Emma: Declarative Dataflows for Scalable Data Analysis

Alexander Alexandrov, Georgi Krastev, Bernd Louis, Volker Markl

FG DIMA, TU Berlin
MOTIVATION
A Billion $$$ Mantra

SQL

Relations

RDBMS

Declarative Data Processing

An effective, formal foundation based on relational algebra and calculus (Codd ’71).

A simple, high-level language for querying data (Chamberlin ’74).

An efficient, low-level execution environment tailored towards the data (Selinger ’79).
With 40+ Years of Success

Declarative Data Processing
Is Being Revised

Declarative Data Processing

SQL

Relations

RDBMS

Second-Order Functions

Distributed Collections

Parallel Dataflow Engines
USE CASE: MINING MOVIE METADATA
Data: People and Movies related by Credits
Data: People and Movies related by Credits
Data: People and Movies related by Credits
Data: People and Movies related by Credits
Task: Finding Director’s Muses

• Let $mc_d^a$ be the number of movies shared between a specific director-actor combination $(d, a)$.

\[
M = \{ (d, a, m) \mid a \text{ acts in } m \land d \text{ directs } m \} \\
mc_d^a = | \{ m \mid M(d, a, m) \} |
\]

• Find $(d, a)$ pairs such that

  – The actor was cast in at least two of the director’s movies;

  \[ d \in P, a \in P. \]

  \[ mc_d^a > 1 \]

  – No other actor was cast in more than $m + 1$ movies of the same director.

  \[ \not\exists x \in P. mc_x^d > mc_d^a + 1 \]
Examples

- Director
  - Christian Petzold
  - Michelangelo Antonioni
  - Tim Burton

- Muse
  - Nina Hoss (collaborated in 5 movies)
  - Monica Vitti (collaborated in 5 movies)
  - Helena Bonham Carter (collaborated in 7 movies)
THE LOST DECLARATIVITY
DSL Design: Choices & Pitfalls

- Domain Specific Languages (DSLs) can be designed and implemented in various different ways.

- The general design strategy greatly affects
  - simplicity and usability,
  - optimization and abstraction potential,
  - user productivity.
The following examples illustrate how modern DSLs for distributed collection processing are affected by their design strategy.
Collection Processing DSLs
$M = \{ (d, a, m) | a \text{ acts in } m \land d \text{ directs } m \}$
SELECT d.name AS director,
       a.name AS actor,
       m.title.title AS movie
FROM   people AS a,
       credits AS ac,
       movies AS m,
       credits AS dc,
       people AS d
WHERE  d.id = dc.personID
AND    m.id = dc.movieID
AND    a.id = ac.personID
AND    m.id = ac.movieID
AND    dc.creditType = 'director'
AND    ac.creditType = 'actor'

\[
M = \{(d, a, m) \mid a \text{ acts in } m \land d \text{ directs } m\}
\]
Spark SQL

```scala
val M = spark.sql(
  s""
  | SELECT d.name AS director,
  |       a.name AS actor,
  |       m.title.title AS movie
  | FROM people AS a,
  |       credits AS ac,
  |       movies AS m,
  |       credits AS dc,
  |       people AS d
  | WHERE d.id = dc.personID
  | AND m.id = dc.movieID
  | AND a.id = ac.personID
  | AND m.id = ac.movieID
  | AND dc.creditType = 'director'
  | AND ac.creditType = 'actor'
"""
).stripMargin
```

**RUNTIME FOR M (IN SECONDS)**

<table>
<thead>
<tr>
<th>Variant 1</th>
<th>Variant 2</th>
<th>Variant 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>11.74</td>
<td>11.54</td>
<td>11.87</td>
</tr>
</tbody>
</table>

**Variant 1:**
Baseline version.

\[ M = \{ (d, a, m) | a \text{ acts in } m \land d \text{ directs } m \} \]
Spark SQL

val M = spark.sql(
  s""
  | SELECT d.name AS director,
  |       a.name AS actor,
  |       m.title AS movie
  | FROM people AS a,
  |     credits AS ac,
  |     movies AS m,
  |     credits AS dc,
  |     people AS d
  |WHERE d.id = dc.personID
  |AND a.id = ac.personID
  |AND m.id = dc.movieID
  |AND m.id = ac.movieID
  |AND ac.credit = 'actor'
  |AND dc.credit = 'director'
  """".stripMargin)

RUNTIME FOR M (IN SECONDS)

Variant 2:
After swapping WHERE clauses.

\[ M = \{ (d, a, m) | a \text{ acts in } m \land d \text{ directs } m \} \]
Spark SQL

```scala
val M = spark.sql(
  s""
  | SELECT d.name AS director,
  |       a.name AS actor,
  |       m.title.title AS movie
  | FROM people AS a,
  |       people AS d,
  |       credits AS ac,
  |       credits AS dc,
  |       movies AS m
  | WHERE d.id = dc.personID
  | AND a.id = ac.personID
  | AND m.id = dc.movieID
  | AND m.id = ac.movieID
  | AND ac.creditType = 'actor'
  | AND dc.creditType = 'director'
  """.stripMargin)
```

**RUNTIME FOR M (IN SECONDS)**

- Variant 1:
- Variant 2:
- Variant 3:

After swapping FROM clauses.

\[
M = \{ (d, a, m) \mid a \text{ acts in } m \land d \text{ directs } m \}\]
Collection Processing DSLs

Standalone

SQL
Collection Processing DSLs

- Standalone
- SQL
  - Declarative 👍
  - Optimizable 👍
  - Integrated 👎
Collection Processing DSLs

- Standalone
  - SQL
- Embedded
- Shallow
  - Declarative
  - Optimizable
  - Integrated
Collection Processing DSLs

Based on Types

Flink:
- Table
- DataSet

Spark:
- DataFrame
- Dataset
- RDD

Embedded
Deep
Shallow

Standalone
- SQL
Spark SQL

```scala
val M = spark.sql(""
| SELECT d.name AS director,
|     a.name AS actor,
|     m.title.title AS movie
| FROM  people AS a,
|       people AS d,
|       credits AS ac,
|       credits AS dc,
|       movies AS m
| WHERE d.id = dc.personID
| AND  a.id = ac.personID
| AND  m.id = dc.movieID
| AND  m.id = ac.movieID
| AND  ac.creditType = 'actor'
| AND  dc.creditType = 'director'
""".stripMargin)
```

### Runtime for M (in seconds)

<table>
<thead>
<tr>
<th>Variant 1</th>
<th>Variant 2</th>
<th>Variant 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>11.74</td>
<td>11.54</td>
<td>11.87</td>
</tr>
</tbody>
</table>

**Variant 1:**
Baseline version.

\[ M = \{ (d, a, m) | a \text{ acts in } m \land d \text{ directs } m \} \]
Spark DataFrame

```scala
val M =
  join(
    join(
      join(
        join(
          a,
          ac.filter("ac.creditType" === "actor"),
          $"ac.personID" === $"a.id"),
          m,
          $"ac.movieID" === $"m.id"),
          dc.filter("dc.creditType" === "director"),
          $"dc.movieID" === $"m.id"),
        d,
        $"dc.personID" === $"d.id"
      ).select($"d.name" as "director",
                $"a.name" as "actor",
                $"m.title.title" as "movie"
    )
```

**Runtime for M (in seconds):**

<table>
<thead>
<tr>
<th>Variant</th>
<th>SPARK SQL</th>
<th>SPARK DATAFRAME</th>
<th>EMMA</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>11.74</td>
<td>11.38</td>
<td>11.54</td>
</tr>
<tr>
<td>2</td>
<td>11.54</td>
<td>11.54</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>11.54</td>
<td></td>
</tr>
</tbody>
</table>

**Variant 1:** Baseline version.

\[ M = \{ (d, a, m) \mid a \text{ acts in } m \land d \text{ directs } m \} \]
Spark DataFrame

```scala
val M =
  join(
    join(
      join(
        a,
        ac,
        "ac.personID" === "a.id"),
      m,
      "ac.movieID" === "m.id"),
    dc,
    "dc.movieID" === "m.id"),
  d,
  "dc.personID" === "d.id"
).select(
  "d.name" as "director",
  "a.name" as "actor",
  "m.title.title" as "movie"
).filter(
  "dc.creditType" === "director" &&
  "ac.creditType" === "actor"
)
```

**RUNTIME FOR M (IN SECONDS)**

<table>
<thead>
<tr>
<th>Variant 1</th>
<th>Variant 2</th>
<th>Variant 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>11.74</td>
<td>11.54</td>
<td>11.87</td>
</tr>
<tr>
<td>11.38</td>
<td>11.54</td>
<td>12.12</td>
</tr>
<tr>
<td>18.37</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Variant 2:
After reordering **filter** calls.

\[ M = \{ (d, a, m) \mid a \text{ acts in } m \land d \text{ directs } m \} \]
Spark DataFrame

```scala
val M =
  join(
    join(
      join(
        m,
        dc,
        "$dc.movieID" === "$m.id"),
        d,
        "$dc.personID" === "$d.id"),
        ac,
        "$ac.movieID" === "$m.id"),
        a,
        "$ac.personID" === "$a.id"
  ).select(
    "$d.name" as "director",
    "$a.name" as "actor",
    "$m.title.title" as "movie"
  ).filter(
    "$dc.creditType" === "director" &&
    "$ac.creditType" === "actor"
)
```

**RUNTIME FOR M (IN SECONDS)**

<table>
<thead>
<tr>
<th>Variant 1</th>
<th>Variant 2</th>
<th>Variant 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>11.74</td>
<td>11.54</td>
<td>11.87</td>
</tr>
<tr>
<td>11.38</td>
<td>12.12</td>
<td>11.54</td>
</tr>
<tr>
<td>12.12</td>
<td>11.38</td>
<td>11.87</td>
</tr>
</tbody>
</table>

Variant 3:  
After reordering `join` calls.

\[ M = \{ (d, a, m) | a \text{ acts in } m \land d \text{ directs } m \} \]
Collection Processing DSLs

- Standalone
  - SQL
- Embedded
- Deep
- Shallow

Based on Types

<table>
<thead>
<tr>
<th>Flink:</th>
<th>Spark:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table</td>
<td>DataFrame</td>
</tr>
<tr>
<td>DataSet</td>
<td>Dataset</td>
</tr>
</tbody>
</table>
Collection Processing DSLs

Based on Types

Flink: Table | DataSet
Spark: DataFrame | Dataset | RDD

Declarative 🍀
Optimizable 🍀🍀
Integrated second-order language 🍀
Integrated first-order language 🍀
Collection Processing DSLs

- Standalone
  - SQL

- Embedded
  - Deep
  - Shallow

Based on Types

<table>
<thead>
<tr>
<th>Flink</th>
<th>Spark</th>
<th>Table</th>
<th>DataFrame</th>
<th>Dataset</th>
<th>Dataset</th>
<th>RDD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Spark Dataset / RDD

// Dataset API (variant 1)
val C = M
  .groupBy { case (d, a, m) => (d, a) }
  .map { case (key, it) => (key, it.size.toLong) }
  .filter { case (_, mc) => mc > 1 }

// RDD API (variant 1)
val C = M
  .groupBy { case (d, a, m) => (d, a) }
  .map { case (key, it) => (key, it.size.toLong) }
  .filter { case (_, mc) => mc > 1 }

Variant 1:
Build groups, then compute (e.g., count) per group.

\[
C = \{ (d, a, mc_d^a) | M(d, a, -); mc_d^a > 1 \} 
\]
Spark Dataset / RDD

// Dataset API (variant 1)
val C = M
groupByKey { case (d, a, m) => (d, a) }
.mapGroups((key, it) => (key, it.size.toLong))
.filter { case (_, mc) => mc > 1 }

// RDD API (variant 1)
val C = M
groupBy { case (d, a, m) => (d, a) }
.map { case (key, it) => (key, it.size.toLong) }
.filter { case (_, mc) => mc > 1 }

RUNTIME FOR C (IN SECONDS)

Variant 1:
Build groups, then compute (e.g., count) per group.

\[ C = \{ (d, a, mc_d^a) \mid M(d, a, -); mc_d^a > 1 \} \]
Spark Dataset / RDD

// Dataset API (variant 2)
val C = M
  .groupByKey { case (d, a, m) => (d, a) }
  .count()
  .filter { case (_, mc) => mc > 1 }

// RDD API (variant 1)
val C = M
  .groupBy { case (d, a, m) => (d, a) }
  .map { case (key, it) => (key, it.size.toLong) }
  .filter { case (_, mc) => mc > 1 }

RUNTIME FOR C (IN SECONDS)

Variant 2:
Build groups and compute (e.g., count) in one step.

\[
C = \{ (d, a, mc^a_d) \mid M(d, a, -); \ mc^a_d > 1 \} 
\]
Spark Dataset / RDD

// Dataset API (variant 2)
val C = M
    .groupByKey { case (d, a, m) => (d, a) }
    .count()
    .filter { case (_, mc) => mc > 1 }

// RDD API (variant 2)
val C = M
    .map { case (d, a, m) => ((d, a), 1L) }
    .reduceByKey((mc1, mc2) => mc1 + mc2)
    .filter { case (_, mc) => mc > 1 }

Variant 2:
Build groups and compute (e.g., count) in one step.

\[ C = \{ (d, a, mc_d^a) | M(d, a, \_); \ mc_d^a > 1 \} \]
Collection Processing DSLs

Based on Types

<table>
<thead>
<tr>
<th></th>
<th>Flink</th>
<th>Spark</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table</td>
<td></td>
<td>DataFrame</td>
</tr>
<tr>
<td>DataSet</td>
<td></td>
<td>Dataset</td>
</tr>
<tr>
<td>RDD</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Standalone

SQL

Embedded

Deep

Shallow
Collection Processing DSLs

- Standalone
  - SQL
- Embedded
  - Deep
  - Shallow

Based on Types

<table>
<thead>
<tr>
<th>Flink</th>
<th>Spark</th>
<th>Table</th>
<th>DataFrame</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Table</td>
<td>DataFrame</td>
</tr>
</tbody>
</table>

- Declarative 🔴
- Optimizable 🔴
- Integrated 🔴
Collection Processing DSLs

Standalone

SQL

Embedded

Deep

Shallow

Based on Types

Flink: Table, DataSet
Spark: DataFrame, Dataset, RDD

Based on Quotation

Emma DataBag compiles to Flink / Spark
Emma Features

```scala
val M = spark.sql(""
| SELECT d.name AS director,
|       a.name AS actor,
|       m.title.title AS movie
| FROM people AS a,
|       people AS d,
|       credits AS ac,
|       credits AS dc,
|       movies AS m
| WHERE d.id = dc.personID
| AND a.id = ac.personID
| AND m.id = dc.movieID
| AND m.id = ac.movieID
| AND ac.creditType = "actor"
| AND dc.creditType = "director"
"").stripMargin
```

### Runtime for M (in Seconds)

<table>
<thead>
<tr>
<th>Variant</th>
<th>SPARK SQL</th>
<th>SPARK DATAFRAME</th>
<th>EMMA</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>11.74</td>
<td>11.38</td>
<td>18.37</td>
</tr>
<tr>
<td>2</td>
<td>11.87</td>
<td>12.12</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>11.54</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Integrated

**SELECT – FROM – WHERE** syntax.

\[ M = \{ (d, a, m) \mid a \text{ acts in } m \land d \text{ directs } m \} \]
Emma Features

```scala
val M = spark.sql(
  s""""
  | yield d.name,
  |     a.name,
  |     m.title.title
  | FROM people AS a,
  |     people AS d,
  |     credits AS ac,
  |     credits AS dc,
  |     movies AS m
  | WHERE d.id = dc.personID
  | AND a.id = ac.personID
  | AND m.id = dc.movieID
  | AND m.id = ac.movieID
  | AND ac.creditType = "actor"
  | AND dc.creditType = "director"
  """").stripMargin)
```

**RUNTIME FOR M (IN SECONDS)**

<table>
<thead>
<tr>
<th>Variants</th>
<th>Spark SQL</th>
<th>Spark DataFrame</th>
<th>Emma</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variant 1</td>
<td>11.74</td>
<td>11.54</td>
<td>11.38</td>
</tr>
<tr>
<td>Variant 2</td>
<td>11.87</td>
<td>11.38</td>
<td>12.12</td>
</tr>
<tr>
<td>Variant 3</td>
<td>11.54</td>
<td>12.12</td>
<td>18.37</td>
</tr>
</tbody>
</table>

Integrated

`SELECT – FROM – WHERE` syntax.

\[ M = \{ (d, a, m) | a \text{ acts in } m \land d \text{ directs } m \} \]
Emma Features

```
val M = spark.sql(
  s""
    | yield d.name,
    |     a.name,
    |     m.title.title
    | for a <- people,
    |     d <- people,
    |     ac <- credits,
    |     dc <- credits,
    |     m <- movies
    | WHERE d.id = dc.personID
    | AND a.id = ac.personID
    | AND m.id = dc.movieID
    | AND m.id = ac.movieID
    | AND ac.creditType = "actor"
    | AND dc.creditType = "director"
  """".stripMargin)
```

**RUNTIME FOR M (IN SECONDS)**

<table>
<thead>
<tr>
<th></th>
<th>SPARK SQL</th>
<th>SPARK DATAFRAME</th>
<th>EMMA</th>
</tr>
</thead>
<tbody>
<tr>
<td>variant 1</td>
<td>11.74</td>
<td>11.54</td>
<td>11.38</td>
</tr>
<tr>
<td>variant 2</td>
<td>11.87</td>
<td>12.12</td>
<td>12.12</td>
</tr>
<tr>
<td>variant 3</td>
<td>11.38</td>
<td>18.37</td>
<td>18.37</td>
</tr>
</tbody>
</table>

Integrated

**SELECT – FROM – WHERE** syntax.

\[ M = \{ (d, a, m) \mid a \text{ acts in } m \land d \text{ directs } m \} \]
Emma Features

```
val M = spark.sql(
  s""
    | yield d.name,
    |     a.name,
    |     m.title.title
    | for a <- people,
    |     d <- people,
    |     ac <- credits,
    |     m <- movies
    | if  d.id == dc.personID
    | if  a.id == ac.personID
    | if  m.id == dc.movieID
    | if  m.id == ac.movieID
    | if  ac.creditType == "actor"
    | if  dc.creditType == "director"
  

  
""").stripMargin)
```

**RUNTIME FOR M (IN SECONDS)**

- **variant 1**: 1174, 1177, 1138
- **variant 2**: 1154, 1212, 1837
- **variant 3**: 16.30, 16.75, 18.37

**SELECT – FROM – WHERE** syntax.

\[
M = \{ (d, a, m) | a \text{ acts in } m \land d \text{ directs } m \}\]
Emma Features

val M = for { 
    a <- people 
    d <- people 
    ac <- credits 
    dc <- credits 
    m <- movies 
    if d.id == dc.personID 
    if a.id == ac.personID 
    if m.id == dc.movieID 
    if m.id == ac.movieID 
    if ac.creditType == "actor" 
    if ac.creditType == "director"
} yield ( 
    d.name, 
    a.name, 
    m.title.title 
}

Integrated
SELECT – FROM – WHERE syntax.

\[ M = \{ (d, a, m) \mid a \text{ acts in } m \land d \text{ directs } m \} \]
Emma Features

// RDD API (variant 1)
val C = M
  .groupBy { case (d, a, m) => (d, a) }
  .map { case (key, vs) => (key, vs.size.toLong) }
  .filter { case (_, mc) => mc > 1 }

---

Integrated optimizations for nested collection processing.

\[ C = \{ (d, a, mc_d^a) \mid M(d, a, -); mc_d^a > 1 \} \]
Emma Features

// Emma API (variant 1)
val C = M
  .groupBy { case (d, a, m) => (d, a) }
  .map { case Group(key, vs) => (key, vs.size) }
  .filter { case (_, mc) => mc > 1 }

Integrated optimizations for nested collection processing.

\[ C = \{ (d, a, mc_d^a) \mid M(d, a, \_); mc_d^a > 1 \} \]
Collection Processing DSLs

Standalone

SQL

Embedded

Deep

Shallow

Based on Types

Flink:
- Table

Spark:
- DataFrame

Based on Quotation

Emma DataBag compiles to Flink / Spark

DataSet

RDD
Collection Processing DSLs

- **Standalone**
  - SQL

- **Embedded**
  - Deep
  - Shallow

**Based on Types**

<table>
<thead>
<tr>
<th></th>
<th>Flink</th>
<th>Spark</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DataBag</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Based on Quotation**

- Emma **DataBag** compiles to Flink / Spark

- Declarative 👍
- Optimizable 👍
- Integrated 👍