SPARK+AI
SUMMIT 2018

Extending Spark SQL API with Easier to Use Array Types Operations

Marek Novotny, ABSA Capital
Jan Scherbaum, ABSA Capital

#Dev3SAIS

Intro

- ABSA (Barclays Africa) is a Pan-African financial services provider
  - With Apache Spark at the core of its data engineering
- We try to fill gaps in the Hadoop eco-system
  - Especially from the perspective of highly regulated industries
  - ABRiS – Avro SerDe for Apache Spark structured APIs
    - https://github.com/AbsaOSS/ABRiS
  - Spline – a lineage tracking for Apache Spark
    - https://absaoss.github.io/spline/

Problem Statement

- Lots of structured data (XML, JSON) from a data lake has to be cleansed and transformed
  - With lineage tracking via Spline
- Limited support for nested structures in Spark structured APIs
- Alternatives
  - RDD/Dataset higher-order functions & UDFs ⇒ No lineage tracking
  - Flatten structure with explode ⇒ Very inefficient
    - 1000 records, 2 levels, 150 elements per level ⇒ 22,500,000 records
  - Convert array elements to columns ⇒ too big logical plan
    - Size of the logical plan is directly proportional to max size of arrays
    - Variable array size

Solution

- Create a custom build of Spark with an extension
  - **concat, flatten, reverse**, zip_with_index, transform
  - github.com/AbsaOSS/spark/tree/branch-2.2.1-ABSA
  - github.com/AbsaOSS/spark/tree/branch-2.3.0-ABSA

- Contribute some array functions to Spark ☺

- Apache Spark has gotten inspired by Presto
  - 33 array/map functions are planned for the version 2.4.0
  - SPARK-23899

concat (col1, col2, …, colN)

- SPARK-23736 resolved by Marek Novotny
- Adds array types support to the existing “concat” function

concat([1, 2, 3], [4, 5])	→	[1, 2, 3, 4, 5]	[1, 2, 3] ≈ array(1, 2, 3)
concat([1, 2], [2, 3], [3, 4])	→	[1, 2, 2, 3, 3, 4]	

- Type coercion

concat([1, 2, 3], [‘a’, ‘b’, ‘c’])	→	[‘1’, ‘2’, ‘3’, ‘a’, ‘b’, ‘c’]

- Null handling

concat(…, *null*, …)	→	*null*
concat([1, *null*], [‘a’, *null*])	→	[‘1’, *null*, ‘a’, *null*]

flatten (col)

- SPARK-23821 resolved by Marek Novotny

- Performs shallow flattening

flatten([[1, 2, 3], [4, 5]])	→	[1, 2, 3, 4, 5]
flatten([['a', 'b'], [], ['b', 'a']])	→	['a', 'b', 'b', 'a']
flatten([[[1, 2], [3]], [[4]]])	→	[[1, 2], [3], [4]]

- Null handling

flatten(*null*)	→	*null*
flatten([[1, 2, 3], *null*, [4, 5]])	→	*null*
flatten([['a', *null*], [*null*, 'b']])	→	['a', *null*, *null*, 'b']

reverse (col)

- SPARK-23926 resolved by Marek Novotny

- Adds array types support to the existing “reverse” function

reverse([2, 1, 4, 3, 5])	→	[5, 3, 4, 1, 2]
reverse([‘b’, ‘a’, ‘d’, ‘c’])	→	[‘c’, ‘d’, ‘a’, ‘b’]
reverse([[‘a’], [], [‘c’, ‘b’], []])	→	[[], [‘c’, ‘b’], [], [‘a’]]

- Null handling

reverse(*null*)	→	*null*
reverse([‘a’, *null*, ‘b’, *null*])	→	[*null*, ‘b’, *null*, ‘a’]

map_entries (col)

- SPARK-23935 resolved by Marek Novotny

- Transforms maps into arrays of key-value pairs

map_entries(map(2 → ‘b’, 1 → ‘a’, 3 → ‘c’))	→	[(2, ‘b’), (1, ‘a’), (3, ‘c’)]
map_entries(map(‘a’ → 1, ‘c’ → 2, ‘b’ → 3))	→	[(‘a’, 1), (‘c’, 2), (‘b’, 3)]

- Null handling

map_entries(*null*)	→	*null*
map_entries(map(1 → *null*, 3 → ‘c’, 2 → *null*))	→	[(1, *null*), (3, ‘c’), (2, *null*)]

map_from_entries (col)

- SPARK-23934 – in progress – github.com/apache/spark/pull/21282

- Transforms arrays of key-value pairs into maps

  map_from_entries([(2, 'a'), (1, 'b'), (3, 'c')]) → map(2 → 'a', 1 → 'b', 3 → 'c')

- Null handling

  map_from_entries([(2, 'a'), (1, *null*)]) → map(2 → 'a', 1 → *null*)
  map_from_entries([(2, 'a'), (*null, 'b'*)]) → RuntimeException
  map_from_entries(*null*) → *null*

sequence (start, stop[, step])

- SPARK-23927 – in progress – github.com/apache/spark/pull/21155/
- **Integral sequence** (byte, short, int, long)
```
sequence(1, 5)          →   [1, 2, 3, 4, 5]
sequence(1, 5, 2)       →   [1, 3, 5]
```
- **Temporal sequence** (date, timestamp)
```
sequence(
    cast('2018-01-01' as date),
    cast('2018-03-01' as date),
    interval 1 month)   →   [2018-01-01,  2018-02-01,  2018-03-01]
```
- **Reverse sequence**
```
sequence(5, 1)          →   [5, 4, 3, 2, 1]
sequence(5, 1, -2)      →   [5, 3, 1]
```
- Supports time zones and DST

array_max (col)
array_min (col)

- SPARK-23917, SPARK-23918 resolved by Marco Gaido

- Supports any orderable type
  (atomic types, structs, arrays, nulls and UDTs)

```
array_max([2, 5, 3])              →    5
array_min([‘b’, ‘a’, ‘c’])        →    ‘a’
array_max([[5, 2], [6, 1]])       →    [6, 1]
array_min([(5, 2), (6, 1)])       →    [5, 2]
array_min([[], [0, 2]])           →    []
```

- Returns *null* for empty array

```
array_max([])                     →    null
```

array_sort (col)

- SPARK-23921 resolved by Kazuaki Ishizaki

- Sorts arrays in ascending order

array_sort([3, 5, 1, 4, 2])	→	[1, 2, 3, 4, 5]
array_sort([‘c’, ‘a’, ‘b’, ‘e’, ‘d’])	→	[‘a’, ‘b’, ‘c’, ‘d’, ‘e’]

- Nulls handling - *array_sort* **vs.** *sort_array*

array_sort([3, 1, *null*, 2])	→	[1, 2, 3, *null*]
sort_array([3, 1, *null*, 2], true)	→	[*null* , 1, 2, 3]
sort_array([3, 1, *null*, 2], false)	→	[3, 2, 1, *null*]

array_join (col, delimiter[, nullRepl])

- SPARK-23916 resolved by Marco Gaido

- Creates a string from an **array of strings** using a delimiter

  array_join([‘fish’, ‘chips’], ‘&’) → ‘fish&chips’

- Can optionally replace nulls

  array_join([‘a’, null, ‘b’], ‘, ’) → ‘a, b’

  array_join([‘a’, null, ‘b’], ‘, ’, ‘_’) → ‘a, _, b’

arrays_overlap (col1, col2)

- SPARK-23922 resolved by Marco Gaido
- Checks for common elements

  arrays_overlap([1, 2, 3], [4, 2]) → true
  arrays_overlap([1, 2, 3], [4, 5]) → false

- Null elements are ignored

  arrays_overlap([**1**, *null*], [**1**, *null*]) → **true**

  …but not always

  arrays_overlap([**1**, *null*], [**2**, *null*]) → ***null***

array_position (col, elem)
element_at (col, index)

- SPARK-23919, SPARK-23924 resolved by Kazuaki Ishizaki
- 1-based indexing like in SQL
- Find a position of the 1st occurrence

  ```
  array_position([‘a’, ‘b’], ‘a’)     →   1
  array_position([‘a’, ‘b’], ‘x’)     →   0
  ```

- Get an element by an index

  ```
  element_at([‘a’, ‘b’, ‘c’], 2)      →   ‘b’
  element_at([‘a’, ‘b’, ‘c’], 42)     →   null
  ```

- Get a value by a key

  ```
  element_at(map(‘name’ → ‘Bob’, ‘age’ → 42), ‘name’)   →   ‘Bob’
  ```

array_repeat (element, count)

- SPARK-23925 resolved by Florent Pepin
- Repeats anything N times

array_repeat('meh', 3)	→	['meh', 'meh', 'meh']
array_repeat(['doh'], 3)	→	[['doh'], ['doh'], ['doh']]
array_repeat(*null*, 3)	→	[*null*, *null*, *null*]
array_repeat('foo', 0)	→	[]
array_repeat('bar', -1)	→	[]

slice (col, start, length)

- SPARK-23930 resolved by Marco Gaido
- 1-based indexing
- Returns a sub-array

slice([‘a’, **‘b’**, **‘c’**, ‘d’], 2, 2)	→	[‘b’, ‘c’]
slice([‘a’, ‘b’, **‘c’**, **‘d’**], -2, 2)	→	[‘c’, ‘d’]

In Progress

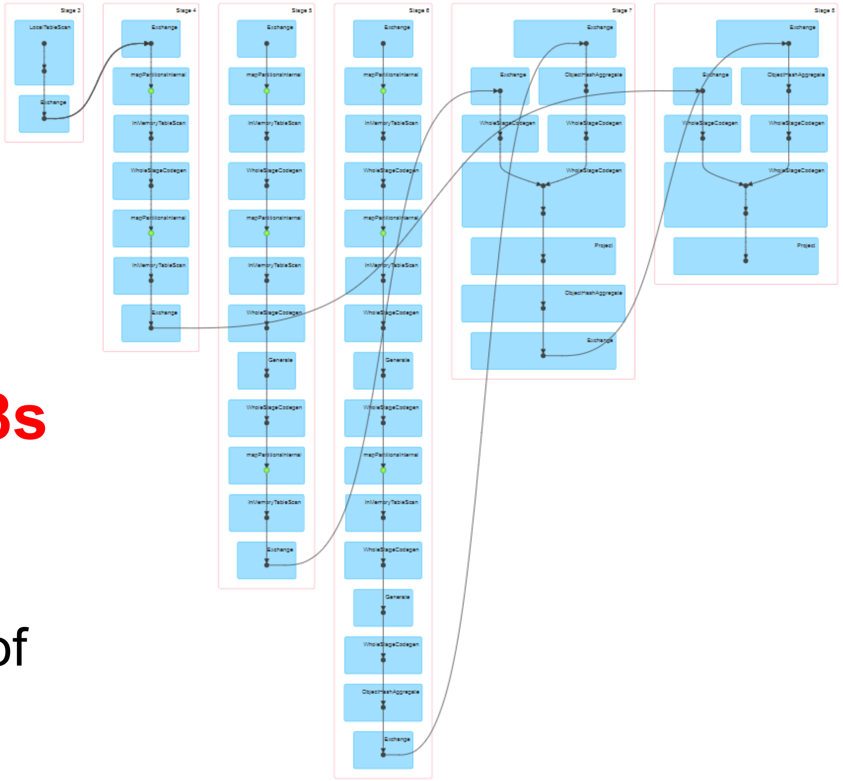
- array_distinct - SPARK-23912
- array_intersect - SPARK-23913
- array_union - SPARK-23914
- array_except - SPARK-23915
- array_remove - SPARK-23920
- shuffle - SPARK-23928
- zip - SPARK-23931
- map_from_arrays - SPARK-23933

Higher-order Functions

- transform - SPARK-23908
- filter - SPARK-23909
- reduce - SPARK-23911
- zip_with - SPARK-23932

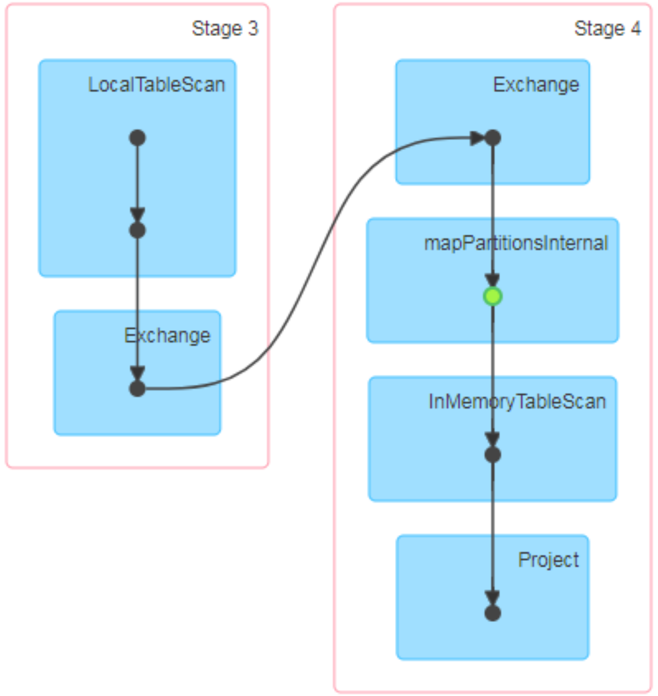
Is it really so crucial?

- 1000 rows with 150x150 matrices represented as array[array[int]]
  - Square its elements
  - **Explosion approach (47.8s on local[4])**
  - This is a small example
    - What happens with millions of records?



…and with our transform

- With our implementation of transform
  - Still lacks wholestage codegen
  - Very simple implementation
  - Same data, same operation
  - **13.7s on local[4]**



Big Thank to Reviewers!

- **Takuya Ueshin** ~ ueshin
- **Kazuaki Ishizaki** ~ kiszk
- **Liang-Chi Hsieh** ~ viirya
- **Takeshi Yamamuro** ~ maropu
- **Hyukjin Kwon** ~ HyukjinKwon
- **Xiao Li** ~ gatorsmile
- **Wenchen Fan** ~ cloud-fan
- **Weichen Xu** ~ WeichenXu123
- **Marco Gaido** ~ mgaido91
- **Reynold Xin** ~ rxin