Sorting data with MapReduce

Partial, Total and Secondary Sorting

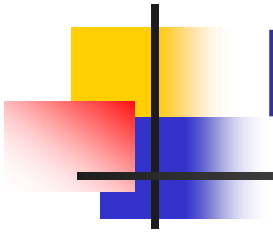
ISEL
INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Partial sorting

Partial sorting

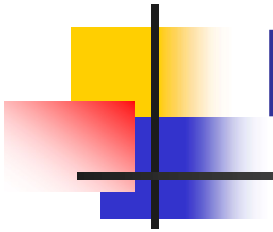
- By default MapReduce results are sorted using the key. The following output refers to the classic "word count"

```
usermr@hadoop: ~/examples/java/Ex01-b
Result sorted by key - MapReduce defaults - (first 5 lines)
hadoop fs -text file:///home/usermr/examples/output/gutenberg/part-r-00000 2>/dev/null | head -n 5
"AS-IS".        8
"Brittain."     2
"Brittish",     1
"Defects".      8
"Defects,"      2

Result sorted (by value) using the linux sort command
hadoop fs -text file:///home/usermr/examples/output/gutenberg/part-r-00000 2>/dev/null | sort -k 2,2 -n -r | head -n 5
the     2700
of      1981
and     1300
to      1282
in      731
usermr@hadoop:~/examples/java/Ex01-b$
```



Partial sorting

- The sort order for keys is controlled by an instance of a `RawComparator`
  1. `mapreduce.job.output.key.comparator.class` property set (explicitly or by calling `setSortComparatorClass()`) is used to create an instance of a `RawComparator`
  2. Else, keys must be a subclass of `WritableComparable`, and the registered comparator for the key class is used
  3. If there is no registered comparator, then a `RawComparator` is used



Partial sorting

- By default, Hadoop assumes a single reducer. This behaviour can be overridden: programmatically `setNumReduceTasks()` or using the property `mapreduce.job.reduces`

- If more then one reducer is used, each output is sorted (by the mappers). However, there is no easy way to combine the files (by concatenation, for example, in the case of plain-text files) to produce a globally sorted file

Partial sorting

usermr@hadoop: ~/examples/java/Ex01-b

```
Result sorted by key - MapReduce defaults - (first 5 lines)
hadoop fs -text file:///home/usermr/examples/output/gutenberg/part-r-00000 2>/dev/null | head -n 5
"AS-IS".        8
"Brittain."     2
"Brittish",     1
"I      2
"More   1

Result sorted (by value) using the linux sort command
hadoop fs -text file:///home/usermr/examples/output/gutenberg/part-r-00000 2>/dev/null | sort -k 2,2 -n -r | head -n 5
the     2700
of      1981
and     1300
to      1282
in      731

Result sorted by key - MapReduce defaults - (first 5 lines)
hadoop fs -text file:///home/usermr/examples/output/gutenberg/part-r-00001 2>/dev/null | head -n 5
"Defects".      8
"Defects,"      2
"House" 2
"Information    2
"Plain  4

Result sorted (by value) using the linux sort command
hadoop fs -text file:///home/usermr/examples/output/gutenberg/part-r-00001 2>/dev/null | sort -k 2,2 -n -r | head -n 5
shall   470
this    402
is      334
any     330
as      258
usermr@hadoop:~/examples/java/Ex01-b$
```

ISEL INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Partial sorting

- The following example shows a MapReduce application capable of filter the input used in the Temperatures examples
- The example removes from the input dataset all the records that do not contain a valid temperature
- The output is stored in sequence files in the format:
  - `<Temperature (float)><Temperature record (Text)>`
- The number of reducers is zero (no reducers used)

Partial sorting


```
public class CleanerMapper
        extends Mapper<LongWritable, Text, FloatWritable, Text> {

  private FloatWritable theKey = new FloatWritable();
  private NcdcRecordParser p = new NcdcRecordParser();

  @Override
  public void map(LongWritable key, Text value, Context context)
        throws IOException, InterruptedException {

    p.parse( value );
    if ( p.isValidTemperature() && p.getAirTemperature()<100.0F ){
      theKey.set( p.getAirTemperature() );
      context.write( theKey, value );
    }
  }
}
```

Partial sorting


```
public class CleanerApplication extends Configured implements Tool {

  public static void main(String[] args)
        throws Exception {
    int exitCode = ToolRunner.run( new CleanerApplication(), args);
    System.exit( exitCode );
  }


  @Override
  public int run(String[] args) throws Exception {
    if (args.length != 2) {
      ...
      return -1;
    }
```

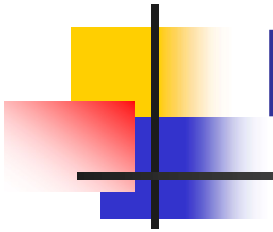
Partial sorting


```
Job job = Job.getInstance( getConf() );

job.setJobName( "Max temperature - Version 5" );
job.setJarByClass( CleanerApplication.class );

FileInputFormat.addInputPath(job, new Path(args[0]) );
FileOutputFormat.setOutputPath(job, new Path(args[1]) );

job.setMapperClass( CleanerMapper.class );
job.setMapOutputKeyClass( FloatWritable.class );
job.setMapOutputValueClass( Text.class );

job.setNumReduceTasks( 0 );
job.setOutputKeyClass( FloatWritable.class );
job.setOutputValueClass( Text.class );
...
```

Partial sorting


```
    // -Dmapreduce.job.outputformat.class=...
    job.setOutputFormatClass( SequenceFileOutputFormat.class );

    // -Dmapreduce.output.fileoutputformat.compress=...
    SequenceFileOutputFormat.setCompressOutput( job, true);

    // -Dmapreduce.output.fileoutputformat.compress.codec=...
    SequenceFileOutputFormat.setOutputCompressorClass( job,
GzipCodec.class );

    // -Dmapreduce.output.fileoutputformat.compress.type=...
    SequenceFileOutputFormat.setOutputCompressionType(job,
CompressionType.BLOCK );

    return job.waitForCompletion(true) ? 0 : 1;
  }
}
```


usermr@hadoop: ~/examples/java/Ex01-a5
Contents of file:///home/usermr/examples/output/temperatures/small/part-m-00000 (1093 entries):
-10.6 0029029810999991901010106004+59500+020350FM-12+002699999V0200501N002119999999N0000001N9-01061+99999102641ADDGF10299199999
-7.2 0029029810999991901010113004+59500+020350FM-12+002699999V0202701N002119999999N0000001N9-00721+99999102671ADDGF10599199999
-8.3 0029029810999991901010120004+59500+020350FM-12+002699999V0202701N007219999999N0000001N9-00831+99999102671ADDGF10599199999
...
2.8 0029029810999991901123106004+59500+020350FM-12+002699999V0201801N008219999999N0000001N9+00281+99999100241ADDGF10899199999
2.8 0029029810999991901123113004+59500+020350FM-12+002699999V0201601N011819999999N0000001N9+00281+99999100031ADDGF10899199999
2.2 0029029810999991901123120004+59500+020350FM-12+002699999V0201401N011819999999N0000001N9+00221+99999099681ADDGF10899199999
Contents of file:///home/usermr/examples/output/temperatures/small/part-m-00001 (1051 entries):
-5.0 0029029810999991903010106004+59500+020350FM-12+002699999V0200501N019019999999N0000001N9-00501+99999099441ADDGF10899199999
-5.0 0029029810999991903010113004+
-2.8 0029029810999991903010120004+
...
1.7 0029029810999991903123106004+
1.1 0029029810999991903123113004+
1.1 0029029810999991903123120004+59500+020350FM-12+002699999V0203601N005119999999N0000001N9+00111+99999
Sorted within each file, but not globally sorted
Equivalent sizes
...
Contents of file:///home/usermr/examples/output/temperatures/small/part-m-00004 (1098 entries):
-20.0 0029228920999991908010106004+60717+028783FM-12+000399999V0202701N001019999999N0000001N9-02001+99999103021ADDGF10899199999
-18.9 0029228920999991908010113004+60717+028783FM-12+000399999V0209991C000019999999N0000001N9-01891+99999103021ADDGF10899199999
-17.2 0029228920999991908010120004+60717+028783FM-12+000399999V0209991C000019999999N0000001N9-01721+99999102961ADDGF10899199999
...
-6.7 0029228920999991908123106004+60717+028783FM-12+000399999V0209991C000019999999N0000001N9-00671+99999104321ADDGF10899199999
-5.6 0035228920999991908123113004+60717+028783FM-12+000399999V0202501N006719999999N0000001N9-00561+99999104431ADDGF10899199999
-6.7 0029228920999991908123120004+60717+028783FM-12+000399999V0202001N006719999999N0000001N9-00671+99999104381ADDGF10899199999
Contents of file:///home/usermr/examples/output/temperatures/small/part-m-00005 (1082 entries):
-16.1 0029029070999991903010106004+64333+023450FM-12+000599999V0200901N004119999999N0000001N9-01611+99999100911ADDGF10699199999
-15.6 0029029070999991903010113004+64333+023450FM-12+000599999V0200901N006219999999N0000001N9-01561+99999100831ADDGF10699199999
-12.2 0029029070999991903010120004+64333+023450FM-12+000599999V0200501N011819999999N0000001N9-01221+99999100401ADDGF10499199999
...
-0.6 0029029070999991903123106004+64333+023450FM-12+000599999V0202701N006219999999N0000001N9-00061+99999101881ADDGF10599199999
0.6 0029029070999991903123113004+64333+023450FM-12+000599999V0202701N006219999999N0000001N9+00061+99999101601ADDGF10599199999
1.1 0029029070999991903123120004+64333+023450FM-12+000599999V0202701N015919999999N0000001N9+00111+99999101401ADDGF10099199999
Total number of entries: 6515



Partial sorting

- In the previous example we had:
  - No reducers
  - The size of each output is similar (input size was similar)
  - The output of each mapper is sorted
- What happens if we use reducers to process the output of the mappers?

Partial sorting


```
public class SortByTemperatureMapper
        extends Mapper<FloatWritable, Text, FloatWritable, Text> {
  @Override
  public void map(FloatWritable key, Text value, Context context)
        throws IOException, InterruptedException {
    context.write( key, value );
  }
}

-----------------------------------------------------------------

public class SortByTemperatureReducer
        extends Reducer<FloatWritable, Text, FloatWritable, Text> {
  @Override
  public void reduce(FloatWritable k, Iterable<Text> vs, Context c)
        throws IOException, InterruptedException {
    for (Text value : vs) {
      context.write(k, value );
    }
  }
}
```

Partial sorting

```
public class SortByTemperatureUsingHashPartitionerApplication
        extends Configured implements Tool {

  public static void main(String[] args) throws Exception {
    int exitCode = ToolRunner.run(
      new SortByTemperatureUsingHashPartitionerApplication(),
      args);
    System.exit( exitCode );
  }

  @Override
  public int run(String[] args) throws Exception {
    if (args.length != 2) {
      ...
      return -1;
    }
    ...
```

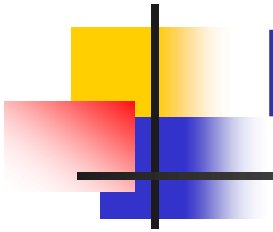
Partial sorting

```
...
Job job = Job.getInstance( getConf() );

job.setJobName( "Max temperature - Version 6" );
job.setJarByClass( SortByTemperatureUsingHashPartitionerApplication.class );

FileInputFormat.addInputPath(job, new Path(args[0]) );
FileOutputFormat.setOutputPath(job, new Path(args[1]) );

job.setInputFormatClass( SequenceFileInputFormat.class );

job.setMapperClass( SortByTemperatureMapper.class );
job.setMapOutputKeyClass( FloatWritable.class );
job.setMapOutputValueClass( Text.class );

...
```

Partial sorting

```
    ...
    job.setOutputFormatClass( SequenceFileOutputFormat.class );
    SequenceFileOutputFormat.setCompressOutput( job, true);
    SequenceFileOutputFormat.setOutputCompressorClass( job, GzipCodec.class );

    SequenceFileOutputFormat.setOutputCompressionType(
        job, CompressionType.BLOCK );

    job.setReducerClass( SortByTemperatureReducer.class );
    job.setOutputKeyClass( FloatWritable.class );
    job.setOutputValueClass( Text.class );

    return job.waitForCompletion(true) ? 0 : 1;
  }
}
```

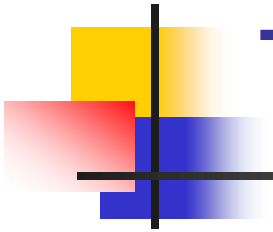
Partial sorting



Partial sorting

- In the previous example we had:
  - Reducers
  - The size of each input (of mappers) is similar
  - The size of each output (of reducers) is very different
  - The output of each mapper is sorted
  - The output of each reducer is sorted
  - Difficult to produce a global sorted output

Total Sorting



Total sorting

- How can we produce a globally sorted file using Hadoop?
  - Use a single partition:
    - This is incredibly inefficient for large files, because one machine must process all the output
  - Produce a set of sorted files that, if concatenated, form a globally sorted file
    - Use a partitioner that respects the total order of the output
    - Choose the partition sizes carefully to ensure that they are fairly even, so job times are not dominated by a single reducer

Total sorting

- Hadoop allows the sampling of the key space:
  - Look at a small subset of the keys to approximate the key distribution, which is then used to construct partitions
- Hadoop comes with a selection of samplers. The `InputSampler` class defines a nested `Sampler` interface whose implementations return a sample of keys given an `InputFormat` and `Job`:

```
public interface Sampler<K,V> {
    K[] getSample(InputFormat<K,V> inf, Job job)
                throws IOException, InterruptedException;
}
```

Total sorting

- Usually, this interface is not called directly. Instead, the `writePartitionFile()` static method (of class `InputSampler`) is used to create a sequence file that stores the keys that define the partitions

```
public static <K, V> void writePartitionFile(Job j, Sampler<K, V> sampler)
    throws IOException, ClassNotFoundException, InterruptedException {
  ...
}
```

- The sequence file is used by `TotalOrderPartitioner` to create partitions for the sort job.

Total sorting

```
public class SortByTemperatureUsingTotalOrderPartitionerApplication
        extends Configured implements Tool {

  public static void main(String[] args) throws Exception {
    int exitCode = ToolRunner.run(
      new SortByTemperatureUsingTotalOrderPartitionerApplication(),
      args);
    System.exit( exitCode );
  }

  @Override
  public int run(String[] args) throws Exception {
    if (args.length != 2) {
      ...
      return -1;
    }
  ...
```

Total sorting

```
    ...
    Configuration conf = getConf();
    Job job = Job.getInstance( conf );
    job.setJobName( "Max temperature - Version 7" );
    job.setJarByClass(
SortByTemperatureUsingTotalOrderPartitionerApplication.class );

    FileInputFormat.addInputPath(job, new Path(args[0]) );
    FileOutputFormat.setOutputPath(job, new Path(args[1]) );

    job.setInputFormatClass( SequenceFileInputFormat.class );
    job.setMapperClass( SortByTemperatureMapper.class );

    job.setMapOutputKeyClass( FloatWritable.class );
    job.setMapOutputValueClass( Text.class );
    ...
```

Total sorting

```
...
job.setReducerClass( SortByTemperatureReducer.class );
job.setOutputKeyClass( FloatWritable.class );
job.setOutputValueClass( Text.class );

job.setOutputFormatClass( SequenceFileOutputFormat.class );
SequenceFileOutputFormat.setCompressOutput( job, true);
SequenceFileOutputFormat.setOutputCompressorClass(
    job, GzipCodec.class );
SequenceFileOutputFormat.setOutputCompressionType(
    job, CompressionType.BLOCK );

job.setPartitionerClass( TotalOrderPartitioner.class );
InputSampler.Sampler<FloatWritable, Text> sampler;
sampler = new
  InputSampler.RandomSampler<FloatWritable, Text>(0.1, 10000, 10);
...
```

Total sorting

```
    InputSampler.writePartitionFile( job, sampler );

    String partitionFile;
    partitionFile = TotalOrderPartitioner.getPartitionFile( conf );
    URI partitionUri = new URI( partitionFile );
    job.addCacheFile( partitionUri );

    System.out.printf(
      "Partition file absolute location: hdfs:///user/%s/%s\n",
      System.getenv("USER"), partitionUri.getPath() );

    return job.waitForCompletion(true) ? 0 : 1;
  }
}
```

Total sorting

```
2019-11-28 19:53:00,660 INFO input.FileInputFormat: Total input files to process : 6
2019-11-28 19:53:00,697 INFO zlib.ZlibFactory: Successfully loaded & initialized native-zlib library
2019-11-28 19:53:00,698 INFO compress.CodecPool: Got brand-new decompressor [.gz]
2019-11-28 19:53:00,702 INFO compress.CodecPool: Got brand-new decompressor [.gz]
2019-11-28 19:53:00,702 INFO compress.CodecPool: Got brand-new decompressor [.gz]
2019-11-28 19:53:00,702 INFO compress.CodecPool: Got brand-new decompressor [.gz]
2019-11-28 19:53:00,745 INFO partition.InputSampler: Using 677 samples
2019-11-28 19:53:01,280 INFO compress.CodecPool: Got brand-new compressor [.deflate]
Partition file absolute location: hdfs:///user/usermr/_partition.lst
2019-11-28 19:53:01,492 INFO client.RMProxy: Connecting to ResourceManager at hadoop.lrcd.e.ipl.pt/10.
2019-11-28 19:53:01,766 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/ha
0079
2019-11-28 19:53:01,811 INFO input.FileInputFormat: Total input files to process : 6
2019-11-28 19:53:01,847 INFO mapreduce.JobSubmitter: number of splits:6
2019-11-28 19:53:01,977 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1574173509068_0079
2019-11-28 19:53:01,979 INFO mapreduce.JobSubmitter: Executing with tokens: []
2019-11-28 19:53:02,183 INFO conf.Configuration: resource-types.xml not found
2019-11-28 19:53:02,183 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
2019-11-28 19:53:02,239 INFO impl.YarnClientImpl: Submitted application application_1574173509068_0079
2019-11-28 19:53:02,271 INFO mapreduce.Job: The url to track the job: http://hadoop.lrcd.e.ipl.pt:8088
2019-11-28 19:53:02,271 INFO mapreduce.Job: Running job: job_1574173509068_0079
2019-11-28 19:53:08,368 INFO mapreduce.Job: Job job_1574173509068_0079 running in uber mode : false
2019-11-28 19:53:08,369 INFO mapreduce.Job:  map 0% reduce 0%
2019-11-28 19:53:15,478 INFO mapreduce.Job:  map 17% reduce 0%
2019-11-28 19:53:19,507 INFO mapreduce.Job:  map 33% reduce 0%
2019-11-28 19:53:22,528 INFO mapreduce.Job:  map 50% reduce 0%
2019-11-28 19:53:24,552 INFO mapreduce.Job:  map 67% reduce 0%
2019-11-28 19:53:25,562 INFO mapreduce.Job:  map 83% reduce 0%
```

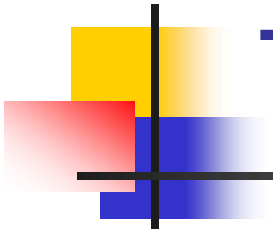

usermr@hadoop: ~/examples/java/Ex01-a7
Partition contents:
-1.1 (null)
3.9 (null)
11.1 (null)
Partitions boundaries
Contents of file:///home/usermr/examples/output/temperatures/small-sorted/part-r-00000 (1515 entries):
-28.9 00292289209999919080111060004+60717+028783FM-12+000399999V0209991C00001999999
-27.2 00290294409999919080110200004+61500+023767FM-12+008499999V0200901N00211999999
-26.7 00290294409999919080101060004+61500+023767FM-12+008499999V0200901N00411999999
...
-1.7 00290298109999919030403130004+59500+020350FM-12+002699999V0200201N01391999999
-1.7 00290298109999919030212130004+59500+020350FM-12+002699999V0203201N02011999999
-1.7 00290298109999919031228060004+59500+020350FM-12+002699999V0201801N00211999999
Contents of file:///home/usermr/examples/output/temperatures/small-sorted/part-r-00001 (1839 entries):
-1.1 00290290709999919010314200004+64333+023450FM-12+000599999V0201801N00621999999
-1.1 00290290709999919011112130004+64333+023450FM-12+000599999V0200201N01591999999
-1.1 00350290709999919010309200004+64333+023450FM-12+0
...
3.3 00290298109999919030508060004+59500+020350FM-12+0
3.3 00290298109999919030507130004+59500+020350FM-12+0
3.3 00290298109999919030507060004+59500+020350FM-12+0
Sorted within each file, and globally sorted
Contents of file:///home/usermr/examples/output/temperatures/small-sorted/part-r-00002 (1428 entries):
3.9 00290298109999919010123130004+59500+020350FM-12+002699999V0202901N01801999999
3.9 00290298109999919010430200004+59500+020350FM-12+002699999V0200701N00721999999
3.9 00290298109999919011111013004+59500+020350FM-12+002699999V0202701N01291999999
...
10.6 00290298109999919011009200004+59500+020350FM-12+002699999V0202301N01181999999
10.6 00290298109999919011201006004+59500+020350FM-12+002699999V0201401N00311999999
10.6 00290298109999919011101106004+59500+020350FM-12+002699999V0201801N00871999999
Contents of file:///home/usermr/examples/output/temperatures/small-sorted/part-r-00003 (1733 entries):
11.1 00290290709999919030906060004+64333+023450FM-12+000599999V0202001N01391999999
11.1 00290290709999919030719200004+64333+023450FM-12+000599999V0203601N01591999999
11.1 00290290709999919030907200004+64333+023450FM-12+000599999V0202501N00211999999
...
28.9 00290294409999919080728130004+61500+023767FM-12+008499999V0202301N00511999999
28.9 00292289209999919080721130004+60717+028783FM-12+000399999V0201601N00931999999
29.4 00290298109999919010720130004+59500+020350FM-12+002699999V0209991C00001999999
Equivalent sizes
Total number of entries: 6515
usermr@hadoop:~/examples/java/Ex01-a7$

Total sorting

```
usermr@hadoop: ~/exam
Partition contents:
-1.1     (null)
3.9      (null)
11.1     (null)
```

< -1.1 ºC	[-1.1ºC, 3.9 ºC [	[3.9ºC, 11.1ºC [	> 11.1ºC
1515/6515 = 23%	1839/6515 = 28%	1428/6515 = 22%	1733/6515 = 27%

- The file that holds the partition boundaries must be shared by all the tasks running in the cluster. This can be achieved using the distributed cache mechanism



Total sorting

- The input dataset determines the best sampler to use:
  - `SplitSampler`, which samples only the first `n` records in a split, is not so good for sorted data
  - `IntervalSampler` chooses keys at regular intervals through the split and makes a better choice for sorted data
  - `RandomSampler` is a good general-purpose

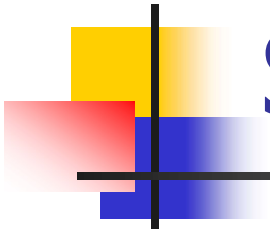
Total sorting

- `InputSampler` and `TotalOrderPartitioner` allow us to choose the number of partitions, i.e., the number of reducers.

- However, `TotalOrderPartitioner` will work only if the partition boundaries are distinct.

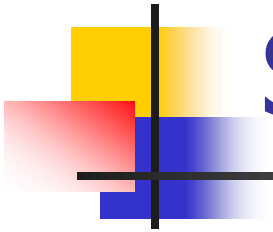
- Choosing a high number of partitions with a small number of keys may lead to collisions

Secondary Sorting



Secondary sorting

- Scenario 1:
  - Sort the temperature readings by year (ascending order) and for each year sort the temperatures by their value (descending order)
  - For each year we choose the first temperature. This is equivalent to the maximum temperature for a particular year



Secondary sorting

- Scenario 2:
  - In a set of videos identify all the peoples that appear in the videos and produce a report that for each people presents:
    - The videos where each people appear
    - For each video the time where the people appear

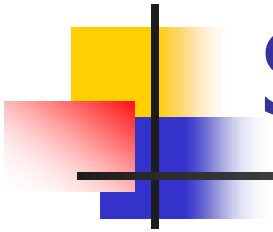
Secondary sorting

- Scenario 3:
  - A set of sensors used to collect scientific data, where we have m sensors that are sampled at regulars time intervals t

$$(t_1, m_1, r_{80521})$$
$$(t_1, m_2, r_{14209})$$
$$(t_1, m_3, r_{76042})$$
$$\ldots$$
$$(t_2, m_1, r_{21823})$$
$$(t_2, m_2, r_{66508})$$
$$(t_2, m_3, r_{98347})$$

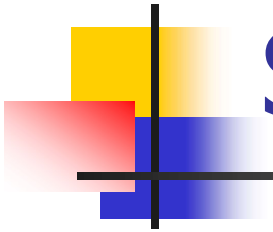
  - For each sensor we want to obtain the values of the sensors ordered by the timestamp

$$m_1 \rightarrow (t_1, r_{80521})$$



Secondary sorting

- MapReduce framework sorts the records by the key before they reach the reducers. For any particular key, however, the values are not sorted.
- In general MapReduce programs are written in such a way that they not to depend on the order in which the values appear to the reduce function.
- However, it is possible to impose an order on the values by sorting and grouping the keys in a particular way



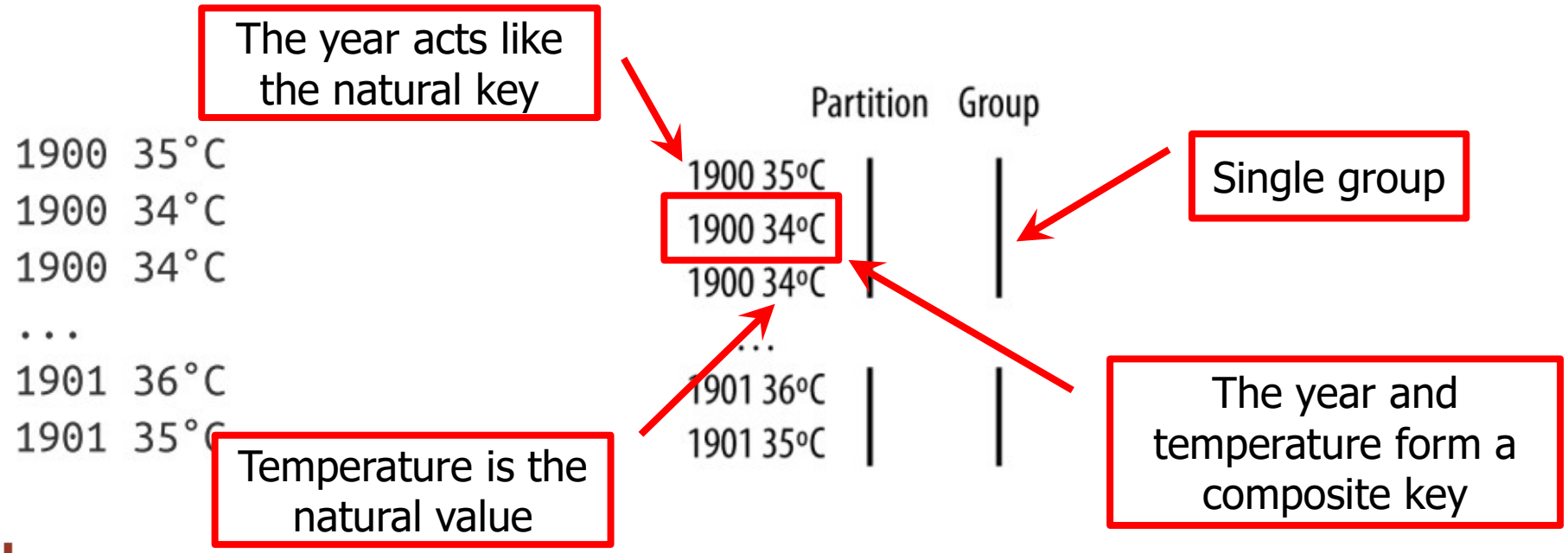
Secondary sorting

- In the previous scenarios, the mechanism used to sort the data is also known (in the context of MapReduce applications) as **Secondary Sorting**

- In order to use secondary sorting we need to create a combined key that depends on the original key but also depends of the values

Secondary sorting

- In the temperature scenario we want to sort the temperatures by the year, and for each year we want to sort the temperature readings



Secondary sorting

- To implement secondary sorting we need to:
  - Make a new **composite key** formed by the natural key and the natural value(s)
  - Implement a sort comparator capable of sorting the **composite key**
  - Implement a partitioner and grouping comparator that can only consider the **key** (not the composite key) for partitioning and sorting purposes

Secondary sorting


```
public class TemperaturePair
        implements WritableComparable<TemperaturePair> {
  private IntWritable year;
  private FloatWritable temperature;

  // Getters and setters for the year and the temperature ...

  public TemperaturePair() {
    this.set(new IntWritable(), new FloatWritable() );
  }

  public TemperaturePair(IntWritable year, FloatWritable temperature) {
    this.set( year, temperature);
  }

  public void set(IntWritable year, FloatWritable temperature) {
    this.year = year;
    this.temperature = temperature;
  }
  ...
```

Secondary sorting


```
...

@Override
public void write(DataOutput out) throws IOException {
  this.year.write( out );
  this.temperature.write( out );
}

@Override
public void readFields(DataInput in) throws IOException {
  this.year.readFields( in );
  this.temperature.readFields( in );
}

@Override
public int hashCode() {
  return this.year.hashCode() * 163 + this.temperature.hashCode();
}
...
```

Secondary sorting


```
  ...
  @Override
  public boolean equals(Object o) {
    if (o instanceof TemperaturePair) {
      TemperaturePair tp = (TemperaturePair)o;
      return year.equals(tp.year)&&temperature.equals(tp.temperature);
    }
    return false;
  }

  @Override
  public String toString() {return this.year +"\t"+ this.temperature;}

  @Override
  public int compareTo(TemperaturePair tp) {
    int cmp = this.year.compareTo( tp.year );
    if (cmp != 0) { return cmp; }
    return this.temperature.compareTo( tp.temperature );
  }
}
```

Secondary sorting


```
public class MaxTemperatureKeyComparator
        extends WritableComparator {

  protected MaxTemperatureKeyComparator() {
    super(TemperaturePair.class, true);
  }

  @Override
  public int compare(WritableComparable w1, WritableComparable w2) {
    TemperaturePair tp1 = (TemperaturePair)w1;
    TemperaturePair tp2 = (TemperaturePair)w2;

    int cmp = tp1.getYear().compareTo( tp2.getYear() );
    if ( cmp!=0 ) { return cmp; }

    // Reverse order
    return -tp1.getTemperature().compareTo( tp2.getTemperature() );
  }
}
```

Secondary sorting


```
public class MaxTemperatureGroupComparator
        extends WritableComparator {

  protected MaxTemperatureGroupComparator() {
    super(TemperaturePair.class, true);
  }

  @Override
  public int compare(WritableComparable w1, WritableComparable w2) {
    TemperaturePair tp1 = (TemperaturePair) w1;
    TemperaturePair tp2 = (TemperaturePair) w2;

    return tp1.getYear().compareTo( tp2.getYear() );
  }
}
```

Secondary sorting


```
public class MaxTemperaturePartitioner
        extends Partitioner<TemperaturePair, FloatWritable> {

  @Override
  public int getPartition(
        TemperaturePair key,
        FloatWritable value,
        int numPartitions) {

    return Math.abs( key.getYear().get() * 127) % numPartitions;
  }
}
```

Secondary sorting


```
public class MaxTemperatureMapper
        extends Mapper<LongWritable,Text,TemperaturePair,FloatWritable> {

  private TemperaturePair theKey = new TemperaturePair();
  private NcdcRecordParser p = new NcdcRecordParser();

  @Override
  public void map(LongWritable key, Text value, Context context)
        throws IOException, InterruptedException {

    this.p.parse( value );

    if ( p.isValidTemperature() && p.getAirTemperature()<100.0F ) {
      this.theKey.setYear( this.parser.getYear() );
      this.theKey.setTemperature( this.parser.getAirTemperature() );

      context.write( this.theKey, this.theKey.getTemperature() );
    }
  }
}
```

Secondary sorting


```
public class MaxTemperatureReducer
     extends Reducer<TemperaturePair,FloatWritable,TemperaturePair,Text>{

  private List<String> valuesAsString = new ArrayList<String>();

  @Override
  protected void reduce(
                TemperaturePair key,
                Iterable<FloatWritable> values,
                Context context)
        throws IOException, InterruptedException {

    this.valuesAsString.clear();

    for (FloatWritable currentTemperature : values) {
      this.valuesAsString.add(0, "" + currentTemperature.get() );
    }

    ...
```

Secondary sorting


```
    ...
    String _out;
    Text out;

    _out = "[" + ... + "]";
    out = new Text(
      String.format(
        "{%d temperatures, %s}", this.valuesAsString.size(),
        _out) );

    context.write( key, out );
  }
}
```

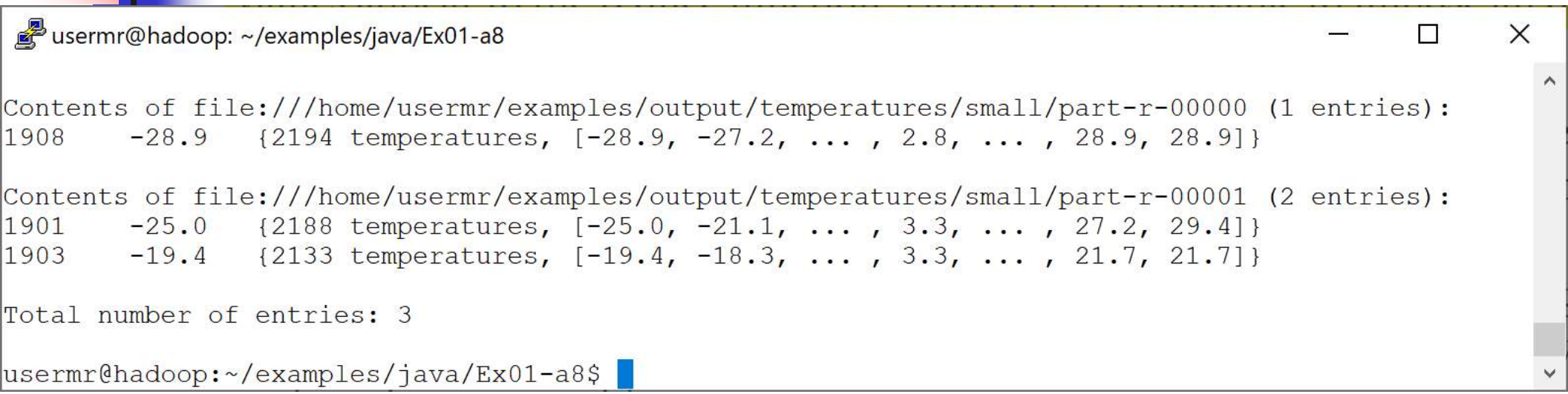
Secondary sorting


```
public class MaxTemperatureApplication
        extends Configured implements Tool {

  public static void main(String[] args) throws Exception {
    int exCode = ToolRunner.run( new MaxTemperatureApplication(), args);
    System.exit( exCode );
  }

  @Override
  public int run(String[] args) throws Exception {
    if (args.length != 2) {
      ...
      return -1;
    }

    Job job = Job.getInstance( getConf() );

    job.setJobName( "Max temperature - Version 8" );
    job.setJarByClass( MaxTemperatureApplication.class );
    ...
```

Secondary sorting


```
    ...
    FileInputFormat.addInputPath(job, new Path(args[0]) );
    FileOutputFormat.setOutputPath(job, new Path(args[1]) );
    job.setMapperClass( MaxTemperatureMapper.class );
    job.setMapOutputKeyClass( TemperaturePair.class );
    job.setMapOutputValueClass( FloatWritable.class );
    job.setPartitionerClass( MaxTemperaturePartitioner.class );
    job.setSortComparatorClass( MaxTemperatureKeyComparator.class );
    job.setGroupingComparatorClass( MaxTemperatureGroupComparator.class );
    job.setReducerClass(MaxTemperatureReducer.class);
    job.setOutputKeyClass( TemperaturePair.class);
    job.setOutputValueClass( Text.class );
    job.setOutputFormatClass( SequenceFileOutputFormat.class );
    SequenceFileOutputFormat.setCompressOutput( job, true);
    SequenceFileOutputFormat.setOutputCompressorClass(
        job, GzipCodec.class );
    SequenceFileOutputFormat.setOutputCompressionType(
        job, CompressionType.BLOCK );
    return job.waitForCompletion(true) ? 0 : 1;
  }
}
```

Secondary sorting



- 2194+2188+2133 = 6515