# Procesamiento de datos masivos

Jesús Morán



- No es streaming puro -> Microbatches
  - ☐ Streaming sobre batch
- No soporta nativamente iteraciones
- Fallos de memoria -> mejoró en las últimas versiones

- Es streaming puro
- Batch sobre streaming
- Soporte nativo para iteraciones
- Gestión de memoria
- Simplifica la configuración y optimización

# **Rendimiento: Flink vs Spark**

**Open Access** 

# Evaluating new approaches of big data analytics frameworks

Norman Spangenberg and Martin Roth and Bogdan Franczyk

Abstract-Big Data analytics

popularity as a tool to process la Spark and Flink are two Apacl

works that facilitate the develop using directly acyclic graph p of these frameworks is challeng strongly rely on complex parat

in-depth understanding of the u Although extensive research has

evaluating the performance of su them benchmark the platforms rather unfair comparison conside design principles. This paper air respect, by directly evaluating

Flink. Our goal is to identify

different architectural choices at

on the perceived end-to-end perfa methodology for correlating

operators execution plan with

methodology to dissect the per

with several representative batcl

to 100 nodes. Our key finding

framework outperforms the oth

job patterns. This paper perform

cases when each framework is

this performance correlates to o

to the specifics of the internal fi

Index Terms—Big Data, perfor

I. INTRO

Spark versus Flink: Understanding Performance in Big Data Analytics Frameworks María S. Pérez-Hernández

in informa Ovidiu-Cristian Marcu Inria Rennes - Bretagne Atlantique the efficie. ovidiu-cristian.marcu@inria.fr

Alexandru Costan IRISA / INSA Rennes alexandru.costan@irisa.fr

Gabriel Antoniu Inria Rennes - Bretagne Atlantique gabriel.antoniu@inria.fr

Ontology Engineering Group Universidad Politecnica de Madrid mperez@fi.upm.es

No hay bala de plata

Los dos son rápidos

gorithms new proce ronments Flink and from diffe Key wor works, big

Abstract

1 Introduct

CrossMark A comparison on scalability for batch big data processing on Apache Spark and Apache

Diego García-Gil<sup>1\*</sup> , Sergio Ramírez-Gallego<sup>1</sup>, Salvador García<sup>1,2</sup> and Francisco Herrera<sup>1,2</sup>

ljgarcia@decsai.ugr.es Department of Computer Science nd Artificial Intelligence, CITIC-UGR Research Center on Information nd Communications Technology), Iniversity of Granada, Calle eriodista Daniel Saucedo Aranda, 8071 Granada, Spain ull list of author information is vailable at the end of the article

The large amounts of data have created a need for new frameworks for processing. The MapReduce model is a framework for processing and generating large-scale datasets with parallel and distributed algorithms. Apache Spark is a fast and general engine for large-scale data processing based on the MapReduce model. The main feature of Spark is the in-memory computation. Recently a novel framework called Apache Flink has emerged, focused on distributed stream and batch data processing. In this paper we perform a comparative study on the scalability of these two frameworks using the corresponding Machine Learning libraries for batch data processing. Additionally we analyze the performance of the two Machine Learning libraries that Spark currently has, MLlib and ML. For the experiments, the same algorithms and the same dataset are being used. Experimental results show that Spark MLlib has better perfomance and overall lower runtimes than Flink.

Keywords: Big data, Spark, Flink, MapReduce, Machine learning

Universidad Internacional de Valencia

#### Flink

The VLDB Journal DOI 10.1007/s00778-014-0357-y

#### REGULAR PAPER

# The Stratosphere platform for big data analytics

Alexander Alexandrov • Rico Bergmann • Stephan Ewen • Johann-Christoph Freytag • Fabian Hueske · Arvid Heise · Odej Kao · Marcus Leich · Ulf Leser · Volker Markl · Felix Naumann · Mathias Peters · Astrid Rheinländer · Matthias J. Sax · Sebastian Schelter · Mareike Höger · Kostas Tzoumas · Daniel Warneke

Received: 10 July 2013 / Revised: 18 March 2014 / Accepted: 1 April 2014 © Springer-Verlag Berlin Heidelberg 2014

Abstract We present Stratosphere, an open-source software stack for parallel data analysis. Stratosphere brings together a unique set of features that allow the expressive, easy, and efficient programming of analytical applications at very large scale. Stratosphere's features include "in situ" data processing, a declarative query language, treatment of user-defined functions as first-class citizens, automatic pro-

Stratosphere is funded by the German Research Foundation (DFG) under grant FOR 1306.

gram parallelization and optimization, support programs, and a scalable and efficient execu Stratosphere covers a variety of "Big Data" use as data warehousing, information extraction and data cleansing, graph analysis, and statistical ar cations. In this paper, we present the overall syst ture design decisions, introduce Stratosphere th ple queries, and then dive into the internal wor system's components that relate to extensibili ming model, optimization, and query execution mentally compare Stratosphere against popular

# Apache Flink $^{\text{TM}}$ : Stream and Batch Processing in a Single Engine

Paris Carbone† Asterios Katsifodimos\* Stephan Ewen‡ Kostas Tzoumas‡ Volker Markl\*

†KTH & SICS Sweden parise,haridi@kth.se

\*TU Berlin & DFKI first.last@tu-berlin.de \*data Artisans first@data-artisans.com

#### Abstract

Apache Flink<sup>®</sup> is an open-source system for processing streaming and batch data. Flink is built on the philosophy that many classes of data processing applications, including real-time analytics, continuous data pipelines, historic data processing (batch), and iterative algorithms (machine learning, graph analysis) can be expressed and executed as pipelined fault-tolerant dataflows. In this paper, we present Flink's architecture and expand on how a (seemingly diverse) set of use cases can be unified under a single execution model.

Data-stream processing (e.g., as exemplified by complex event processing systems) and static (batch) data processing (e.g., as exemplified by MPP databases and Hadoop) were traditionally considered as two very different

#### **Características**

- Tiene un optimizador
  - ☐ Las transformaciones no se ejecutan secuencialmente
- Evaluación perezosa
- Entorno de ejecución
- Registros de un dataset: sin clave, con clave, con clave compuesta
- Sistema de ejecución Streaming: DataStream y DataSet

#### **Fuentes de datos**

- Archivos:
  - readFile(inputFormat, path)
  - □ readTextFile(path): Utiliza TextInputFormat (cada registro es una línea del archivo)
  - □ readCsvFile(path): lee un CSV y nos lo guarda como una tupla
- Desde colección:
  - ☐ fromCollection(Collection)
  - □ fromElements(T ...)
  - □ generateSequence(from, to): desde un número a otro
- Otros:
  - createInput(inputFormat): cualquier fuente de datos con inputFormat Ejemplo: mongoInputFormat

- Conectores streaming:
  - □ Kafka
  - RabbitMQ
  - ☐ Google Pub/Sub
  - □ Ni-Fi
  - □ Kinesis
  - Cassandra
  - □ Redis
  - Elasticsearch
  - ☐ Sistema de archivos de Hadoop
  - □ Twitter

#### **Sinks**

## Arhivo:

- writeAsText(): utiliza TextOutputFormat (cada registro se guarda en una línea)
- writeAsCsv(...): utiliza CsvOutputFormat (guarda como csv indicando delimitador, etc.)
- write(): utiliza FileOutputFormat y nos sirve para crear nuestros otros métodos customizados de almacenamiento en archivo

#### Otro:

- print(),printToErr(), print(String msg) y printToErr(String msg): muestra los datos por consola
- output(): exporta a cualquier fuente de datos con OutputFormat, por ejemplo MongoOutputFormat.

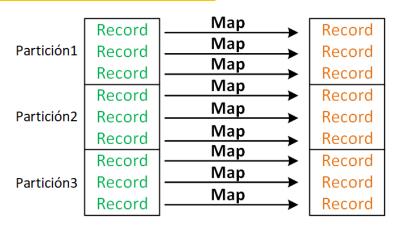


# map

☐ Por cada registro siempre emite un registro

```
Data before map:
Partition 0: uno, dos, tres
Partition 1: cuatro, cinco, seis
Partition 2: siete, ocho, nueve
```

```
Data after map:
Partition 0: 3, 3, 4
Partition 1: 6, 5, 4
Partition 2: 5, 4, 5
```



# map

☐ Por cada registro siempre emite un registro

map(ocho) -> 4

map(nueve) -> 5

```
DataSource<String> data = env.fromCollection(Arrays.asList("uno", "dos", "tres",
                                                                         "cuatro", "cinco", "seis",
                                                                         "siete", "ocho", "nueve"));
         DataSet<Integer> lettersOfRegisters = data.map(new sumLetters());
          public static final class sumLetters implements MapFunction<String, Integer> {
               private static final long serialVersionUID = -6900227250953473523L;
              @Override
                                                                                                              Map
                                                                                                 Record
                                                                                                                          Record
              public Integer map(String value) throws Exception {
                                                                                                              Map
                                                                                        Partición1
                                                                                                 Record
                                                                                                                          Record
                   return value.length();
                                                                                                             Map
                                                                                                 Record
                                                                                                                          Record
                                                                                                              Map
                                                                                                 Record
                                                                                                                          Record
                                                                                                              Map
                                                                                        Partición2
                                                                                                 Record
                                                                                                                          Record
                                                                                                              Map
                                                                                                 Record
                                                                                                                          Record
                                   Task 0: map(uno) -> 3
                                                                                                              Map
                                                              Data after map:
Data before map:
                                                                                                                          Record
                                                                                                 Record
                                          map(dos) -> 3
                                                              Partition \theta: 3, 3, 4
                                                                                                             Map
Partition 0: uno, dos, tres
                                          map(tres) -> 4
                                                                                        Partición3
                                                                                                 Record
                                                                                                                          Record
Partition 1: cuatro, cinco, seis
                                                              Partition 1: 6, 5, 4
                                   Task 1: map(cuatro) -> 6
                                                                                                             Map
Partition 2: siete, ocho, nueve
                                                                                                 Record
                                                                                                                          Record
                                                               Partition 2: 5, 4,
                                          map(cinco) -> 5
                                          map(seis) -> 4
                                   Task 2: map(siete) -> 5
```

**Viu** Universidad

Internacional

de Valencia

# map

☐ Por cada registro siempre emite un registro

```
DataSource<String> data = env.fromCollection(Arrays.asList("uno", "dos", "tres",
                                                                  "cuatro", "cinco", "seis",
                                                                  "siete", "ocho", "nueve"));
     DataSet<Integer> lettersOfRegisters = data.map(new MapFunction<String, Integer>() {
         private static final long serialVersionUID = 1L;
         @Override
         public Integer map(String value) throws Exception {
             return value.length();
                                                                                                                 Map
                                                                                                  Record
                                                                                                                               Record
                                                                                                                 Map
                                                                                        Partición1
                                                                                                  Record
                                                                                                                               Record
     });
                                                                                                                 Map
                                                                                                  Record
                                                                                                                               Record
                                                                                                                 Map
                                                                                                  Record
                                                                                                                               Record
                                                                                                                 Map
                                   Task 0: map(uno) -> 3
Data before map:
                                                                                        Partición2
                                                                                                  Record
                                                                                                                               Record
                                                               Data after map:
                                           map(dos) -> 3
Partition 0: uno, dos, tres
                                                                                                                 Map
                                                               Partition \theta: 3, 3, 4
                                                                                                  Record
                                                                                                                               Record
                                           map(tres) -> 4
Partition 1: cuatro, cinco, seis
                                                               Partition 1: 6, 5, 4
                                                                                                                 Map
                                   Task 1: map(cuatro) -> 6
Partition 2: siete, ocho, nueve
                                                                                                  Record
                                                                                                                               Record
                                                               Partition 2:
                                           map(cinco) -> 5
                                                                                                                 Map
                                                                                                                               Record
                                           map(seis) -> 4
                                                                                        Partición3
                                                                                                  Record
                                                                                                                 Map
                                   Task 2: map(siete) -> 5
                                                                                                  Record
                                                                                                                               Record
                                           map(ocho) -> 4
                                           map(nueve) -> 5
```

# flatMap

□ Por cada registro puede emitir 0, 1 o varios registros

```
Record
                                                                                                                                    Record
DataSource<String> data = env.fromCollection(Arrays.asList(
                                                                                              Partición1
                                                                                                        Record
                                                                                                                                    Record
                                 "son tres palabras", "cuatro", "-1",
                                 "uno", "-1", "otra",
                                                                                                        Record
                                                                                                                                    Record
                                 "-1", "-1", "tercer particion con cinco palabras"));
                                                                                                        Record
                                                                                                                                    Record
DataSet<String> newData = data.flatMap(new FlatMapFunction<String, String>() {
                                                                                              Partición2
                                                                                                        Record
                                                                                                                                    Record
    private static final long serialVersionUID = -6900227250953473523L;
                                                                                                        Record
                                                                                                                                    Record
    @Override
                                                                                                        Record
                                                                                                                                    Record
    public void flatMap(String value, Collector<String> out) throws Exception {
        if (! value.equals("-1"))
                                                                                              Partición3
                                                                                                        Record
                                                                                                                                    Record
            for (String word : value.split(" "))
                                                                                                        Record
                                                                                                                                    Record
                out.collect(word);
                                                                                                                                    Record
        .........
       Data before flatMap:
                                                                                                                                    Record
       Partition 0: son tres palabras, cuatro, -1
       Partition 1: uno, -1, otra
       Partition 2: -1, -1, tercer particion con cinco palabras
                                                                                        Data after flatMap:
                                                                                         Partition 0: son, tres, palabras, cuatro
                                                                                        Partition 1: uno, otra
                           Task 0: flatMap(son tres palabras) -> son, tres, palabras
                                                                                         Partition 2: tercer, particion, con, cinco, palabras
                                   flatMap(cuatro) -> cuatro
                                   flatMap(-1) ->
                           Task 1: flatMap(uno) -> uno
                                   flatMap(-1) ->
                                   flatMap(otra) -> otra
                                                                                                                                  Viu Universidad
                           Task 2: flatMap(-1) ->
                                                                                                                                     Internacional
                                   flatMap(-1) ->
                                                                                                                                     de Valencia
                                   flatMap(tercer particion con cinco palabras) -> tercer, particion, con, cinco, palabras
```

#### **Tareas**

filter

Por cada registro o lo emite o no

```
Data before filter:
Partition 0: 1, -1, 3
Partition 1: 4, 5, -1
Partition 2: -1, -1, 9
```

```
Task 0: filter(1) -> true
filter(-1) -> false
filter(3) -> true
Task 1: filter(4) -> true
filter(5) -> true
filter(-1) -> false
Task 2: filter(-1) -> false
filter(-1) -> false
filter(9) -> true
```

```
Record
          Record
Partición1
          Record
          Record
                                       Record
                                       Record
          Record
Partición2
          Record
                                       Record
          Record
          Record
Partición3
          Record
          Record
                                       Record
```

```
Data after filter:
Partition 0: 1, 3
Partition 1: 4, 5
Partition 2: 9
```

# mapPartition

☐ Por cada partición puede emitir 0, 1 o n registros

```
DataSource<String> data = env.fromCollection(Arrays.asList(
                       "las cuatro palabras emitidas", "-1", "esto no se emite",
                       "se", "emite", "-1",
                       "lo último que se emite", "-1", "esto tampoco se emite"));
DataSet<String> newData = data.mapPartition(
                                   new MapPartitionFunction<String, String> () {
   private static final long serialVersionUID = -1760362441299143862L;
    @Override
   public void mapPartition(Iterable<String> values, Collector<String> out)
                                                          throws Exception {
       for (String value : values) {
           if (value.equals("-1"))
               break:
           else
               out.collect(value):
      Data before mapPartition:
                     las cuatro palabras emitidas, -1, esto no se emite
      Partition 1: se, emite, -1
                     lo último que se emite, -1, esto tampoco se emite
```

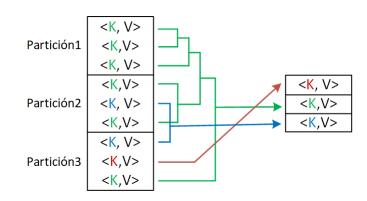
```
Record
                                        Record
Partición1
          Record
                                        Record
          Record
                                        Record
          Record
                                        Record
Partición2
          Record
                                        Record
          Record
                                        Record
          Record
                                        Record
          Record
Partición3
                                        Record
          Record
                                        Record
                                        Record
                                        Record
```

Universidad

Internacional de Valencia

```
Data after filter:
Partition 0: las cuatro palabras emitidas
Partition 1: se, emite
Partition 2: lo último que se emite
```

```
DataSet<Tuple2<Integer, Integer>> data = env.fromCollection(Arrays.asList(
                                new Tuple2<Integer, Integer>(1999, 7),
                                new Tuple2<Integer, Integer>(1999, 5),
                                new Tuple2<Integer, Integer>(1999, 10),
                                new Tuple2<Integer, Integer>(1999, 4),
                                new Tuple2<Integer, Integer>(2000, 3),
                                new Tuple2<Integer, Integer>(2000, 7),
                                new Tuple2<Integer, Integer>(2000, 10),
                                new Tuple2<Integer, Integer>(2001, 3),
                                new Tuple2<Integer, Integer>(1999, 5)));
DataSet<Tuple2<Integer, Integer>> newData = data.groupBy(0).reduce(
                            new ReduceFunction<Tuple2<Integer, Integer>>() {
    private static final long serialVersionUID = -7573583914328613340L;
    @Override
    public Tuple2<Integer, Integer> reduce(Tuple2<Integer, Integer> accum,
                            Tuple2<Integer, Integer> value) throws Exception {
        if (value.f1 > accum.f1) //value.temperature > accum.temperature
            return value:
        else
            return accum;
});
```



```
DataSet<Tuple2<Integer, Integer>> data = env.fromCollection(Arrays.asList(
                                new Tuple2<Integer, Integer>(1999, 7),
                                new Tuple2<Integer, Integer>(1999, 5),
                                new Tuple2<Integer, Integer>(1999, 10),
                                new Tuple2<Integer, Integer>(1999, 4),
                                new Tuple2<Integer, Integer>(2000, 3),
                                new Tuple2<Integer, Integer>(2000, 7),
                                new Tuple2<Integer, Integer>(2000, 10),
                                new Tuple2<Integer, Integer>(2001, 3),
                                new Tuple2<Integer, Integer>(1999, 5)));
DataSet<Tuple2<Integer, Integer>> newData = data.groupBy(0).reduce(
                            new ReduceFunction<Tuple2<Integer, Integer>>() {
    private static final long serialVersionUID = -7573583914328613340L;
    @Override
    public Tuple2<Integer, Integer> reduce(Tuple2<Integer, Integer> accum,
                            Tuple2<Integer, Integer> value) throws Exception {
        if (value.f1 > accum.f1) //value.temperature > accum.temperature
            return value:
        else
            return accum;
});
```



```
DataSet<Tuple2<Integer, Integer>> data = env.fromCollection(Arrays.asList(
                                new Tuple2<Integer, Integer>(1999, 7),
                                new Tuple2<Integer, Integer>(1999, 5),
                                new Tuple2<Integer, Integer>(1999, 10),
                                new Tuple2<Integer, Integer>(1999, 4),
                                new Tuple2<Integer, Integer>(2000, 3),
                                new Tuple2<Integer, Integer>(2000, 7),
                                new Tuple2<Integer, Integer>(2000, 10),
                                new Tuple2<Integer, Integer>(2001, 3),
                                new Tuple2<Integer, Integer>(1999, 5)));
DataSet<Tuple2<Integer, Integer>> newData = data.groupBy(0).reduce(
                            new ReduceFunction<Tuple2<Integer, Integer>>() {
    private static final long serialVersionUID = -7573583914328613340L;
    @Override
    public Tuple2<Integer, Integer> reduce(Tuple2<Integer, Integer> accum,
                            Tuple2<Integer, Integer> value) throws Exception {
        if (value.f1 > accum.f1) //value.temperature > accum.temperature
            return value:
        else
            return accum;
});
```

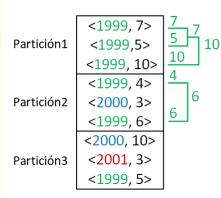
```
Partición1 \begin{array}{|c|c|c|c|c|}\hline <1999,7> \\ <1999,5> \\ <1999,10> \\\hline <1999,4> \\ <2000,3> \\ <1999,6> \\\hline <2000,10> \\ Partición3 & <2001,3> \\ <1999,5> \\\hline \end{array}
```



```
DataSet<Tuple2<Integer, Integer>> data = env.fromCollection(Arrays.asList(
                                new Tuple2<Integer, Integer>(1999, 7),
                                new Tuple2<Integer, Integer>(1999, 5),
                                new Tuple2<Integer, Integer>(1999, 10),
                                new Tuple2<Integer, Integer>(1999, 4),
                                new Tuple2<Integer, Integer>(2000, 3),
                                new Tuple2<Integer, Integer>(2000, 7),
                                new Tuple2<Integer, Integer>(2000, 10),
                                new Tuple2<Integer, Integer>(2001, 3),
                                new Tuple2<Integer, Integer>(1999, 5)));
DataSet<Tuple2<Integer, Integer>> newData = data.groupBy(0).reduce(
                            new ReduceFunction<Tuple2<Integer, Integer>>() {
    private static final long serialVersionUID = -7573583914328613340L;
    @Override
    public Tuple2<Integer, Integer> reduce(Tuple2<Integer, Integer> accum,
                            Tuple2<Integer, Integer> value) throws Exception {
        if (value.f1 > accum.f1) //value.temperature > accum.temperature
            return value:
        else
            return accum;
});
```

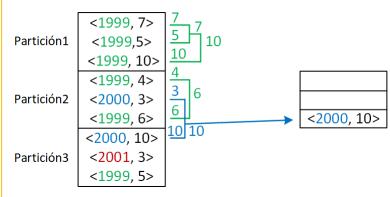


```
DataSet<Tuple2<Integer, Integer>> data = env.fromCollection(Arrays.asList(
                                new Tuple2<Integer, Integer>(1999, 7),
                                new Tuple2<Integer, Integer>(1999, 5),
                                new Tuple2<Integer, Integer>(1999, 10),
                                new Tuple2<Integer, Integer>(1999, 4),
                                new Tuple2<Integer, Integer>(2000, 3),
                                new Tuple2<Integer, Integer>(2000, 7),
                                new Tuple2<Integer, Integer>(2000, 10),
                                new Tuple2<Integer, Integer>(2001, 3),
                                new Tuple2<Integer, Integer>(1999, 5)));
DataSet<Tuple2<Integer, Integer>> newData = data.groupBy(0).reduce(
                            new ReduceFunction<Tuple2<Integer, Integer>>() {
    private static final long serialVersionUID = -7573583914328613340L;
    @Override
    public Tuple2<Integer, Integer> reduce(Tuple2<Integer, Integer> accum,
                            Tuple2<Integer, Integer> value) throws Exception {
        if (value.f1 > accum.f1) //value.temperature > accum.temperature
            return value:
        else
            return accum;
});
```

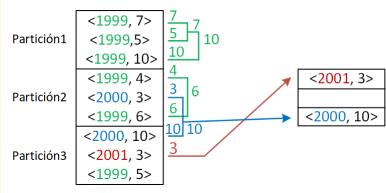




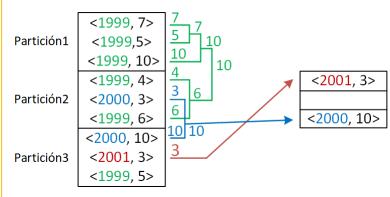
```
DataSet<Tuple2<Integer, Integer>> data = env.fromCollection(Arrays.asList(
                                new Tuple2<Integer, Integer>(1999, 7),
                                new Tuple2<Integer, Integer>(1999, 5),
                                new Tuple2<Integer, Integer>(1999, 10),
                                new Tuple2<Integer, Integer>(1999, 4),
                                new Tuple2<Integer, Integer>(2000, 3),
                                new Tuple2<Integer, Integer>(2000, 7),
                                new Tuple2<Integer, Integer>(2000, 10),
                                new Tuple2<Integer, Integer>(2001, 3),
                                new Tuple2<Integer, Integer>(1999, 5)));
DataSet<Tuple2<Integer, Integer>> newData = data.groupBy(0).reduce(
                            new ReduceFunction<Tuple2<Integer, Integer>>() {
    private static final long serialVersionUID = -7573583914328613340L;
    @Override
    public Tuple2<Integer, Integer> reduce(Tuple2<Integer, Integer> accum,
                            Tuple2<Integer, Integer> value) throws Exception {
        if (value.f1 > accum.f1) //value.temperature > accum.temperature
            return value:
        else
            return accum;
});
```



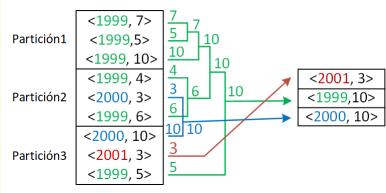
```
DataSet<Tuple2<Integer, Integer>> data = env.fromCollection(Arrays.asList(
                                new Tuple2<Integer, Integer>(1999, 7),
                                new Tuple2<Integer, Integer>(1999, 5),
                                new Tuple2<Integer, Integer>(1999, 10),
                                new Tuple2<Integer, Integer>(1999, 4),
                                new Tuple2<Integer, Integer>(2000, 3),
                                new Tuple2<Integer, Integer>(2000, 7),
                                new Tuple2<Integer, Integer>(2000, 10),
                                new Tuple2<Integer, Integer>(2001, 3),
                                new Tuple2<Integer, Integer>(1999, 5)));
DataSet<Tuple2<Integer, Integer>> newData = data.groupBy(0).reduce(
                            new ReduceFunction<Tuple2<Integer. Integer>>() {
    private static final long serialVersionUID = -7573583914328613340L;
    @Override
    public Tuple2<Integer, Integer> reduce(Tuple2<Integer, Integer> accum,
                            Tuple2<Integer, Integer> value) throws Exception {
        if (value.f1 > accum.f1) //value.temperature > accum.temperature
            return value:
        else
            return accum;
});
```



```
DataSet<Tuple2<Integer, Integer>> data = env.fromCollection(Arrays.asList(
                                new Tuple2<Integer, Integer>(1999, 7),
                                new Tuple2<Integer, Integer>(1999, 5),
                                new Tuple2<Integer, Integer>(1999, 10),
                                new Tuple2<Integer, Integer>(1999, 4),
                                new Tuple2<Integer, Integer>(2000, 3),
                                new Tuple2<Integer, Integer>(2000, 7),
                                new Tuple2<Integer, Integer>(2000, 10),
                                new Tuple2<Integer, Integer>(2001, 3),
                                new Tuple2<Integer, Integer>(1999, 5)));
DataSet<Tuple2<Integer, Integer>> newData = data.groupBy(0).reduce(
                            new ReduceFunction<Tuple2<Integer. Integer>>() {
    private static final long serialVersionUID = -7573583914328613340L;
    @Override
    public Tuple2<Integer, Integer> reduce(Tuple2<Integer, Integer> accum,
                            Tuple2<Integer, Integer> value) throws Exception {
        if (value.f1 > accum.f1) //value.temperature > accum.temperature
            return value:
        else
            return accum;
});
```

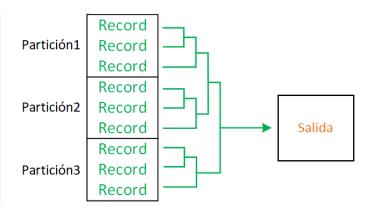


```
DataSet<Tuple2<Integer, Integer>> data = env.fromCollection(Arrays.asList(
                                new Tuple2<Integer, Integer>(1999, 7),
                                new Tuple2<Integer, Integer>(1999, 5),
                                new Tuple2<Integer, Integer>(1999, 10),
                                new Tuple2<Integer, Integer>(1999, 4),
                                new Tuple2<Integer, Integer>(2000, 3),
                                new Tuple2<Integer, Integer>(2000, 7),
                                new Tuple2<Integer, Integer>(2000, 10),
                                new Tuple2<Integer, Integer>(2001, 3),
                                new Tuple2<Integer, Integer>(1999, 5)));
DataSet<Tuple2<Integer, Integer>> newData = data.groupBy(0).reduce(
                            new ReduceFunction<Tuple2<Integer. Integer>>() {
    private static final long serialVersionUID = -7573583914328613340L;
    @Override
    public Tuple2<Integer, Integer> reduce(Tuple2<Integer, Integer> accum,
                            Tuple2<Integer, Integer> value) throws Exception {
        if (value.f1 > accum.f1) //value.temperature > accum.temperature
            return value:
        else
            return accum;
});
```



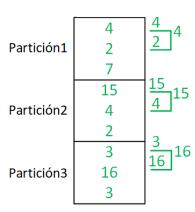
#### **Tareas**

- reduce
  - Emite 1 registro (por DataSet o por clave)

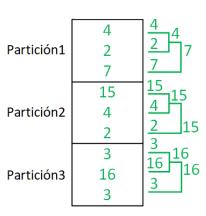


```
Partición1 2 7 15 Partición2 4 2 3 Partición3 16 3
```

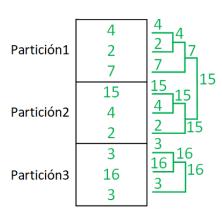
- reduce
  - Emite 1 registro (por DataSet o por clave)



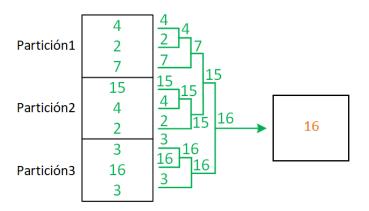
- reduce
  - Emite 1 registro (por DataSet o por clave)



- reduce
  - Emite 1 registro (por DataSet o por clave)



- reduce
  - Emite 1 registro (por DataSet o por clave)



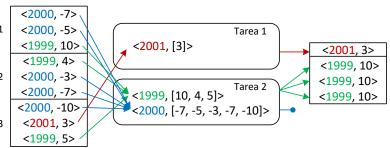
#### **Tareas**

- reduceGroup
  - ☐ Emite 0, 1 o N registros (por DataSet o **por clave**)
  - ☐ Puede ejecutarse con o sin Combine

```
DataSet<Tuple2<Integer, Integer>> newData = data.groupBy(0).reduceGroup(
        new GroupReduceFunction<Tuple2<Integer, Integer>,
                                             Tuple2<Integer, Integer>>() {
   private static final long serialVersionUID = -466800343189074682L;
    @Override
    public void reduce(Iterable<Tuple2<Integer, Integer>> values,
                    Collector<Tuple2<Integer, Integer>> out) throws Exception {
        int maxTemp = Integer.MIN VALUE;
        int year = -1;
        int count = 0:
                                                                             Partición1
        for (Tuple2<Integer, Integer> value : values) {
            year = value.f0;
            if (value.fl > maxTemp) //temperature > maxTemp
                maxTemp = value.fl;
                                                                             Partición2
            count++;
        if (maxTemp > 0) {
                                                                             Partición3
            for (int i = 0; i < count; i++)
                out.collect(new Tuple2<Integer, Integer>(vear, maxTemp));
```

Si la temperatura máxima de un año < 0 -> no emite

Si la temperatura máxima de un año > 0 -> emite <año, temperatura máxima> tantas veces como pares <clave, valor> tenga

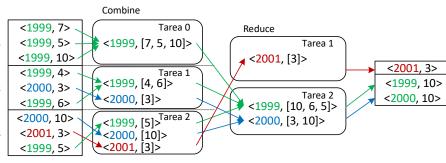


#### **Tareas**

- reduceGroup
  - ☐ Emite 0, 1 o N registros (por DataSet o **por clave**)
  - Puede ejecutarse con o sin Combine

```
public class miReduceCombine implements
    GroupReduceFunction<Tuple2<Integer, Integer>, Tuple2<Integer, Integer>>,
    GroupCombineFunction<Tuple2<Integer, Integer>, Tuple2<Integer, Integer>>{
    private static final long serialVersionUID = 7760143877649257585L;
    @Override
    public void reduce(Iterable<Tuple2<Integer, Integer>> values.
                        Collector<Tuple2<Integer, Integer>> out) {
        Tuple2<Integer, Integer> yearMaxTemp =
            new Tuple2<Integer, Integer>(Integer.MIN VALUE, Integer.MIN VALUE);
        for (Tuple2<Integer, Integer> value : values)
            if (value.fl > yearMaxTemp.fl) //temperature > max temperature Partición1
                vearMaxTemp = value:
        out.collect(yearMaxTemp);
                                                                            Partición2
    @Override
    public void combine(Iterable<Tuple2<Integer, Integer>> values,
                        Collector<Tuple2<Integer, Integer>> out)
            throws Exception {
                                                                            Partición3
        Tuple2<Integer, Integer> yearMaxTemp =
            new Tuple2<Integer, Integer>(Integer.MIN VALUE, Integer.MIN VALUE);
        for (Tuple2<Integer, Integer> value : values)
            if (value.fl > yearMaxTemp.fl) //temperature > max temperature
                vearMaxTemp = value:
        out.collect(yearMaxTemp);
```

#### Combine pude no ejecutarse



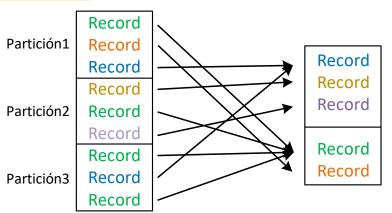
Viu Universidad Internacional de Valencia

#### Distinct

### ☐ Elimina los duplicados

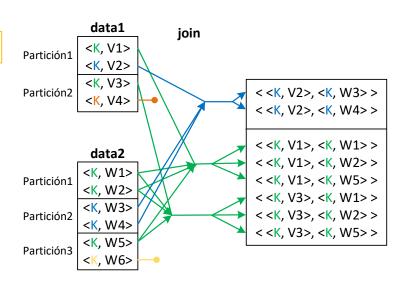
```
Data before distinct:
Partition 0: 1, 2, 3
Partition 1: 4, 1, 6
Partition 2: 1, 3, 1

Data after distinct:
Partition 0: 3, 4, 6
Partition 1: 1, 2
```



#### **Tareas**

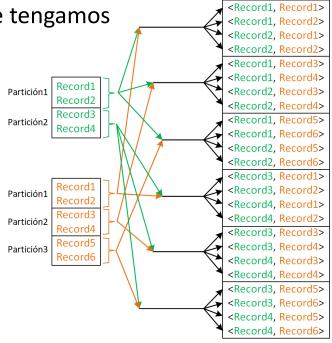
- join
  - ☐ Ejecuta un inner join
  - ☐ Oros joins soportados por Flink: leftOuterJoin y rightOuterJoin



# Cross (producto cartesiano

□ Crea todos los pares de todos los datos que tengamos

```
Data 1 before operation: [tarjeta1, tarjeta2, tarjeta3, tarjeta4]
Data 2 before operation: [Ordenador, altavoces, coche, ruedas,
teclado, otro]
Data after operation: [(tarjeta1,Ordenador), (tarjeta2,Ordenador),
(tarjeta3,Ordenador), (tarjeta4,Ordenador), (tarjeta1,altavoces),
(tarjeta2,altavoces), (tarjeta3,altavoces), (tarjeta4,altavoces),
(tarjeta1,coche), (tarjeta2,coche), (tarjeta3,coche), (tarjeta4,coche),
(tarjeta1,ruedas), (tarjeta2,ruedas), (tarjeta3,ruedas), (tarjeta4,ruedas),
(tarjeta1,teclado), (tarjeta2,teclado), (tarjeta3,teclado),
(tarjeta4,teclado), (tarjeta1,otro), (tarjeta2,otro), (tarjeta3,otro),
(tarjeta4,otro)]
```



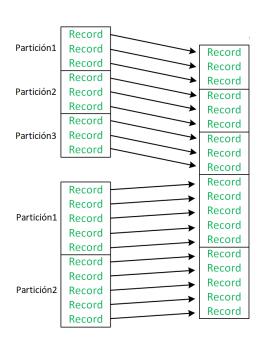
## union

#### □ Une dos DataSets

```
Data 1 before operation: [1, 2, 3, 4, 5, 6, 7, 8, 9]

Data 2 before operation: [10, 11, 12, 13, 14, 15, 16, 17, 18, 19]

Data after operation: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
```



#### **Tareas**

Otras:

- ☐ Rebalance: re-particionar los datos
- Particionado personalizado
- ☐ First: obtener n elementos
- Collect: obtener una lista con el DataSet
- ☐ RichFunction: funciones que proporcionan información de contexto
- □ ...

DataSet: cada vez en más desuso

Table: para datos estructurados

DataStream: para datos (des)estructurados

PyFlink: en implementación

# Gracias

Viu Universidad Internacional de Valencia