Failure Recovery of a Stream Analytics System at the example Apache Flink

Data Analytics Systems form the core of the business of many IT-Companies such as Google or Facebook. As such, these companies heavily depend on these systems running all the time. This is a problem, because no IT-System is infallible. Be it bugs in the software or operating system or hardware failures. As a result, a crash of the Data Analytics System is just a question of when and not if. To minimize the losses to the company from this failures, a fast recovery of the system, restoring its state, while losing only a small amount of progress is important.

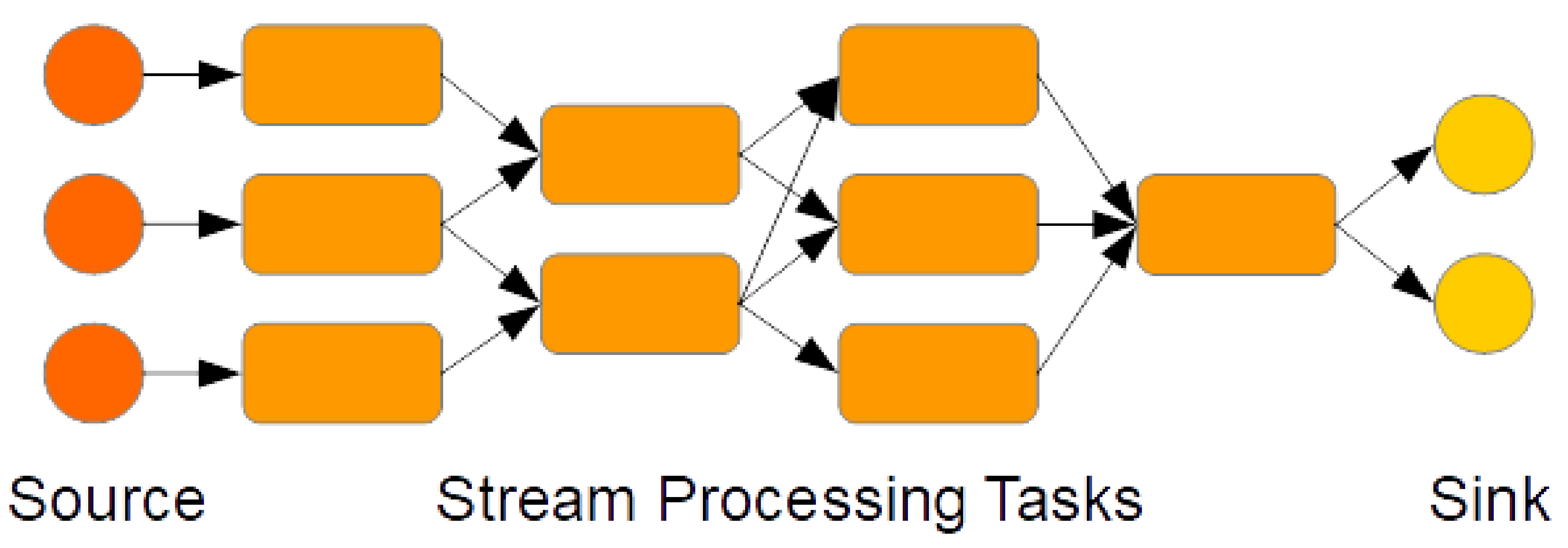
This poster will present the way Apache Flink – a Batch and Stream Processing Framework – solved this problem.

Stream Analytics Example

The diagram on the right shows schematically how a streaming pipeline might look like, especially in Flink.

The input data stream can come from a variety of sources. Be it logs, click streams from websites or sensor data.

This it is then digested by the Flink-Application itself. This application consists of multiple stream processing tasks which are ordered in a directed acyclic graph and so many task ingest as input the output of other tasks. Many tasks also contain state information, so that they can compute information over multiple instances of data and events. This state information is at risk of being lost during a failure – often a crash of the entire system. To prevent this backups in the form of checkpoints of the state of all tasks in the system are made, which, in the case of a failure, can be used to restore the system.

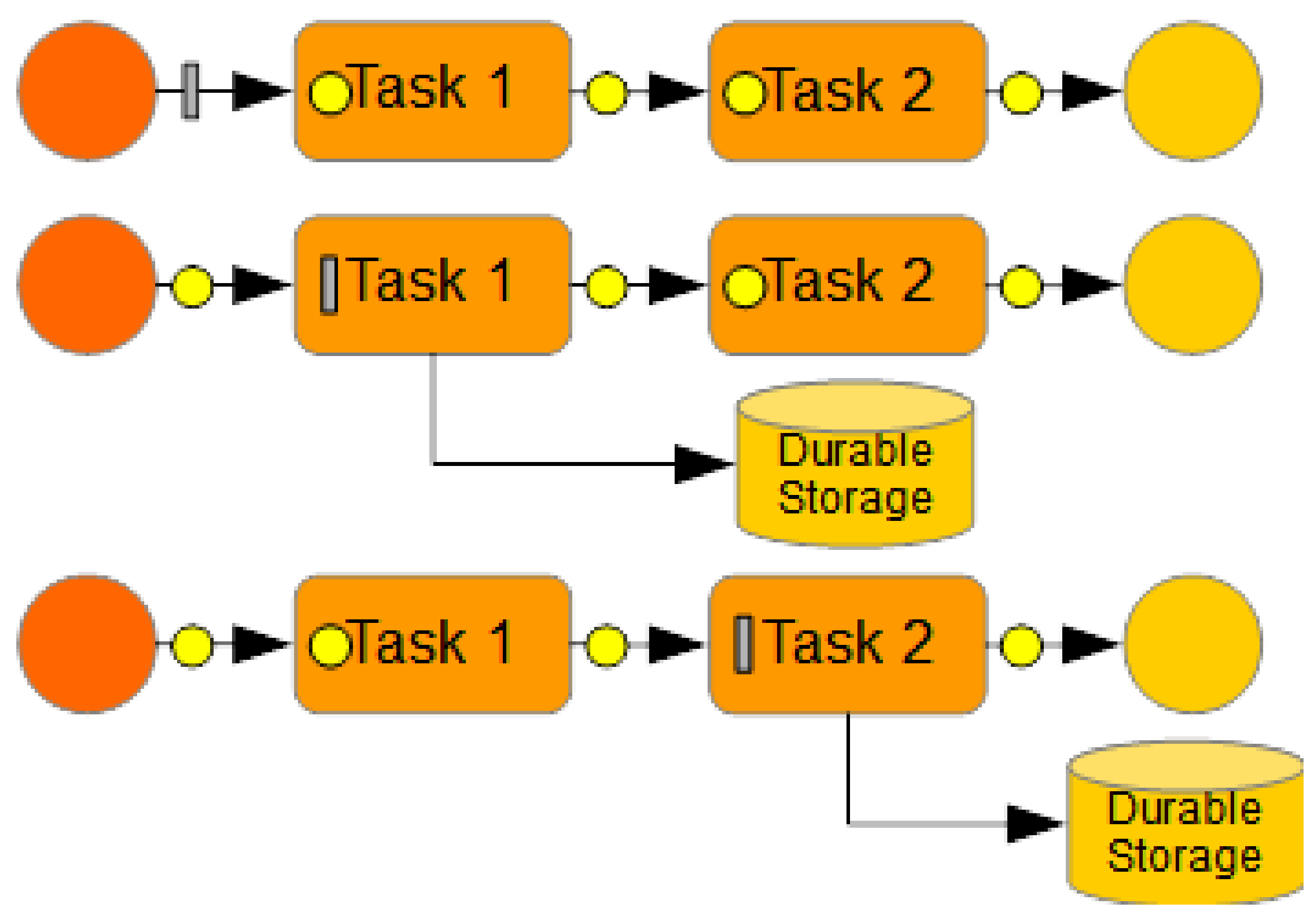


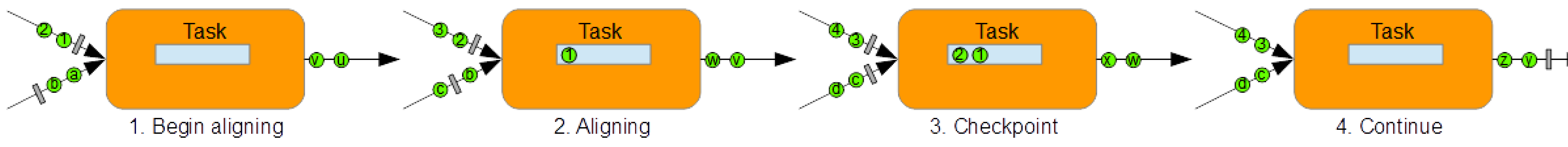
Finally, the output of the Flink application is then used for a variety of purposes, which often include further processing. Saving the output in a database or event log is commonly done. The other typical use case is a live report or dashboard of the results giving a real time overview over the data.

Checkpoint Creation

To allow the recovery of the system after a crash, checkpoints of the system have to be taken. These will be later used to restore the data analytics system. The smaller the interval between checkpoints, the less progress is lost during a crash. As the creation of checkpoint slows the system or even halts its work, this becomes an important balancing act between the amount of progress lost during a crash and the slowdown of the system.

A simple way to create checkpoints would be to stop the entire system and then write the state of every task to durable storage. As this can take many minutes, if not over an hour, stopping the system for so long is usually not an option. Instead Apache Flink uses a system of checkpoint barriers, based on the Chandy-Lamport-Algorithm, to create the checkpoint while the system is running. Checkpoint barriers (the grey rectangles in the diagrams) are introduced via the data sources, with a checkpoint barrier assigned to every data source at the same logical time if multiple data sources are used. If the barrier reaches a task, said task will write its state to durable storage. This will momentarily slow down the task, but it will not halt the entire system. Afterwards the barrier will progress further through the task graph and the next task will write its state to durable storage. However, even this system has limits. If the system uses multiple data sources, after processing the first checkpoint barrier, a task will have to pause processing incoming data from this data source until the checkpoint barriers from all other sources have passed it as well at which point the checkpoint will be written. During this time, all unprocessed data is kept in a buffer. This process is shown in the second diagram.





Recovery

The recovery of the system after failure is rather simple. The state of every task is reset to its value from the last checkpoint. In addition, the input stream of data is also reset, in this case to the position the checkpoint barrier was emitted. This is another, sometimes overlooked, task of the checkpoint, it guarantees, that no input data prior to the emission of the checkpoint barrier is needed to restore the system and this data can thus be safely discarded. This is especially useful because the amount of input data can be so large, that only a very limited amount of can be retained.

Another important aspect is the prevention of duplicate output. Some processed data may have been sent to the sink after the checkpoint was completed but before the failure. There are two way for preventing outputting this data twice, which may lead to errors in the data in the sinks. If said sinks do not support transactional behavior, the easiest solution is to only output the data to the sinks when the corresponding checkpoint has been written. Support for transactional behavior by the sinks, for example in Apache Kafka, offers a more elegant solution. In this case , each individual datum also contains a transactional fingerprint. The sink now compares this fingerprint with the ones it received previously, and if duplicates are found, these duplicates are discarded. As a result, data can be outputted independently of the checkpoints.

Jan Behrens
Bachelor

Hasso Plattner Institute, Potsdam, Germany

E-Mail: jan.behrens@student.hpi.de

Additional Source: https://ci.apache.org/projects/flink/ Apache Flink Documentation