Concepts and Models of Parallel and Data-centric Programming

MapReduce Design Patterns – Summarization Patterns

Lecture, Summer 2020

Simon Schwitanski
Dr. Christian Terboven

Outline

Numerical Summarization

- Calculation of aggregate statistical values
  - Counting, Minimum, Maximum, Average, Standard Deviation, …
- Intent: Group records by *key* field, calculate aggregate value per group
- Motivation
  - Large datasets difficult to interpret and evaluate manually by a human
  - Numerical summarization gives abstracted overview of data
- Conditions for application
  - Data is numerical (or should be counted)
  - Data can be grouped by specific (key) fields

Numerical Summarization – Structure

- Mapper outputs key(s) to group by and values that should be aggregated (“summary field”)
- Reducer applies desired aggregation function to group values

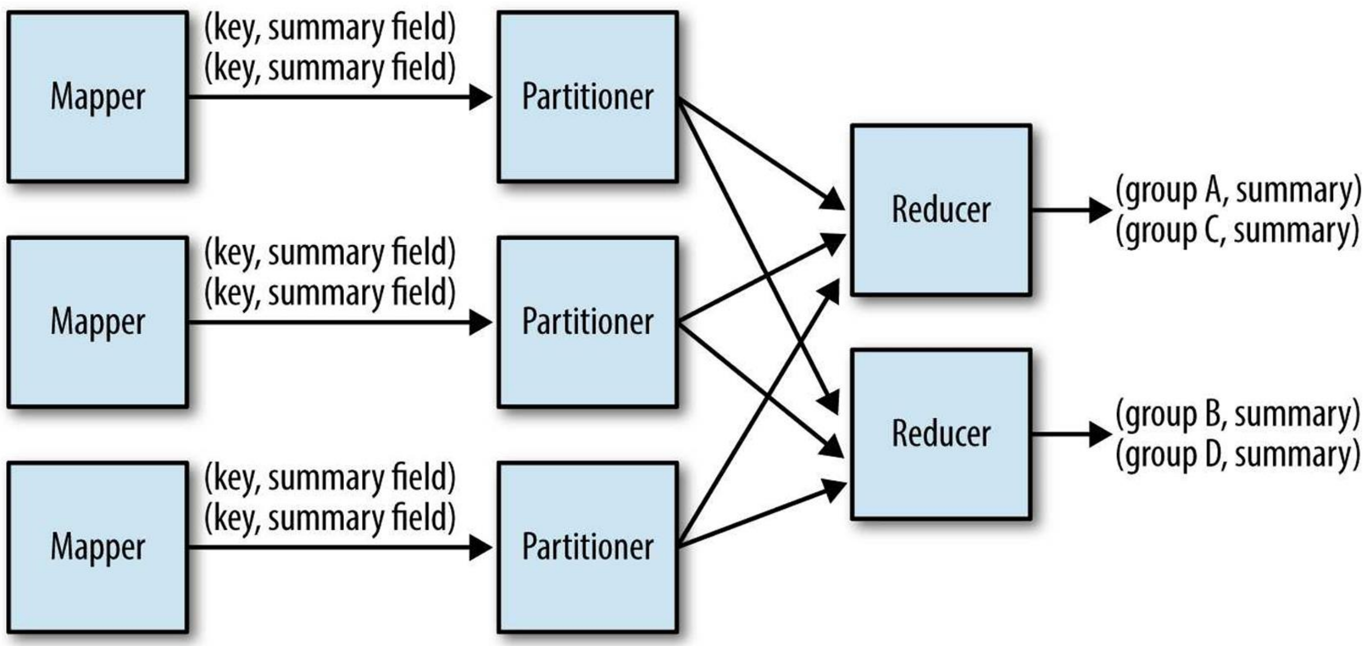


Illustration: Miner, Donald and Shook, Adam. “MapReduce Design Patterns: Building Effective Algorithms and Analytics for Hadoop and Other Systems”, p.16, O'Reilly Media, 2012

Numerical Summarization – Applications

- Word count
  - As previously discussed
- Record count
  - Count number of items in a certain interval (number of user comments per week etc.)
- Minimum, maximum and count of some particular event
- Average, median, standard deviation of numerical data
- Note: Numerical summarization similar to following SQL statement

```
SELECT MYFUNC(mycol) FROM mytable GROUP BY groupcol;
```

where `MYFUNC` is the desired aggregate function.

Numerical Summarization – Performance

- Perfectly suited for MapReduce (primarily designed for summarizations)
- Using a Combiner is crucial for large datasets
- Bottleneck can occur at Reducer
  - Imbalances in reduced data (much more intermediate data with a certain key than other keys) can lead to load imbalances

Working Data Set (1)

- Stack Exchange provides dump of posts, comments etc.
- Comments.xml

```
<row Id="6" PostId="13" Score="0" Text="Honestly, i was just about to ask
this!" CreationDate="2014-02-11T23:46:29.183" UserId="8" />
```

- Posts.xml

```
<row Id="1" PostTypeId="1" AcceptedAnswerId="2" CreationDate="2014-02-
11T22:24:09.530" Score="42" ViewCount="5664" Body="I'm making a simple
Arduino web server [...]" OwnerUserId="3" Title="Is an Arduino capable of
running 24/7?" AnswerCount="11" CommentCount="2" FavoriteCount="9" />

<row Id="2" PostTypeId="2" ParentId="1" CreationDate="2014-02-
11T22:36:57.700" Score="57" Body="You shouldn't have any issues keeping
it on all the time [...]" OwnerUserId="11" CommentCount="7" />
```

Working Data Set (2)

- Users.xml

```
<row Id="9" Reputation="131" CreationDate="2014-02-11T22:31:15.467"
DisplayName="orangeocelot" LastAccessDate="2016-08-05T14:39:40.290"
Location="Ireland" Views="4" UpVotes="7" DownVotes="0"
AccountId="1391850" />
```

- Data is parsed in the examples using a helper function

```
public class MRDPUtils {
        public static Map<String, String>
            transformXmlToMap(String xml) {...}
}
```

- First entry of map pair is the key, second entry the value

Numerical Summarization – Minimum Example (1)

- **Problem:** Given a list of user comments, determine for each user the earliest time he commented.
- Structure of comments:

```
<row Id="6" PostId="13" Score="0" Text="Honestly, i was just about to ask
this!" CreationDate="2014-02-11T23:46:29.183" UserId="8" />
```

- One KV pair for each comment

Pseudocode:

```
map(String docid, String comment):
    parsed = transformToMap(comment)
    Emit(parsed.UserId, timestamp(parsed.CreationDate))

reduce(String userId, Iterator values):
    long min = MAX_LONG
    for each timestamp ts in values:
        if ts < min:
            min = ts
    Emit(userId, min)
```

Numerical Summarization – Minimum Example (2)

- Structure of comments:

```
<row Id="6" PostId="13" Score="0" Text="Honestly, i was just about to ask
this!" CreationDate="2014-02-11T23:46:29.183" UserId="8" />
```

Mapper:

```java
public static class EarliestCommentMapper extends
            Mapper<Object, Text, Text, LongWritable> {
    // Output key and value Writable
    private Text outUserId = new Text();
    private LongWritable outDateLong = new LongWritable();

    // This object will format the creation date string into a Date object
    private final static SimpleDateFormat frmt = new SimpleDateFormat(
            "yyyy-MM-dd'T'HH:mm:ss.SSS");
```

Numerical Summarization – Minimum Example (3)

Mapper (continued):

```
    @Override
    public void map(Object key, Text value, Context context)
            throws IOException, InterruptedException {
        // Parse the input string into a nice map
        Map<String, String> parsed =
            MRDPUtils.transformXmlToMap(value.toString());

        String userId = parsed.get("UserId");    // extract user id
        String strDate = parsed.get("CreationDate"); // extract creation date
        Date creationDate = frmt.parse(strDate); // translate to Date object

        // Write out the user ID with creation date (as long)
        outUserId.set(userId);
        outDateLong.set(creationDate.getTime());
        context.write(outUserId, outDateLong);
    }
}
```

Numerical Summarization – Minimum Example (4)

Reducer:

```java
public static class EarliestCommentReducer extends
        Reducer<Text, LongWritable, Text, LongWritable> {
    private LongWritable result = new LongWritable();

    @Override
    public void reduce(Text key, Iterable<LongWritable> values,
                       Context context)
            throws IOException, InterruptedException {
        // Set initial value
        long min = Long.MAX_VALUE;

        // Iterate through all input values for this key
        for (LongWritable val : values) {
            long longValue = val.get();
            if (longValue < min)
                min = longValue;
        }

        // Write out result
        result.set(min);
        context.write(key, result);
}}
```

Numerical Summarization – Minimum Example (5)

Job configuration:

```java
public static void main(String[] args) throws Exception {
  Configuration conf = new Configuration();

  Job job = Job.getInstance(conf, "StackOverflow Comment Date Min");
  job.setJarByClass(EarliestCommentDate.class);

  job.setMapperClass(EarliestCommentMapper.class);
  job.setReducerClass(EarliestCommentReducer.class);
  job.setOutputKeyClass(Text.class);
  job.setOutputValueClass(LongWritable.class);
  FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
  FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));
  System.exit(job.waitForCompletion(true) ? 0 : 1);
}
```

Numerical Summarization – Running on Cluster

- “Our” Hadoop cluster 2018: 11 CLAIX-2016-MPI nodes
  - 1 master, 10 workers
  - 2 x Intel Xeon E5-2650v4, 12 cores each
  - 128 GB RAM per node
  - Interconnect: Intel Omni-Path, up to 100 Gbit/s network bandwidth
  - Access to files via Lustre file system, no local storage → 2020: Local storage
- You will submit and run example tasks on our custom Hadoop cluster in the exercises.
- Dataset: All comments on StackOverflow (XML file)
  - Entries: 66,432,644
  - Size: 17 GB

Numerical Summarization – Running on Cluster

Output Data:

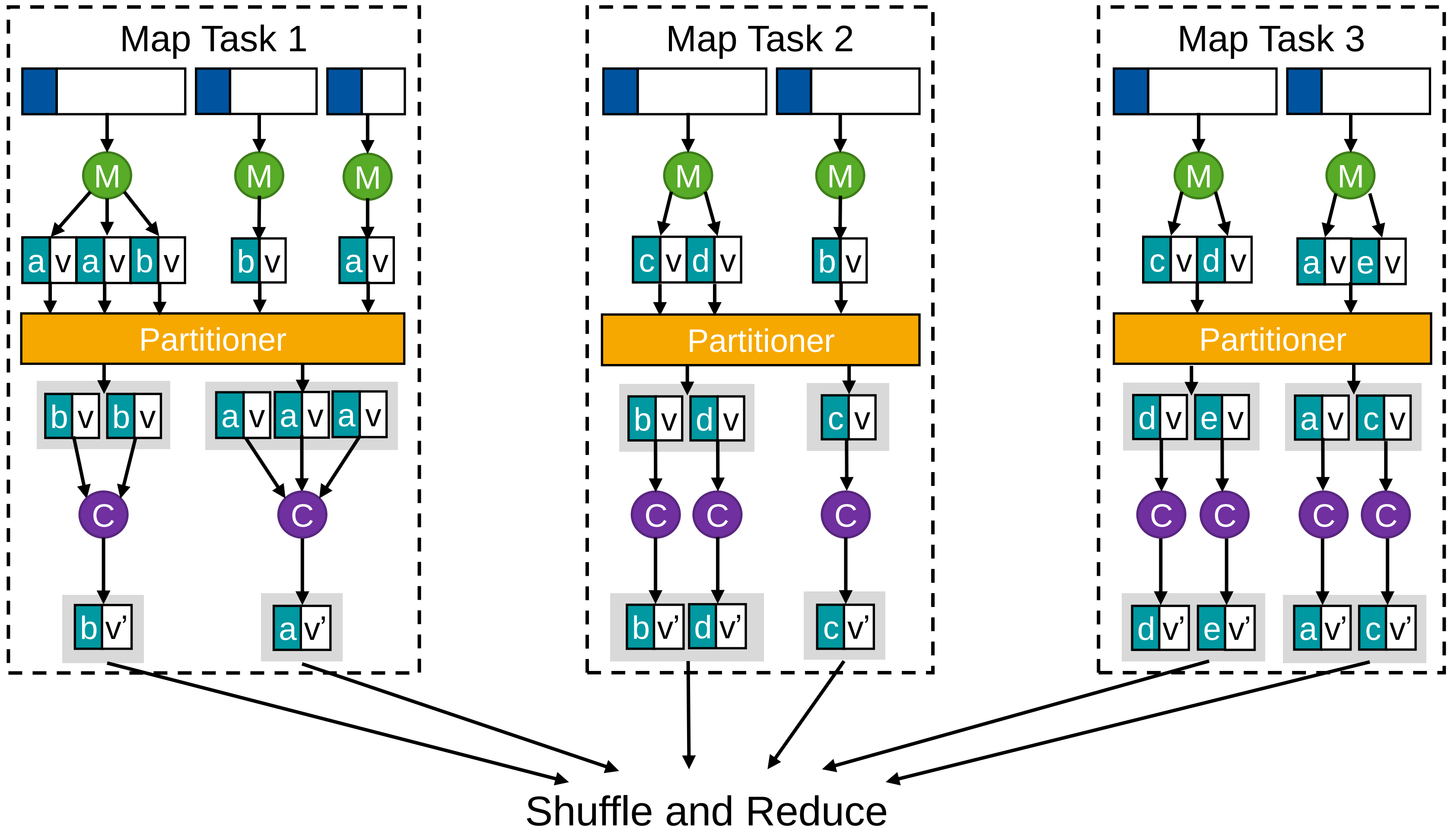
User ID	Earliest Comment (Timestamp)
10	1392258686390
10003	1433121838880
10007	1432600989783
10017	1431534771737
…	…

Output Log:

```
Map input records=66432644
Map output records=65722799
Map output bytes=1019807085
Map output materialized bytes=1151291983
Input split bytes=15458
Combine input records=0
Combine output records=0
Reduce input groups=2502332
Reduce shuffle bytes=1151291983 ≈ 1 GB
Reduce input records=65722799
Reduce output records=2502332
```

Illustration adapted from Dean, Jeffrey, and Sanjay Ghemawat. "MapReduce: simplified data processing on large clusters." *Communications of the ACM* 51.1 (2008): 107-113.

Combiner Function – Reminder



$$Combine(k_2, list(v_2)) \rightarrow list(k_2, v_2)$$

Numerical Summarization – Minimum with Combiner

- Job configuration:

```java
public static void main(String[] args) throws Exception {
  Configuration conf = new Configuration();

  Job job = Job.getInstance(conf, "StackOverflow Comment Date Min");
  job.setJarByClass(EarliestCommentDate.class);

  job.setMapperClass(EarliestCommentMapper.class);
  job.setCombinerClass(EarliestCommentReducer.class);
  job.setReducerClass(EarliestCommentReducer.class);
  job.setOutputKeyClass(Text.class);
  job.setOutputValueClass(LongWritable.class);
  FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
  FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));
  System.exit(job.waitForCompletion(true) ? 0 : 1);
}
```

Numerical Summarization – Running on Cluster

Output Log without Combiner:

```
Map input records=66432644
Map output records=65722799
Map output bytes=1019807085
Map output materialized bytes=1151291983
Input split bytes=15458
Combine input records=0
Combine output records=0
Reduce input groups=2502332
Reduce shuffle bytes=1151291983 ≈ 1 GB
Reduce input records=65722799
Reduce output records=2502332
```

Output Log with Combiner:

```
Map input records=66432644
Map output records=65722799
Map output bytes=1019807085
Map output materialized bytes=204036720
Input split bytes=15458
Combine input records=65722799
Combine output records=11569585
Reduce input groups=2502332
Reduce shuffle bytes=204036720 ≈ 200 MB
Reduce input records=11569585
Reduce output records=2502332
```

- Network I/O reduced by factor 5 due to combiner
- Effort: Adding one LOC (set combiner in job configuration)
- However: Nearly no impact on runtime
  - Reason: Network interconnect extremely fast, network I/O makes no difference for 200 MB / 1 GB, data set too small

Numerical Summarization – Average (1)

- Calculation of minimum, maximum and count simple
- Can we do the same with a measure like average?
- First answer: Yes!
- **Example:** Get average comment length posted per hour of day.

```
map(String docid, String comment):
    parsed = transformToMap(comment)
    Emit(getHour(parsed.CreationDate), length(parsed.Text))

reduce(String hour, Iterator values):
    int count = 0
    int sum = 0
    for each value v in values
        count++
        sum += v
    Emit(hour, ((float) sum) / count)
```

- Can we use this Reducer as Combiner for average calculation?

Numerical Summarization – Average (2)

- No, because the average operation is *not associative*.
- Average of two values: $avg(a,b) = \frac{a+b}{2}$
- Associativity: $avg(avg(a,b),c) = avg(a,avg(b,c))$
- Counterexample: $a = 2, b = 6, c = 1$
  - $avg(avg(2,6),10) = avg(4,10) = 7$
  - $avg\big(2, avg(6,10)\big) = avg(2,8) = 5$
- Thus: Associativity **not** fulfilled for average operation.
- Using the previously defined Reducer as Combiner will not work.

Numerical Summarization – Average (3)

- Average operation for a set D of values: $\mathrm{avg}(D) = \frac{sum(D)}{count(D)}$
- Idea: Mapper and Reducer output *count* sent along with *average* of values → Reproducing *sum* for recomputing *average* possible:

$$\mathrm{sum}(D) = avg(D) \cdot count(D)$$

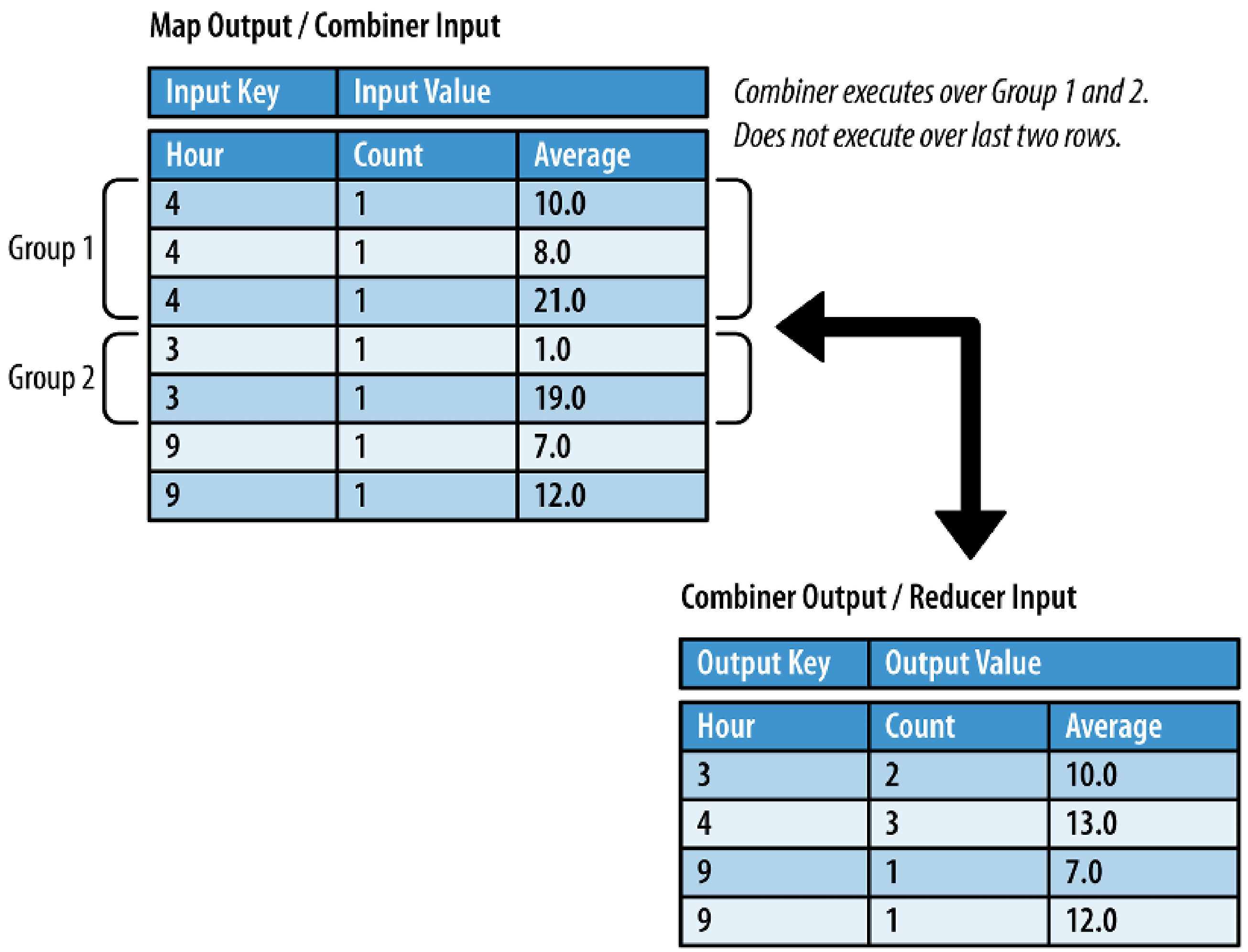
```
map(String docid, String comment):
    parsed = transformToMap(comment)
    Emit(getHour(parsed.CreationDate), new Pair(1, length(parsed.Text)))

reduce(String hour, Iterator values):
    float sum = 0;
    float count = 0;
    for each pair (countold, average) in values:
        sum += countold * average
        count += countold
    Emit(hour, new Pair(count, sum / count))
```

Reproduce *sum* using "previous" *count* and *average*

- Now, Reducer can be also used as Combiner.

Numerical Summarization – Average (4)



Map Output / Combiner Input

Input Key	Input Value	
Hour	Count	Average
4	1	10.0
4	1	8.0
4	1	21.0
3	1	1.0
3	1	19.0
9	1	7.0
9	1	12.0

Combiner executes over Group 1 and 2.
Does not execute over last two rows.

Combiner Output / Reducer Input

Output Key	Output Value	
Hour	Count	Average
3	2	10.0
4	3	13.0
9	1	7.0
9	1	12.0

Illustration: Miner, Donald and Shook, Adam. “MapReduce Design Patterns: Building Effective Algorithms and Analytics for Hadoop and Other Systems”, p.22, O'Reilly Media, 2012