Introduction to Apache Flink

Christoph Boden | stv. Projektkoordinator Berlin Big Data Center (BBDC) | TU Berlin DIMA
Introducing the

http://bbdc.berlin
“Data Scientist” – “Jack of All Trades!”

Domain Expertise (e.g., Industry 4.0, Medicine, Physics, Engineering, Energy, Logistics)
Mathematical Programming
Linear Algebra
Stochastic Gradient Descent
Error Estimation
Active Sampling
Regression
Monte Carlo
Statistics
Sketches
Hashing
Convergence
Decoupling
Iterative Algorithms
Curse of Dimensionality

Machine Learning
Statistics
Data Analysis

Relational Algebra / SQL
Data Warehouse/OLAP
NF²/XQuery
Resource Management
Hardware Adaptation
Fault Tolerance
Memory Management
Parallelization
Scalability
Memory Hierarchy
Data Analysis Language
Compiler
Query Optimization
Indexing
Data Flow
Control Flow
Real-Time

Application
Data Science
DM
Scalable Data Management
ML
Machine Learning + Data Management = X

Think ML-algorithms in a scalable way
declarative
Process iterative algorithms in a scalable way

Goal: Data Analysis without System Programming!

Technology X

Mathematical Programming
Linear Algebra
Error Estimation
Active Sampling
Regression Monte Carlo

Feature Engineering
Representation Algorithms (SVM, GPs, etc.)

Statistic
Hashing
Isolation
Convergence
Curse of Dimensionality
Iterative Algorithms
Control flow

ML

Relational Algebra/SQL
Data Warehouse/OLAP
NF²/XQuery
Scalability
Hardware adaption
Fault Tolerance
Resource Management

Declarative Languages
Automatic Adaption
Scalable processing

DM

Parallelization
Compiler
Memory Management
Memory Hierarchy
Data Analysis Language
Query Optimization
Dataflow
Indexing

Declarative Languages
Automatic Adaption
Scalable processing
**X = Big Data Analytics – System Programming!**

(„What“, not „How“)

**Technology X**

- Think ML-algorithms in a scalable way
- Analysis of “data in motion”
- Multimodal analysis
- Numerical stability
- Declarative specification
- Automatic optimization, parallelization and hardware adaption of dataflow and control flow with user-defined functions, iterations and distributed state
- Scalable algorithms and debugging
- Algorithmic fault tolerance
- Iterative algorithms in a scalable way
- Consistent intermediate results
- Software-defined networking

**Description of „What“?**
(declarative specification)
Technology X

**Description of „How“?**
(State of the art in scalable data analysis)
Hadoop, MPI

- Larger human base of „data scientists“
- Reduction of „human“ latencies
- Cost reduction

Data Analyst

Machine
Application Examples:
Technology Drivers and Validation

Technology X

ML

- Think ML-algorithms in a scalable way
- Declarative
- Process iterative algorithms in a scalable way

DM

text data flows

multimodal data

integration: video, images, text

windowing

text data flows

- hierarchical numerical simulation data
- numerical stability

Application Example:
Marketplace for information
- economics-based

Application Example:
Health
- society-based

Application Example:
Material science
- science-based
Open source data infrastructure

Applications
- Hive
- Cascading
- Mahout
- Pig
- ...

Data processing engines
- MapReduce
- Flink
- Spark
- Storm
- Tez

App and resource management
- Yarn
- Mesos

Storage, streams
- HDFS
- HBase
- Kafka
- ...

Other relevant technologies:
- Mahout
- Cascading
- Tez
Engine paradigms & systems

MapReduce (OSDI'04)

Dryad, Nephele (EuroSys'07)

Apache Hadoop 1

Apache Tez

PACTs (SOCC’10, VLDB’12)

Apache Flink

RDDs (HotCloud’10, NSDI’12)

Apache Spark
# Engine comparison

<table>
<thead>
<tr>
<th>API</th>
<th>MapReduce on k/v pairs</th>
<th>Transformations on k/v pair collections</th>
<th>Iterative transformations on collections</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Paradigm</strong></td>
<td>MapReduce</td>
<td>RDD</td>
<td>Cyclic dataflows</td>
</tr>
<tr>
<td><strong>Optimization</strong></td>
<td>none</td>
<td>Optimization of SQL queries</td>
<td>Optimization in all APIs</td>
</tr>
<tr>
<td><strong>Execution</strong></td>
<td>Batch sorting</td>
<td>Batch with memory pinning</td>
<td>Stream with out-of-core algorithms</td>
</tr>
</tbody>
</table>
APACHE FLINK
An open source platform for scalable batch and stream data processing.
Data sets and operators

Program

Parallel Execution
Rich operator and functionality set

Map, Reduce, Join, CoGroup, Union, Iterate, Delta Iterate, Filter, FlatMap, GroupReduce, Project, Aggregate, Distinct, Vertex-Update, Accumulators
Base-Operator: Map
Base-Operator: Reduce
Base-Operator: Cross
Base-Operator: Join
Base-Operator: **CoGroup**
WordCount in Java

```java
ExecutionEnvironment env =
    ExecutionEnvironment.getExecutionEnvironment();

DataSet<String> text = readTextFile(input);

DataSet<Tuple2<String, Integer>> counts =
    text.map(l -> l.split("\W+"))
        .flatMap((String[] tokens, Collector<Tuple2<String, Integer>> out) -> {
            Arrays.stream(tokens)
                .filter(t -> t.length() > 0)
                .forEach(t -> out.collect(new Tuple2<>(t, 1)));
        })
        .groupBy(0)
        .sum(1);

env.execute("Word Count Example");
```
val env = ExecutionEnvironment
  .getExecutionEnvironment

val input = env.readTextFile(textInput)

val counts = text
  .flatMap { line => line.split("\W+") }
  .filter { term => term.nonEmpty }
  .map { term => (term, 1) }
  .groupBy(0)
  .sum(1)

env.execute()
Long operator pipelines

```java
DataSet<Tuple...> large = env.readCsv(...);
DataSet<Tuple...> medium = env.readCsv(...);
DataSet<Tuple...> small = env.readCsv(...);

DataSet<Tuple...> joined1 =
    large.join(medium)
    .where(3).equals(1)
    .with(new JoinFunction() { ... });

DataSet<Tuple...> joined2 =
    small.join(joined1)
    .where(0).equals(2)
    .with(new JoinFunction() { ... });

DataSet<Tuple...> result = joined2.groupBy(3)
    .max(2);
```
Beyond Key/Value Pairs

```java
DataSet<Page> pages = ...;
DataSet<Impression> impressions = ...;

DataSet<Impression> aggregated =
    impressions
    .groupBy("url")
    .sum("count");

pages.join(impressions).where("url").equalTo("url")
```

// custom data types

```java
class Impression {
    public String url;
    public long count;
}
class Page {
    public String url;
    public String topic;
}
```
Flink’s optimizer

• inspired by optimizers of parallel database systems
  – cost models and reasoning about interesting properties

• physical optimization follows cost-based approach
  – Select data shipping strategy (forward, partition, broadcast)
  – Local execution (sort merge join/hash join)
  – keeps track of interesting properties such as sorting, grouping and partitioning

• optimization of Flink programs more difficult than in the relational case:
  – no fully specified operator semantics due to UDFs
  – unknown UDFs complicate estimating intermediate result sizes
  – no pre-defined schema present
Optimization example

val orders = DataSource(...)  
val items = DataSource(...)  

val filtered = orders filter { ... }  
val prio = filtered join items where { _.id } isEqualTo { _.id }  
  map {(o,li) => PricedOrder(o.id, o.priority, li.price)}  
val sales = prio groupBy {p => (p.id, p.priority)} aggregate ({$.price},SUM)
Memory management

- Flink manages its own memory
- User data stored in serialize byte arrays
- In-memory caching and data processing happens in a dedicated memory fraction
- Never breaks the JVM heap
- Very efficient disk spilling and network transfers
Built-in vs. driver-based iterations

Loop outside the system, in driver program

Iterative program looks like many independent jobs

Dataflows with feedback edges

System is iteration-aware, can optimize the job
"Iterate" operator

- Built-in operator to support looping over data
- Applies step function to partial solution until convergence
- Step function can be arbitrary Flink program
- Convergence via fixed number of iterations or custom convergence criterion
“Delta Iterate” operator

- Compute next workset and changes to the partial solution until workset is empty
- Generalizes vertex-centric computing of Pregel and GraphLab
ReCap: What is Apache Flink?

Apache Flink is an open source platform for scalable batch and stream data processing.

- The core of Flink is a distributed streaming dataflow engine.
  - Executing dataflows in parallel on clusters
  - Providing a reliable foundation for various workloads

- **DataSet** and **DataStream** programming abstractions are the foundation for user programs and higher layers

http://flink.apache.org
Working on and with Apache Flink

• Flink homepage
  https://flink.apache.org

• Flink Mailing Lists
  https://flink.apache.org/community.html#mailing-lists

• Flink Meetup in Berlin
  http://www.meetup.com/de/Apache-Flink-Meetup/
Flink community

#unique contributor ids by git commits
Evolution of Big Data Platforms

1G
Relational Databases

2G
Hadoop
Scale-out, Map/Reduce, UDFs

3G
Spark
In-memory Performance and Improved Programming Model

4G
Flink
In-memory + Out of Core Performance, Declarativity, Optimisation, Iterative Algorithms, Streaming/Lambda
Is Apache Flink Europe’s Wild Card into the Big Data Race?

How an ultra-fast data engine for Hadoop could secure Europe’s place in the future of open-source

The cards are dealt anew!

https://medium.com/chasing-buzzwords/is-apache-flink-europes-wild-card-into-the-big-data-race-a189fcf27c4c
Forbes on Apache Flink:

• „[…] Flink, which is also a top-level project of the Apache Software Foundation, has just recently begun to attract many of the same admiring comments directed Spark’s way 12-18 months ago. Despite sound technical credentials, ongoing development, big investments, and today’s high-profile endorsement from IBM, it would be unwise (and implausible) to crown Spark as the winner just yet. […]”


• http://www.datanami.com/2015/06/12/8-new-big-data-projects-to-watch/
• Two day developer conference with in-depth talks from
  – developers and contributors
  – industry and research users
  – related projects

• Flink training sessions (in parallel)
  – System introduction, real-time stream processing, APIs on top

• Flink Forward registration & call for abstracts is open now at
  http://flink-forward.org/