Optimizing Cloud MapReduce for Processing Stream Data using Pipelining

作者: Rutvik Karve, Devendra Dahiphale, Amit Chhajer

報告: 饒展榕
OUTLINE

• INTRODUCTION
• LITERATURE SURVEY
• OUR PROPOSED ARCHITECTURE
• ADVANTAGES, FEATURES AND APPLICATIONS
• CONCLUSIONS AND FUTURE WORK
INTRODUCTION

• Cloud MapReduce (CMR) is a framework for processing large data sets of batch data in cloud.
• The Map and Reduce phases run sequentially, one after another.
• This leads to:
  1. Compulsory batch processing
  2. No parallelization of the map and reduce phases
  3. Increased delays.
We propose a novel architecture to support streaming data as input using pipelining between the Map and Reduce phases in CMR, ensuring that the output of the Map phase is made available to the Reduce phase as soon as it is produced.
• This ‘Pipelined MapReduce’ approach leads to increased parallelism between the Map and Reduce phases; thereby

1. Supporting streaming data as input
2. Reducing delays
3. Enabling the user to take ‘snapshots’ of the approximate output generated in a stipulated time frame.
4. Supporting cascaded MapReduce jobs.
A. *MapReduce*

- The model consists of two phases: a Map phase and a Reduce phase.
- Initially, the data is divided into smaller 'splits'.
- These splits of data are independent of each other and can hence be processed in parallel fashion.
- Each split consists of a set of (key, value) pairs that form records.
B. Hadoop

• Hadoop is an implementation of the MapReduce programming model developed by Apache.

• The Hadoop framework is used for batch processing of large data sets on a physical cluster of machines.
• It incorporates a distributed file system called Hadoop Distributed File System (HDFS), a Common set of commands, scheduler, and the MapReduce evaluation framework.

• Hadoop is popular for processing huge data sets, especially in social networking, targeted advertisements, internet log processing etc.
C. Cloud MapReduce

- Cloud MapReduce [2] is a light-weight implementation of MapReduce programming model on top of the Amazon cloud OS, using Amazon EC2 instances.
• The architecture of CMR, as described in [2] consists of one input queue, multiple reduce queues which act as staging areas for holding the intermediate (key, value) pairs produced by the Mappers, a master reduce queue that holds the pointers to the reduce queues, and an output queue that holds the final results.
D. Online MapReduce (Hadoop Online Prototype)

- HOP is a modification to traditional Hadoop framework that incorporates pipelining between the Map and Reduce phases, thereby supporting parallelism between these phases, and providing support for processing streaming data.
• A downstream dataflow element can begin processing before an upstream producer finishes.

• It carries out online aggregation of data to produce incrementally correct output.
OUR PROPOSED ARCHITECTURE

• Our proposal aims at bridging this gap between heavyweight HOP and the light-weight, scalable Cloud MapReduce implementation, by providing support for processing stream data in Cloud MapReduce.
• The challenges involved in the implementation include:
• 1. Providing support for streaming data at input
• 2. A novel design for output aggregation
• 3. Handling Reducer failures
• 4. Handling windows based on timestamps.
1) The first design option:

- As in CMR, the Mapper pushes each intermediate (key, value) pair to one of the reduce queues based on hash value of the intermediate key.
- The hash function can be user-defined or a default function provided, as in CMR.
2) *The second design option:*

- Alternatively, we can have a single queue between the Mappers and the Reducers, with all the intermediate (key, value) pairs generated by all the Mappers pushed to this intermediate queue.
• Reducers poll this IntermediateQueue for records.
• Aggregation is carried out as follows:
• There are a fixed number of Output splits.
• Whenever a Reducer reads a (key, value) record from the IntermediateQueue, it applies a user-defined Reduce function to the record, to produce an output (key, value) pair.
• It then selects an output split by hashing on the output key produced.
3) **Hybrid Approach**

- Have multiple ReduceQueues, each linked statically to a particular Reducer, but instead of linking the output splits to the Reducer statically, use hashing on the output key of the (key, value) pair generated by the Reducer to select an output split.
• This will require fewer changes to the existing CMR architecture, but will involve static linking of Reducers to ReduceQueues.
ADVANTAGES, FEATURES AND APPLICATIONS

A. Advantages

1. Either design allows Reducers to start processing as soon as data is made available by the Mappers. This allows parallelism between the Map and Reduce phases.

2. A downstream processing element can start processing as soon as some data is available from an upstream element.

3. The network is better utilized as data is continuously pushed from one phase to the next.

4. The final output is computed incrementally.

5. Introduction of a pipeline between the Reduce phase of one job and the Map phase of the next job will support Cascaded MapReduce jobs.
B. Features

1) Time windows
2) Snapshots
3) Cascaded MapReduceJobs
C. Applications

• With these features, the design is particularly suited to stream processing of data.

• Typically analysis and processing of web feeds, click-streams, micro-blogging, and stock market quotes are some of the popular and typical stream processing applications.
CONCLUSIONS AND FUTURE WORK

• It is also inherently scalable as it is cloud-based.
• This also gives it a „light-weight“ nature, as the handling of distributed resources is done by the Cloud OS.
• Future work will include supporting rolling windows for obtaining outputs of arbitrary time-intervals of the input stream.