Google's MapReduce Programming Model - Revisited (by Ralf Lammel)

Diogo Pratas

MAPi doctoral program
Towards a Linear Algebra of Programming
Review Article

Thematic Seminar

Outline

Outline

Introduction

- Google's **MapReduce** is a programming model for processing large data sets in a massively parallel manner.
- The model is inspired by the **map and reduce functions** commonly used in **functional programming**.
- The authors **reverse-engineer** the seminal papers on MapReduce and Sawzall, using the functional programming language **Haskell**, specifically:
  - the basic program **skeleton that underlies MapReduce computations**;
  - the **parallelism opportunities** executing MapReduce computations;
  - the fundamental **characteristics of Sawzall's aggregators** as an advancement of the MapReduce approach;

Outline

MapReduce

- **MapReduce** "abstraction **is inspired by the map and reduce primitives** present in **Lisp** and **many other functional languages**" [2].
- **MapReduce model** is based on the following concepts:
  - iteration over the input;
  - computation of key/value pairs from each piece of input;
  - grouping of all intermediate values by key;
  - iteration over the resulting groups;
  - reduction of each group;

MapReduce

- **Map**
  - **Perform a function on individual values in a data set to create a new list of values**
    Example: square x = x * x
    map square [1,2,3,4,5]
    returns [1,4,9,16,25]

- **Reduce**
  - **Combine values in a data set to create a new value**
    Example: sum = (each element in the array, total +=)
    reduce [1,2,3,4,5]
    returns 15 (the sum of the elements)

MapReduce

- Find all pages that link to a certain page
- **Map Function**
  - Outputs <**target, source**> pairs for each link to a target URL found in a source page;
  - For each page we know what pages it links to
- **Reduce Function**
  - Concatenates the list of all source URLs associated with a given target URL and emits the pair: <**target, list(source)**>;
  - For a given web page, we know what pages link to it.

MapReduce

The computation takes a set of **input key/value pairs**, and produces a set of **output key/value pairs**. The user of the MapReduce library expresses the computation as two functions: map and reduce:

- **Map**, written by the user, takes an input pair and produces a set of intermediate key/value pairs. The MapReduce library **groups together all intermediate values** associated with the same intermediate key I and passes them to the reduce function.
- **map(inKey, inValue)** $- >$ **(outKey, intermediateValue) list**

MapReduce

- **Reduce**, written by the user, accepts an intermediate key I and a set of values for that key. It **merges together these values to form a possibly smaller set of values**. Typically just zero or one output value is produced per reduce invocation.

- The intermediate values are supplied to the user's reduce function via an iterator, allowing handle lists of values that are too large to fit in memory.

- **reduce(outKey, intermediateValue list)** $->$ **outValue list**

- Formalizing: $(|\mathbf{r}|).(\mathbf{mapF})$

Outline

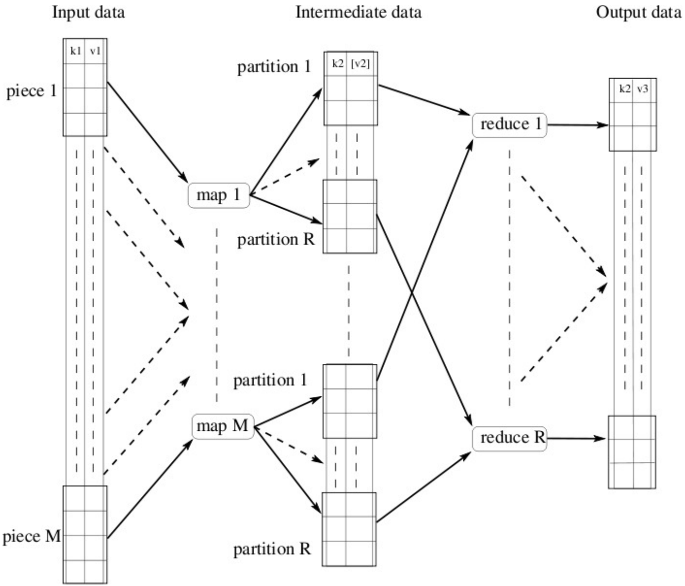
Parallel MapReduce computations

- The **programming model readily enables parallelism**, and the MapReduce implementation takes care of the complex details of distribution such as **load balancing, network performance and fault tolerance**.

- The programmer has to provide **parameters for controlling distribution and parallelism**, such as the number of reduce tasks to be used. Defaults for the control parameters may be inferable.

- the next figure presents the strategy for distributed execution...

Parallel MapReduce computations



Outline

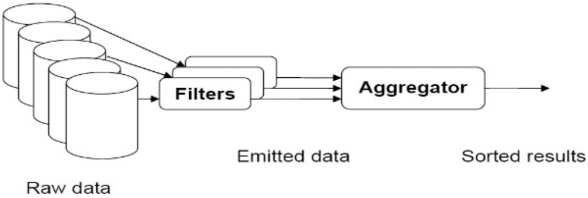
Sawzall

- **Sawzall** is a procedural domain-specific programming language, used by **Google to process large numbers of individual log records**.
- Built on **top of MapReduce**.
- Sawzall runs in the **map phase**.
- **Output** of map phase is **data items for aggregators**.

```
count: table sum of int;
total: table sum of float;
sumOfSquares: table sum of float;
x: float = input
emit count <- 1
emit total <- x
emit sumOfSquares <- x * x
```

Sawzall program will read the input and produce three results: the **number of records, the sum of the values, and the sum of the squares of the values**.

Sawzall



- emit - sends data to external aggregator;
- Drawing line between filtering and aggregating enables **high degree of parallelism**;
- Collection, Sample, Sum, Maximum, Quantile, Top, Unique;
- Possible to process data as part of mapping phase (ex sum);
- Possible to index aggregators;
- Creates a distinct aggregator for each unique value of index;

Outline

Summary

- **MapReduce** and **Sawzall** is one of the best examples of the power of **functional programming**, to **list processing** in particular.
- The authors used functional programming language (Haskell) for the discovery of a rigorous description of the **MapReduce programming model** and its advancement as the domain-specific language **Sawzall**.
- The authors have shown the model is stunningly **simple**, **robust**, and effectively **supports parallelism**.
- As a side effect, it was presented general illustration for the utility of functional programming in a semi-formal approach to design with excellent **support for executable specification**.
- This illustration may motivate others to deploy **functional programming for their future projects**.

Outline

Final considerations

References

1 M.M. Fokkinga. Mapreduce — a two-page explanation for laymen. Unpublished Technical Report, 2008.

2 J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In OSDI'04, 6th Symposium on Operating Systems Design and Implementation, Sponsored by USENIX, in cooperation with ACM SIGOPS, pages 137–150, 2004.

3 R. Pike, S. Dorward, R. Griesemer, and S. Quinlan. Interpreting the data: Parallel analysis with Sawzall. Scientific Programming, 14, Sept. 2006. Special Issue: Dynamic Grids and Worldwide Computing.

Final considerations

Thank you !

pratas@ua.pt