Sistemas Distribuidos I
(75.74)

Distribución y Coordinación de Procesos

Coordinación de Actividades, MPI, Apache Beam

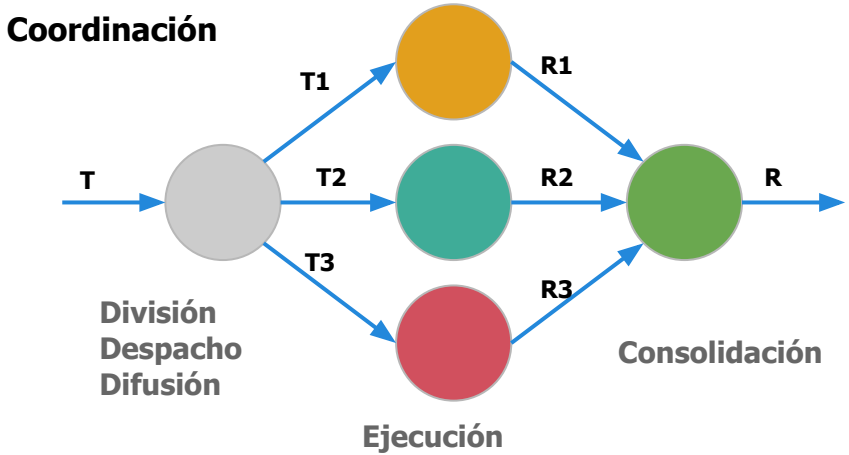
Docentes

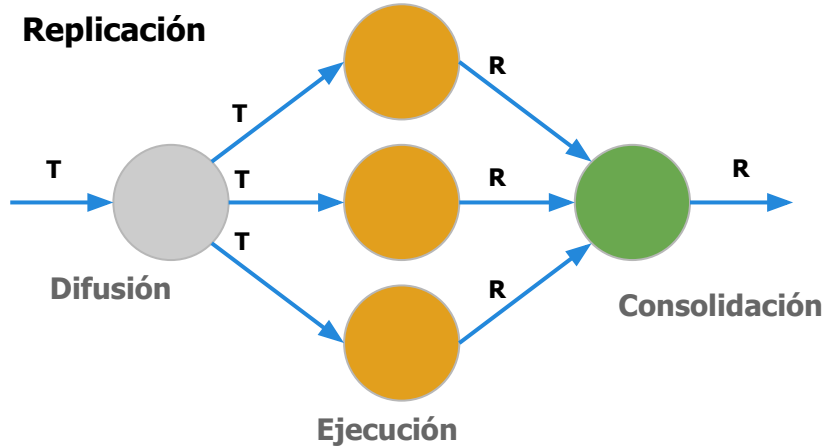
- Pablo D. Roca
- Ezequiel Torres Feyuk
- Guido Albarello

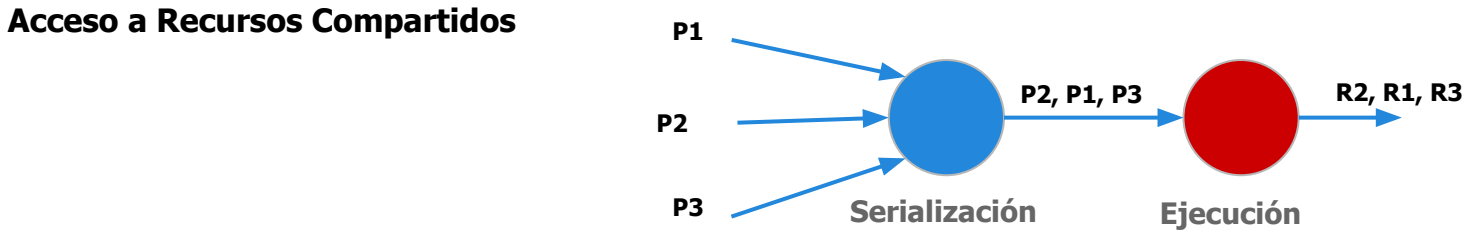
Agenda

- **Coordinación de Actividades**
- Open MPI
- Apache Flink y Apache Beam

Coordinación de Actividades





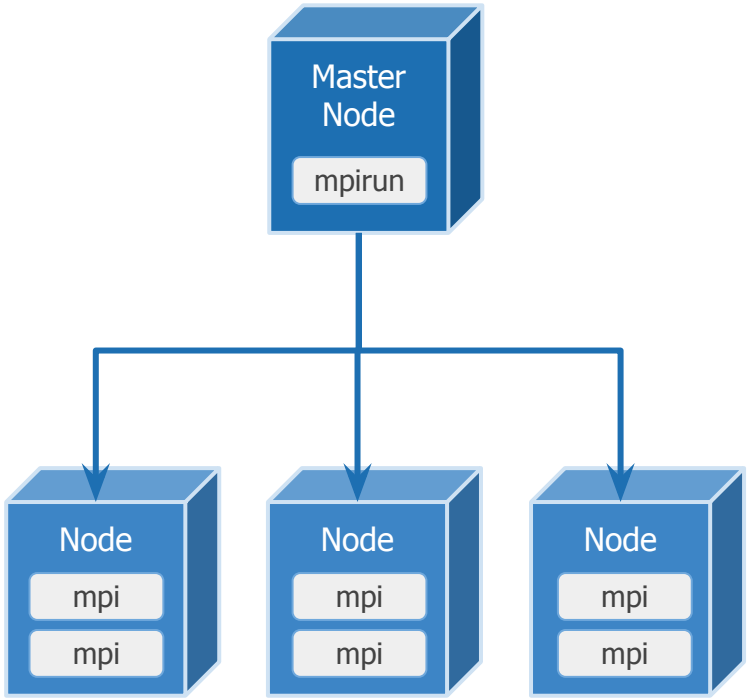


Agenda

- Coordinación de Actividades
- **Open MPI**
- Apache Flink y Apache Beam

Open MPI | Introducción

- Basado en transmisión y recepción de mensajes.
- Ejecución transparente de 1 a N nodos.
- Se utiliza como una librería con abstracciones de uso general con foco en el cómputo distribuido
- Implementa un middleware de comunicación de grupos:
  - MPI_Recv, MPI_Send, MPI_Bcast, MPI_gather, etc.

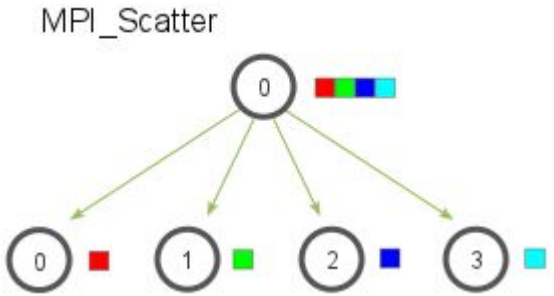


Open MPI | Send, Recv, Broadcast

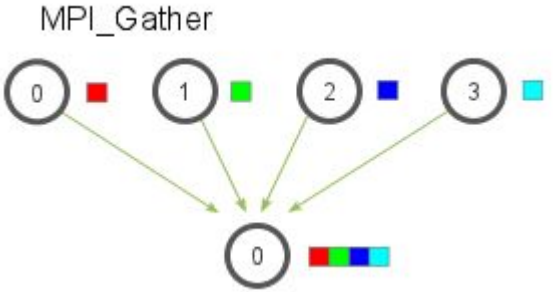
```
MPI_Send(void* data,
    int count,
    MPI_Datatype type,
    int dest,
    int tag,
    MPI_Comm group)

MPI_Recv(void* data,
    int count,
    MPI_Datatype type,
    int source,
    int tag,
    MPI_Comm group,
    MPI_Status* status)
MPI_Bcast(void* data,
    int count,
    MPI_Datatype type,
    int root,
    MPI_Comm group)
```

```
$ mpirun -np 4 ./main
...
MPI_Init(NULL, NULL);
int world_size, world_rank;
MPI_Comm_size(MPI_COMM_WORLD, &world_size);
MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
if (world_rank == 0) {
    int data[4] = {1, 2, 3, 4};
    for (size_t i = 1; i < world_size; ++i)
        MPI_Send(data, 4, MPI_INT, i, 0,
            MPI_COMM_WORLD);
} else {
    int data[4];
    MPI_Recv(data, 4, MPI_INT, 0, 0,
MPI_COMM_WORLD, MPI_STATUS_IGNORE);
}
//again...
int data[4] = {1, 2, 3, 4};
MPI_Bcast(data, 4, MPI_INT, 0, MPI_COMM_WORLD);
MPI_Finalize();
```

```
MPI_Scatter(
    void* send_data,
    int send_count,
    MPI_Datatype send_datatype,
    void* recv_data,
    int recv_count,
    MPI_Datatype recv_datatype,
    int root,
    MPI_Comm group)
```

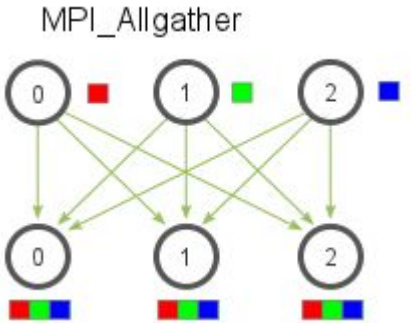


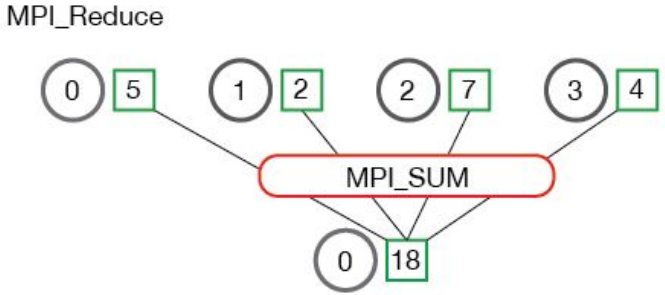

```
MPI_Gather(
    void* send_data,
    int send_count,
    MPI_Datatype send_datatype,
    void* recv_data,
    int recv_count,
    MPI_Datatype recv_datatype,
    int root,
    MPI_Comm group)
```



Open MPI | AllGather, Reduce

```
MPI_Allgather(
    void* send_data,
    int send_count,
    MPI_Datatype send_datatype,
    void* recv_data,
    int recv_count,
    MPI_Datatype recv_datatype,
    MPI_Comm group)

MPI_Reduce(
    void* send_data,
    void* recv_data,
    int count,
    MPI_Datatype datatype,
    MPI_Op op,
    int root,
    MPI_Comm group)

MPI_Op: { MPI_MAX, MPI_MIN, MPI_SUM, MPI_PROD,
          MPI_LAND, MPI_LOR, MPI_BAND, MPI_BOR,
          MPI_MAXLOC, MPI_MINLOC }
```





Agenda

- Coordinación de Actividades
- Open MPI
- **Apache Flink y Apache Beam**

- Plataforma para procesamiento distribuido de datos.
- Incluye motor de ejecución de *pipelines* de transformación.
- Define *framework* Java/Scala para crear pipelines:
  - SQL y Table API permiten definir tablas dinámicas (lógicas) con los flujos de datos y utilizar álgebra relacional
  - Dataset y DataStream API permiten definir secuencias de procesamiento con formato DAG

Reference: https://ci.apache.org/projects/flink/flink-docs-stable/

- **Dataflow**: DAG de operaciones sobre un flujo de datos
  - ***Streams:*** un flujo de información que eventualmente no finaliza
  - ***Batchs:*** un conjunto de datos (*dataset*) de tamaño conocido

```
DataStream<String> lines = env.addSource(
                                new FlinkKafkaConsumer<>(…));

DataStream<Event> events = lines.map((line) -> parse(line));

DataStream<Statistics> stats = events
        .keyBy("id")
        .timeWindow(Time.seconds(10))
        .apply(new MyWindowAggregationFunction());

stats.addSink(new RollingSink(path));
```

Source

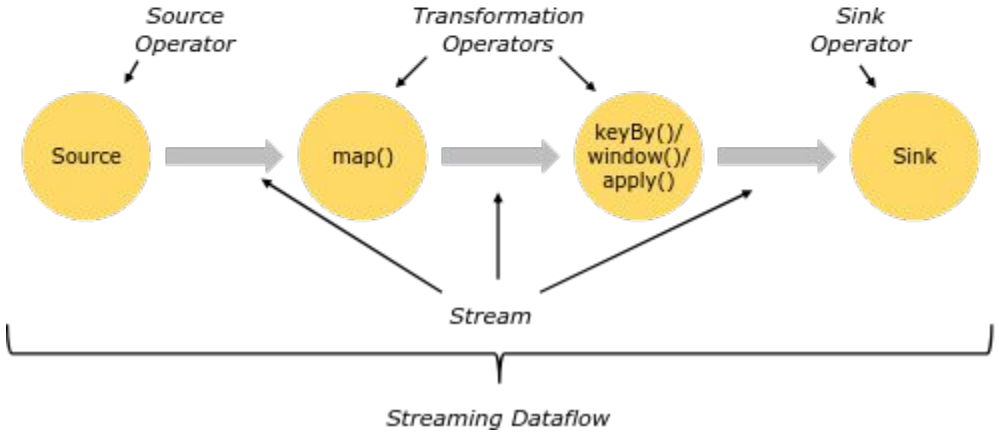
Transformation

Transformation

Sink

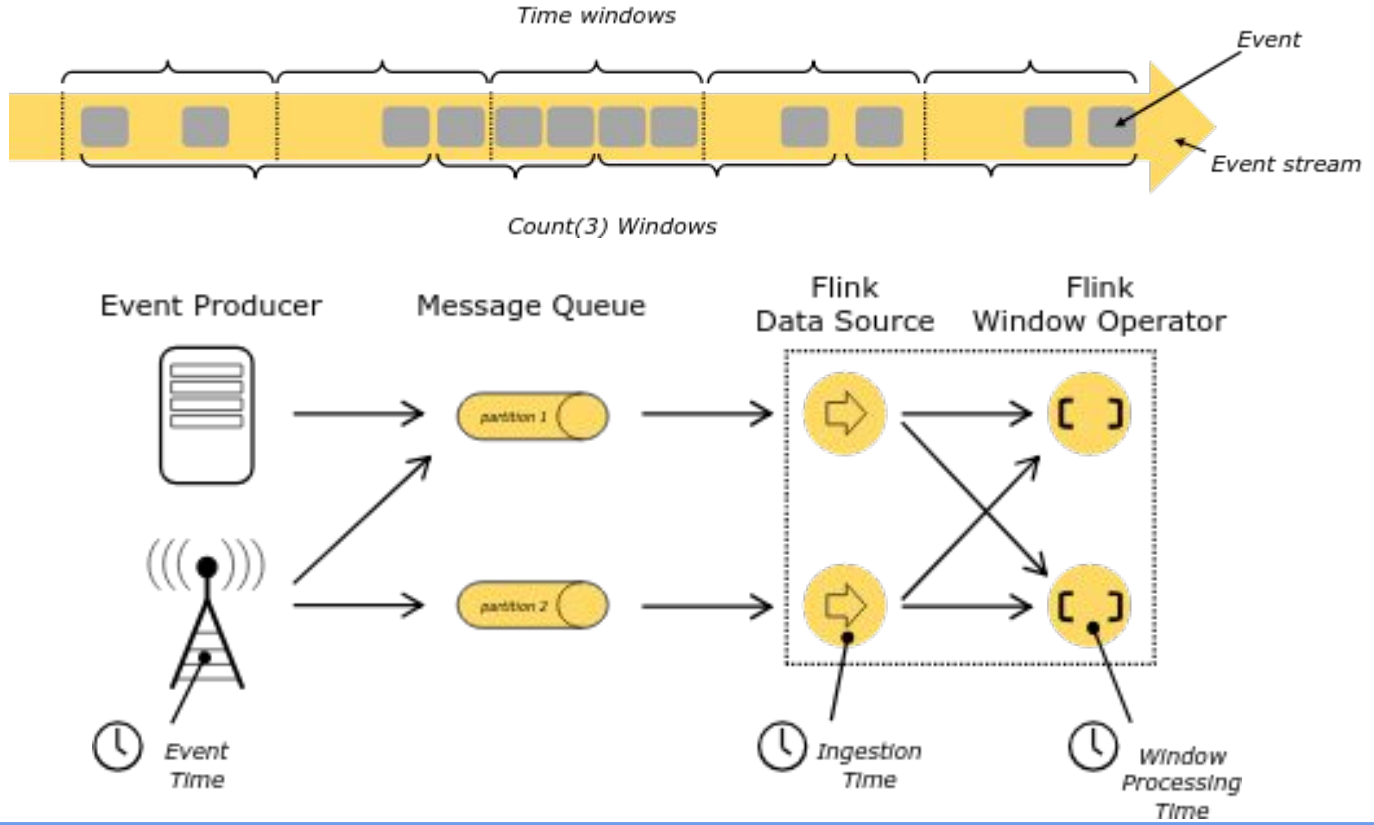
Flink | Bloques de un *Pipeline*

- ***Source:*** bloque capaz de inyectar datos al *pipeline*
- ***Transformation:*** también conocido como operador. Es un nodo de modificación de datos o filtrado de los mismos
- ***Sink:*** bloque de destino de la información. Almacenamiento final del *pipeline*



Reference: https://ci.apache.org/projects/flink/flink-docs-stable/

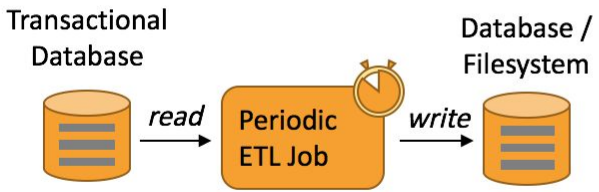
Flink | Ventanas para *Streaming*



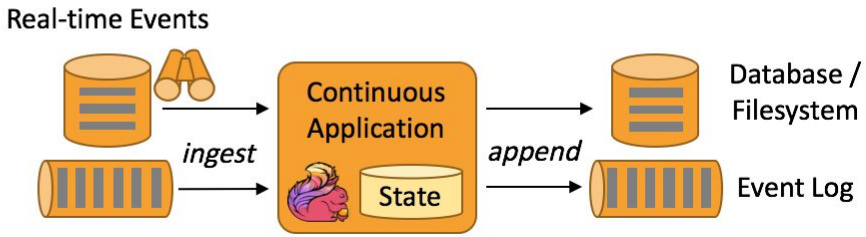
Reference: https://ci.apache.org/projects/flink/flink-docs-stable/

- ***Extract Transform Load (ETL):*** operaciones programadas de carga y modificación de datos para posterior análisis con origen y destino definidos en una DB.
- ***Data Pipelines:*** tareas de procesamiento recurrentes, basadas en la ocurrencia de eventos.

Periodic ETL

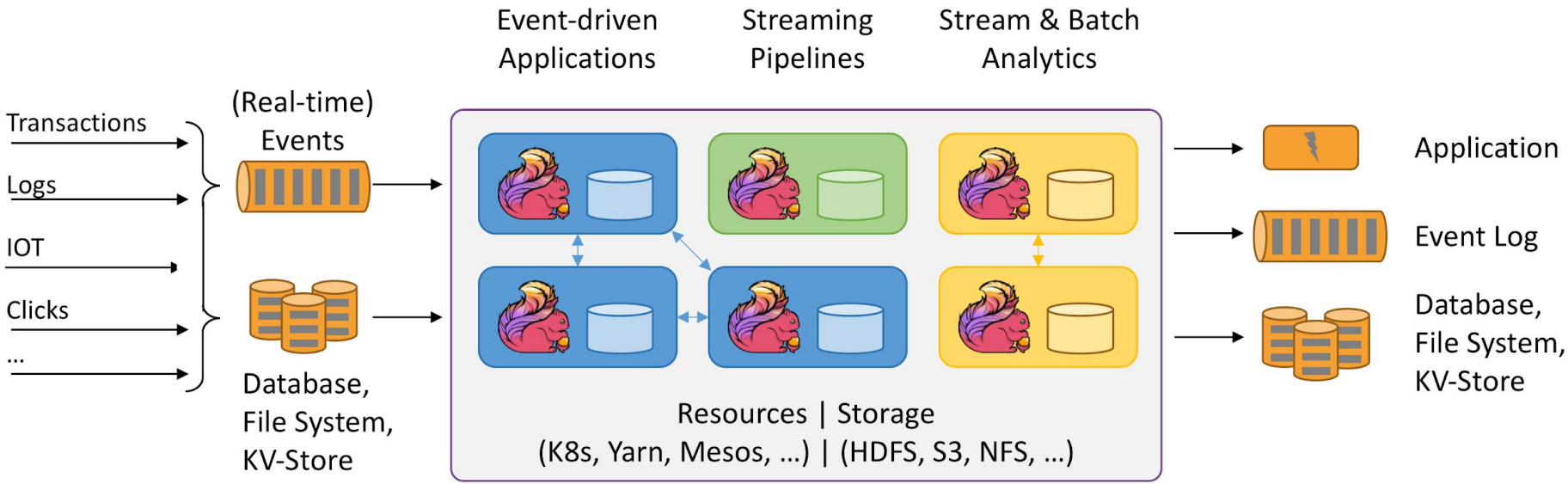


Data Pipeline



Reference: https://ci.apache.org/projects/flink/flink-docs-stable/

Flink | Ej. Ecosistema de *BigData* con Flink



Reference: https://ci.apache.org/projects/flink/flink-docs-stable/

Beam | Introducción

Modelo de definición de *pipelines* de procesamiento de datos con portabilidad de lenguajes y motores de ejecución (*runners*).

beam

Soporta distintos lenguajes de programación:

- Java
- Python
- Go

Soporta distintos *Runners:*

- Ejecución directa (*Stand-alone o DirectRunner)*
- Motores de *cluster*: Apache Hadoop, Apache Flink, Apache Spark
- Plataformas *cloud*: Google Dataflow, IBM Streams

Beam | Bloques de un *Pipeline*

Los componentes básicos son similares a los definidos en Flink:

- ***Input* y *Output:*** origen y destino respectivamente de los datos (símil *Source* y *Sink*).
- **PCollection:** colecciones paralelizables de elementos (símil *Streams*)
- ***Transformations:*** modificaciones aplicadas a las PCollections, elemento a elemento (símil *Operators*)

Bibliografía

- Open MPI:
  - https://www.open-mpi.org/doc/current/
- Apache Flink
  - https://ci.apache.org/projects/flink/flink-docs-stable/
- Apache Beam
  - https://beam.apache.org/documentation/
- Ejemplos de MPI
  - https://github.com/7574-sistemas-distribuidos/MPI-examples
- Ejemplos de Apache Flink y Apache Beam
  - https://github.com/7574-sistemas-distribuidos/apache-beam-example