight into its compositional sub-systems. In a top-down approach an overview of the system is formulated, specifying but not detailing any first-level subsystems. Each subsystem is then refined in yet greater detail, sometimes in many additional subsystem levels, until the entire specification is reduced to base elements.

Top down approach starts with the big picture. It breaks down from there into smaller segments.

1.5 TeraSort

TeraSort is a standard map/reduce sort, except for a custom partitioner that uses a sorted list of $N - 1$ sampled keys that define the key range for each reduce. Partitioner will compare each line with the $N - 1$ samples, and distribute lines into adapted block. Because of the random samples, the data in those N blocks will be as average as possible. Finally, every blocks of data will be processed in parallel.

2 Algorithm: TSCube

2.1 Estimate

MAP-ESTIMATE(OriginData)

```
1  for line in sys.stdin
2       do Generate a random real number q, q ∈ [0,1]
3          if q < ρ
4             then for each Batch B in Cube
5                       do k = B(e)
6                          EMIT k, e => 1
```

This is the first round of TSCube: Estimate. For the sake of getting the samples, we need to know this formula:

$$\rho = \ln(np)/m$$

where p is the number of partitions, n is the number of total rows of data, and $m = n/p$. So the ρ is the proportion of samples in total data. And q is a random number, so according to probability, it will allow the number of records of proportion of ρ to emit to reducer.

REDUCE-ESTIMATE(KVLIST)

```
1  for line in sys.stdin
2      do Divide KVList into p partitions
3         Calculate the length between two boundaries
4         ▷ len = length(KVList)/p
5           then Every len KVrow
6                EMIT k=>e
```

The KVList is a group of samples, which are random sampled from the total records. If we will divide those original data into p partitions, just select boundaries from that group of samples. Finally, these $p-1$ boundaries will be incredibly uniformly distributed in original data.

2.2 Materialize

MAP-MATERIALIZE(ORIGINDATA, ESTIMATEBOUNDARIES)

```
1  for line in sys.stdin
2      do for each Batch B in Cube
3             do k = B(e)
4                Compare k with p − 1 boundaries
5                EMIT k=>e to partition pk
```

Last round, Estimate generate some boundaries, and in this mapper, we need to EMIT these batches to the partition which they should go. Since this round, it really starts to process original data.

REDUCE-MATERIALIZE(KVLIST)

```
1  for line in sys.stdin
2      do Top − DownApproach
3         EMIT k=>{v1,v2...}
```

Each record is processed in an established measure with Top-Down Approach. And top-down just divides every transmitted batch into detailed segments. And then emit them.

2.3 Postprocess

MAP-POSTPROCESS(KVGROUP)

```
1  for line in sys.stdin
2      do EMIT k=>{v1,v2...}
```

Just emit to reducer.

```
REDUCE-POSTPROCESS(KVGROUP)
1  for line in sys.stdin
2       do Merge {v1,v2...}
3          EMIT <Group,Vfinal>
```

Because of each reducer process part of data, some values with same keys are not be merged. So in this round, all records which possess the same keys will be merged into an aggregate, and then get the final result.

3 Analysis

3.1 Why using Hadoop?

HDFS is a Map/Reduce system, actually it just a system that need human intelligence to design an approach manually, and it does nothing to optimize the method.

Like Merge Sort, divides data, hand out to subsystem and finally merge them into an aggregate. The design philosophy is related to divide and conquer. Each subsystem just responsible to a part of data, and they can solve them lightly and concurrently.

3.2 Why using Batch Cube?

Batch, as the term suggests, data is divided into a few batches, and then transmit to the next step. Originally, we should split these data into different combinations, however, the size of data is too large, too much depletion is in the process of transmission. For the sake of efficiency, only transfer a few batches, then, in reducers, post process will follow.

3.3 Why using Top-Down Approach?

Top-Down is a method to process the data from the high-dimensions down to the low-dimensions, and for the low-dimensions, we only need to process data from the high-dimensions, and do not solve it as much as the high-dimensions, just follow the predecessors.

3.4 Why using TeraSort and KeyFieldBasedPartitioner?

The most typical application for Terasort is sorting in big data field. But in this project, we use Terasort to improve the performance of hadoop jobs.

As we take MapReduce into implement, it cannot be ignored the fact that, because of the unfair distribution of tasks, the reduce tasks are not always finished at the same time. The problem is that we can do nothing before all reduce tasks being finished, and thus any extreme reduce task will degrade the performance of hadoop greatly.

To solve this problem, we introduce Terasort, through which we can achieve the goal that there is a relatively balanced workload of each reduce task. In brief, all data

is devided into t partitions equally in map tasks, and then each reduce task takes charge of only one partition. The two key points of Terasort are as follows:

3.4.1 Sampling

The aim of sampling is to ensure that then in the partition process all data can be distributed uniformly. It can be proved that when the sampling proportion $\rho = ln(np)/m$, the algorithm has a best performance. In this project, for each piece of input data, we generate a random decimal q between 0 and 1, if $q < \rho$, then we select this piece of data, or we just ignore it. In this way, based on the principle of probability, when a map task is finished, we just retain all of its input data with a percentage of ρ. All map tasks are the same, and then all output data of map tasks are mixed and sorted during the shuffle process. The ordered data are all delivered to the same reduce task, and in this case we use the means of interval sampling to select the required amount of data(In this project, the number is t). But here a premise is that the total number of data is known and this is the reason why at the end of each map task, the total number of printed lines is delivered to the reduce task.

3.4.2 Partition

As mentioned before, we have gotten $t - 1$ boundaries in ascending order, so compare each piece of input data to all these boundaries, we know which partition it belongs to. In this project, we adopt the means of binarysearch to improve performance on assumption of a large amount of boundaries. But how can a piece of data be sent to the desirable partition? It is KeyFieldBasedPartitioner. To do that, we set a new field before the key of each piece of data seperated by $'.'$. For example, the original $< k, v >$ belonging to the fifth partition becomes $< 4.k, v >$ where $4 = 5 - 1$, also $< k, v >$ belonging to the sencond partition becomes $< 1.k, v >$. On the other hand, we specify this field as the basis of partition process by the arguments of $-D\ map.output.key.field.separator =' .'$ and $-D\ num.key.fields.for.partirion = 1$. Because the ASCII of value in this field is adjacent, we believe that different values can be mapped to different partitions after the effect of *hashcode*() function and modulo operation. In this way, each partition has a approximately equal amount of data and thus each reduce task has a balanced workload.

4 Impressions

Song: The most essential but brief thing that I realized is balance. Also, we can call that uniform. Batch may reduce the transmission of data records, however, TeraSort can make the whole data records be processed uniformly and distributed every balanced pieces in reducers. It is a very easy method, also is an important idea. Make each reducers process uniform numbers of data records, and then no extra time to wait each other. A core thought is that there is no perfect idea or algorithm, but there exist a more adapted idea which can fulfill our final design.

Ruan: Before coding, we tried our best to understand the second MapReduce processes, but we got stuck in the map process about how to obtain the partition pk that k falls in. In our understanding, partition is done after map process, so how can we achieve it in the map process? Also, we found that TotalOrderPartitioner is an ideal

stratgy instead of KeyFieldBasedPartitioner. Unfortunately, later we realized that there is not hadoop interface for python to achieve TotalOrderPartitioner, and thus we felt it reasonable to adopt KeyFieldBasedPartitioner. Only then did we actually realize the sinificant differences between theory and reality. However, in this case, another confusing problem is we don't know whether the hash function can map different values to different partitions or not. Anyhow, finally we found it actually achieves the goal.

5 References

[1] Jeffrey Dean and Sanjay Ghemawat. *MapReduce: Simplied Data Processing on Large Clusters*. Communications of the ACM-50th anniversary issue: 1958-2008

[2] Jim Gray, Surajit Chaudhuri, Adam Bosworth, Andrew Layman, Don Reichart, Murali Venkatrao, Frank Pellow, and Hamid Pirahesh. *Data cube: A relational aggregation operator generalizing group-by, cross-tab, and subtotals*. Data Min. Knowl. Discov., 1(1):29–53, January 1997.

[3] Arnab Nandi, Cong Yu, Philip Bohannon, and Raghu Ramakrishnan. *Data cube materialization and mining over mapreduce*. IEEE Transactions on Knowledge and Data Engineering, 24(10):1747–1759, 2012.

[4] Yufei Tao, Wenqing Lin, and Xiaokui Xiao. *Minimal mapreduce algorithms*. In Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data, SIGMOD '13, pages 529–540, New York, NY, USA, 2013. ACM.

[5] Owen O'Malley. *TeraByte Sort on Apache Hadoop*. Yahoo!, May 2008.