



Big Data Analytics: Approaches and Tools

Московский городской педагогический университет

Лекция 11-2. Обработка графов Spark GraphX

Основные темы

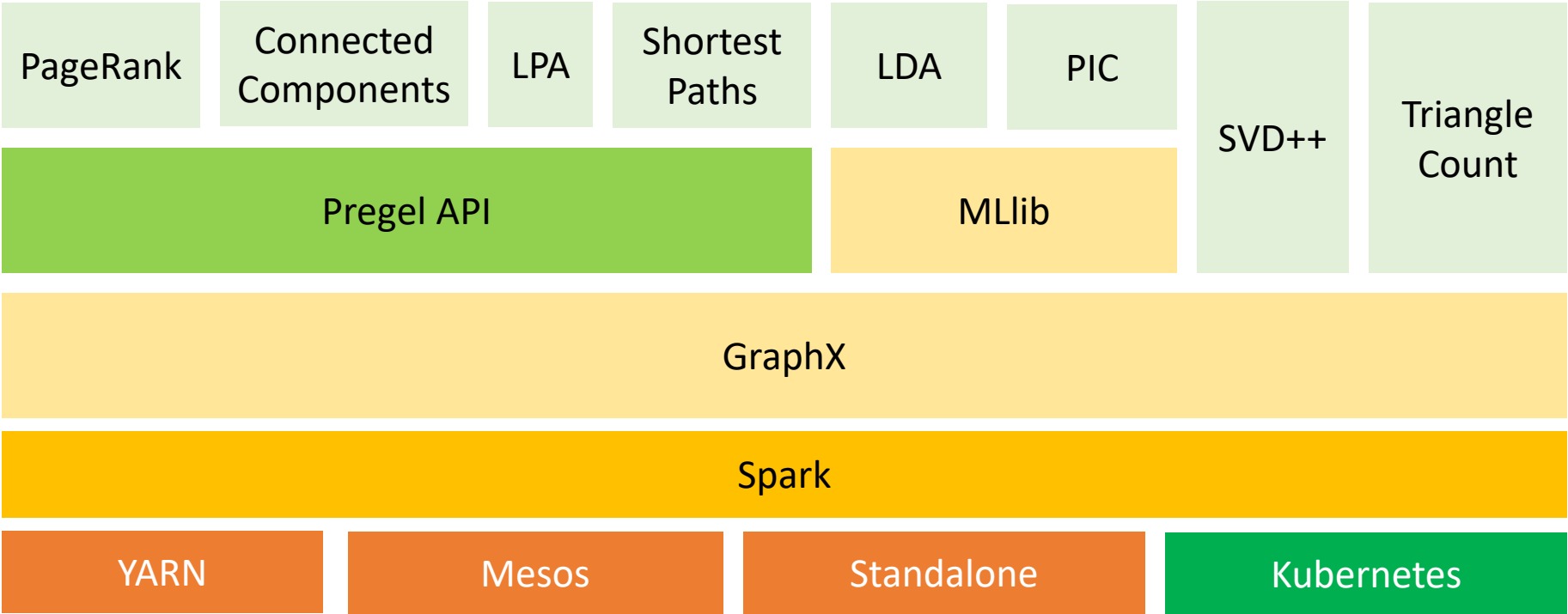
- Назначение и особенности
- Модель данных и вычислений
- Распределенный граф в GraphX
- Операторы
- Сравнение

Spark GraphX

Назначение и особенности GraphX

- Построен на основе Spark RDD
- Граф в GraphX – направленный мульти-граф с определяемыми пользователем свойствами/атрибутами вершин и ребер
- Граф представляется как распределенная неизменяемая структура, состоящая из RDD вершин и ребер
- Предоставляет набор операторов для выполнения операций над графом (subgraph, joinVertices, aggregateMessages, Pregel API)
- Содержит набор высокоуровневых алгоритмов, таких как PageRank, ConnectedComponents и др.

Стек



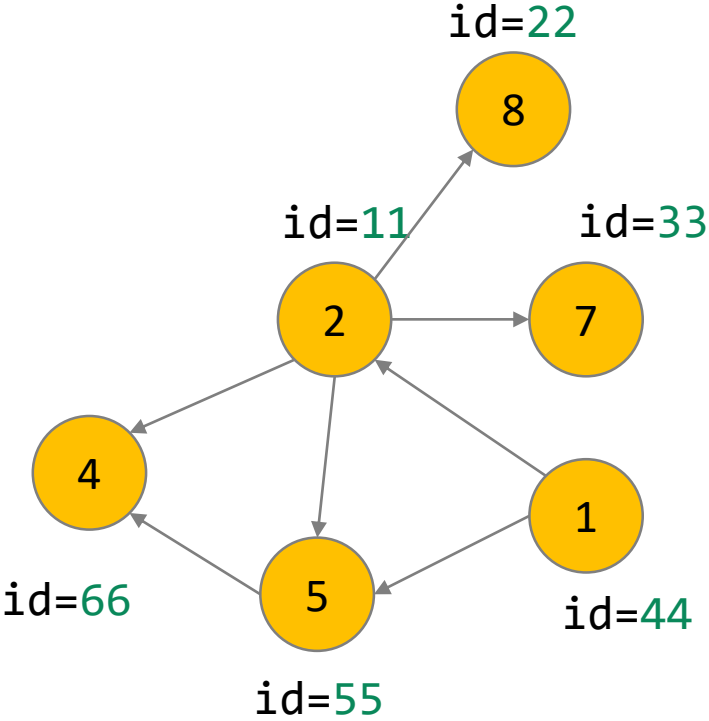


Словарь

- Vertex-cut
- Vertex Partition
- Edge Partition
- Triplet
- Операторы
- Pregel API
- Routing Table
- Bitmask

Модель данных и вычислений

Граф как коллекция данных



Вершины

id	value
11	2
22	8
33	7
44	1
55	5
66	4

Ребра

srcId	value	dstId
11	1	22
11	1	33
11	1	55
11	1	66
44	1	11
44	1	55
55	1	66

Стадия 1. Построение triplets

vertices

id	value
11	2
22	8
33	7
44	1
55	5
66	4

```
CREATE VIEW triplets AS
SELECT e.srcId, e.dstId, e.value, s.value, d.value
FROM edges AS e
JOIN vertices AS s JOIN vertices AS d
WHERE e.srcId=s.id AND e.dstId=d.id
```

edges

srcId	value	dstId
11	1	22
11	1	33
11	1	55
11	1	66
44	1	11
44	1	55
55	1	66

triplets

srcId	dstId	edgeValue	srcValue	dstValue
11	22	1	2	8
11	33	1	2	7
11	55	1	2	5
11	66	1	2	4
44	11	1	1	2
44	55	1	1	5
55	66	1	5	4

Стадия 2. Агрегирование

triplets

srcId	dstId	edgeValue	srcValue	dstValue
11	22	1	2	8
11	33	1	2	7
11	55	1	2	5
11	66	1	2	4
44	11	1	1	2
44	55	1	1	5
55	66	1	5	4

```
SELECT t.dst, reduceUDF(mapUDF)  
FROM triplets AS t  
GROUP BY t.dstId
```

Пример для PageRank

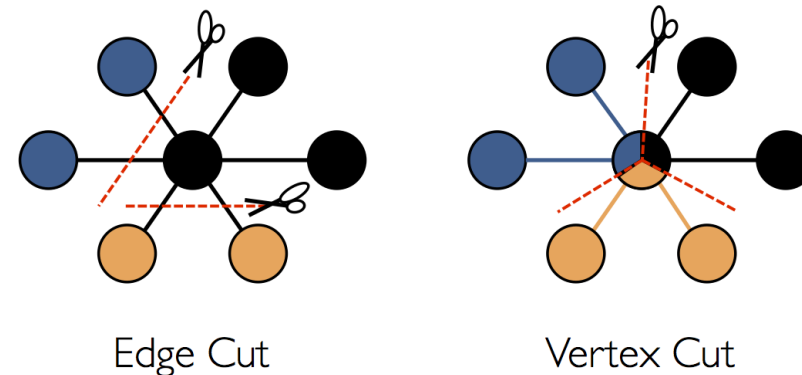
```
mapF => t.srcValue*t.edgeValue  
reduceF => 0.15 + 0.85*sum(map)
```

```
SELECT t.dst, 0.15+0.85*sum(t.srcValue*t.edgeValue)  
FROM triplets AS t  
GROUP BY t.dstId
```

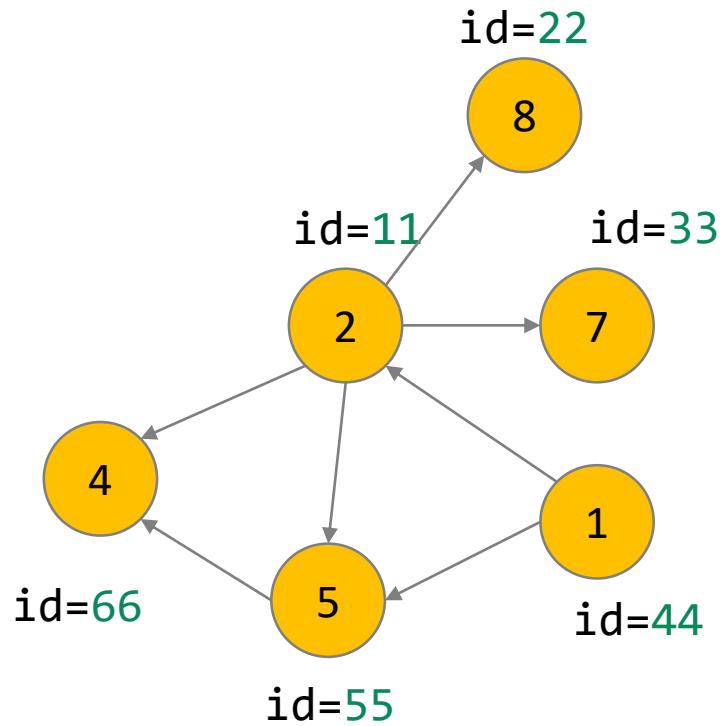
Распределенный граф в GraphX

Особенности Vertex-cut

- В отличие от **edge-cut**, когда вершины равномерно распределяются по машинам, при **vertex-cut** равномерно распределяются ребра, таким образом вершины могут охватывать несколько машин
- Графы в реальном мире имеют больше ребер чем вершин, поэтому вершины доставляются в место хранения/обработки ребер
- Уменьшая количество машин, охватываемых каждой вершиной, уменьшается коммуникация и общий объем памяти для хранения



Распределенное представление графа в GraphX



edges

Partition A

srcId	value	dstId
11	1	22
11	1	33

Partition B

srcId	value	dstId
44	1	11
44	1	55

Partition C

srcId	value	dstId
11	1	55
11	1	66
55	1	66

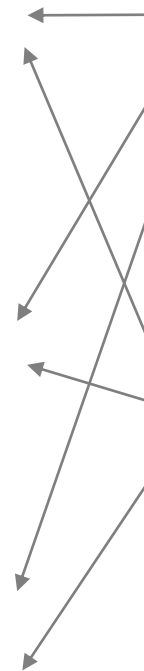
vertices

Partition I

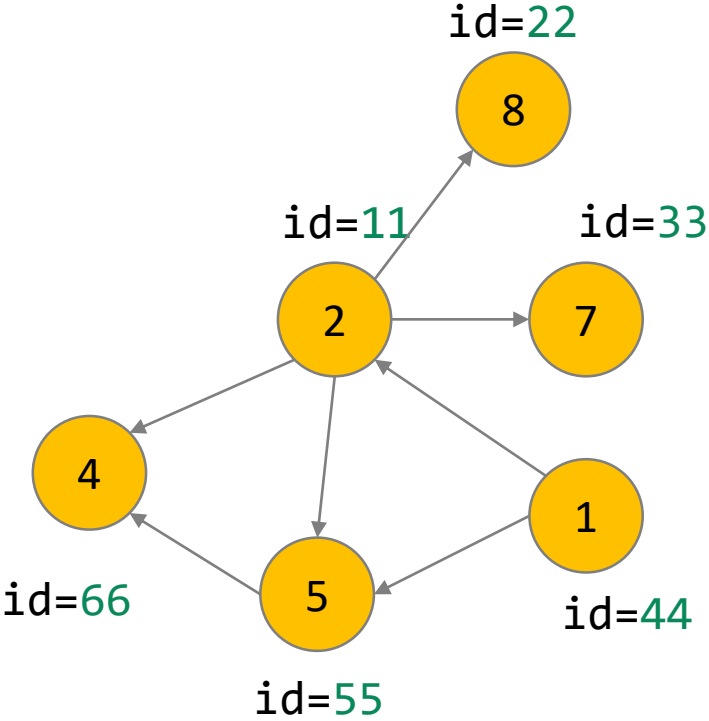
id	value
11	2
22	8
33	7

Partition II

id	value
44	1
55	5
66	4



Routing Table



edges

Partition A

srcId	value	dstId
11	1	22
11	1	33



Partition B

srcId	value	dstId
44	1	11
44	1	55



Partition C

srcId	value	dstId
11	1	55
11	1	66
55	1	66



vertices

Partition I

id	value
11	2
22	8
33	7

routing table

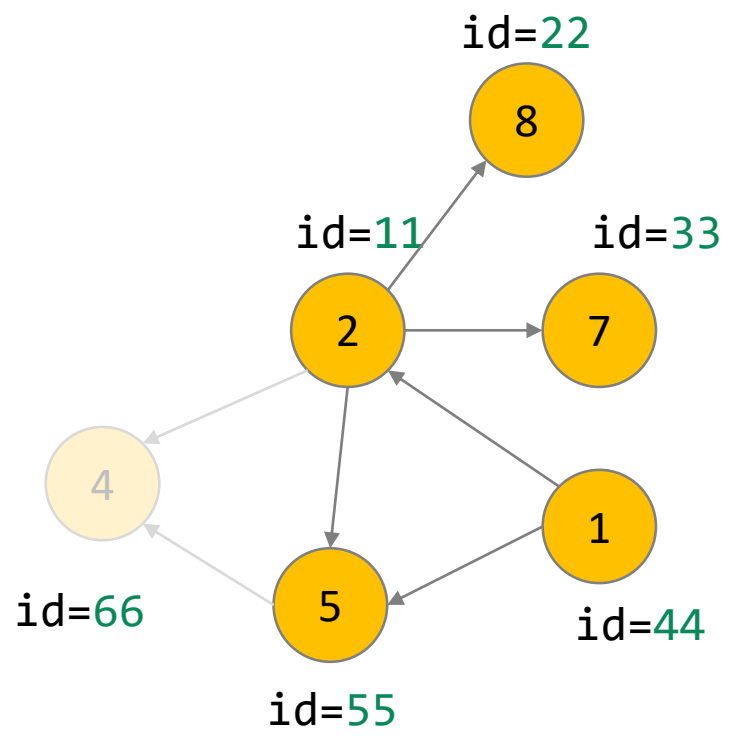
partId	vertices
A	11,22,33
B	11
C	11

Partition II

id	value
44	1
55	5
66	4

partId	vertices
A	
B	44,55
C	55,66

Bitmask



edges

Partition A

srcId	value	dstId
11	1	22
11	1	33

Partition B

srcId	value	dstId
44	1	11
44	1	55

Partition C

srcId	value	dstId
11	1	55
11	1	66
55	1	66

vertices

Partition I

id	value
11	2
22	8
33	7

vertex bitmask

bitmask
1
1
1

routing table

partId	vertices
A	11,22,33
B	11
C	11

Partition II

id	value
44	1
55	5
66	4

bitmask
1
1
0

partId	vertices
A	
B	44,55
C	55,66

Особенности вершин в GraphX

- **Routing table** используется для копирования туда, где расположены части (partitions) **edges**, только необходимых вершин, которые соответствуют src/dst в данной части. Это необходимо для ускорения выполнения **join** операций при формировании **triplets**
- **Bitmask** используется для сокрытия/исключения вершин при выполнении операций
- Вершины хранятся на стороне **частей edges** и обновляются при изменении значений соответствующих вершин в **vertices**

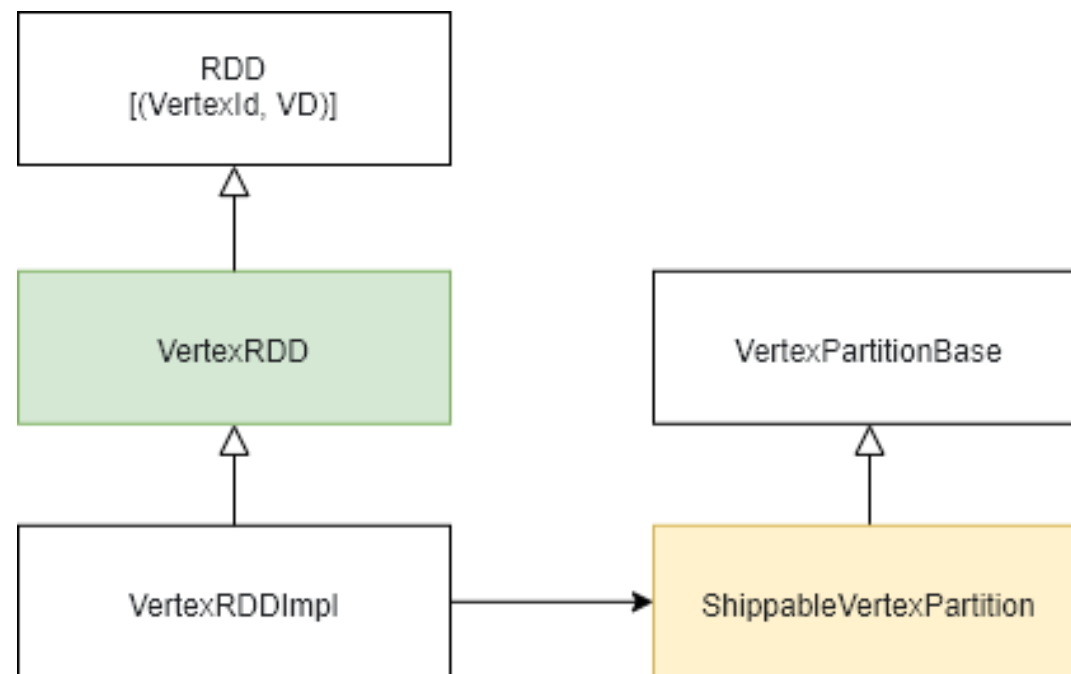
Класс VertexRDD

VertexRDD – абстрактный класс для представления вершин:

- Вершина представляется в виде VertexId (Long) и атрибута типа VD
- Набор вершин делится на части (partitions) по VertexId
- Расширяет/наследует RDD[(VertexId, VD)]

ShippableVertexPartition

- **index** : VertexIdToIndexMap – **индекс** для определения позиции вершины по её **id**
- **value** : Array[VD] – массив значений вершины (позиция вершины определяется по index)
- **mask** : BitSet – множество активных вершин
- **routingTable** : RoutingTablePartition



Класс EdgeRDD

EdgeRDD – абстрактный класс для представления ребер:

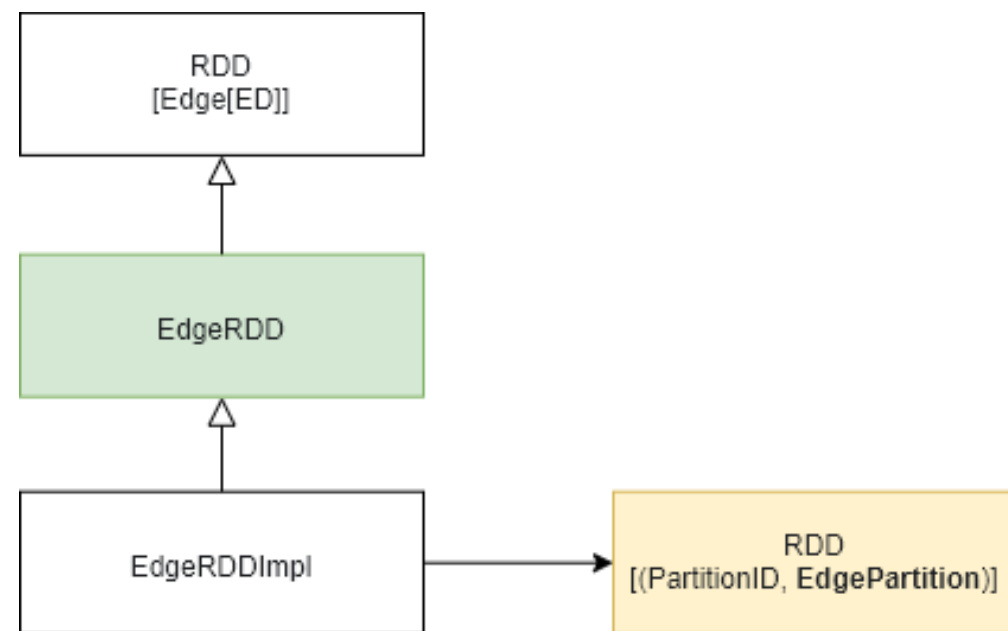
- Ребро представляется в виде экземпляра **Edge** класса
- Набор ребер делится на части (partitions) в соответствии с **PartitionStrategy**
- Расширяет/наследует **RDD[Edge[ED]]**

Edge:

- srcId : VertexID
- dstId : VertexID
- attr : ED

PartitionStrategy:

- CanonicalRandomVertexCut
- EdgePartition1D
- EdgePartition2D
- RandomVertexCut



EdgePartition:

- localSrcIds : Array[Int] – локальный id вершины источника
- localDstIds : Array[Int] – локальный id вершины назначения
- data : Array[ED] – значение ребра
- global2local : ~HashMap[VertexID, Int] – перевод глобального id вершины в локальный id
- local2global : Array[VertexID] – перевод локального id вершины в глобальный id
- vertexAttrs : Array[VD] – значения вершин
- activeSet : VertexSet – множество активных вершин

index		Partition A			local2global		attributes		global2local	
srcId	offset	localSrcId	data	localDstId	localId	globalId	localId	value	globalId	localId
11	0	0	1	1	0	11	0	1	11	0
77	2	0	1	2	1	22	1	1	22	1
		3	1	1	2	33	2	1	33	2
					3	77	3	1	77	3

Класс Graph

Коллекции графа

```
val vertices: VertexRDD[VD]  
val edges: EdgeRDD[ED]  
val triplets: RDD[EdgeTriplet[VD, ED]]
```

Информация о графе

```
val numEdges: Long  
val numVertices: Long  
val inDegrees: VertexRDD[Int]  
val outDegrees: VertexRDD[Int]  
val degrees: VertexRDD[Int]
```

Структурные операторы (изменяют структуру)

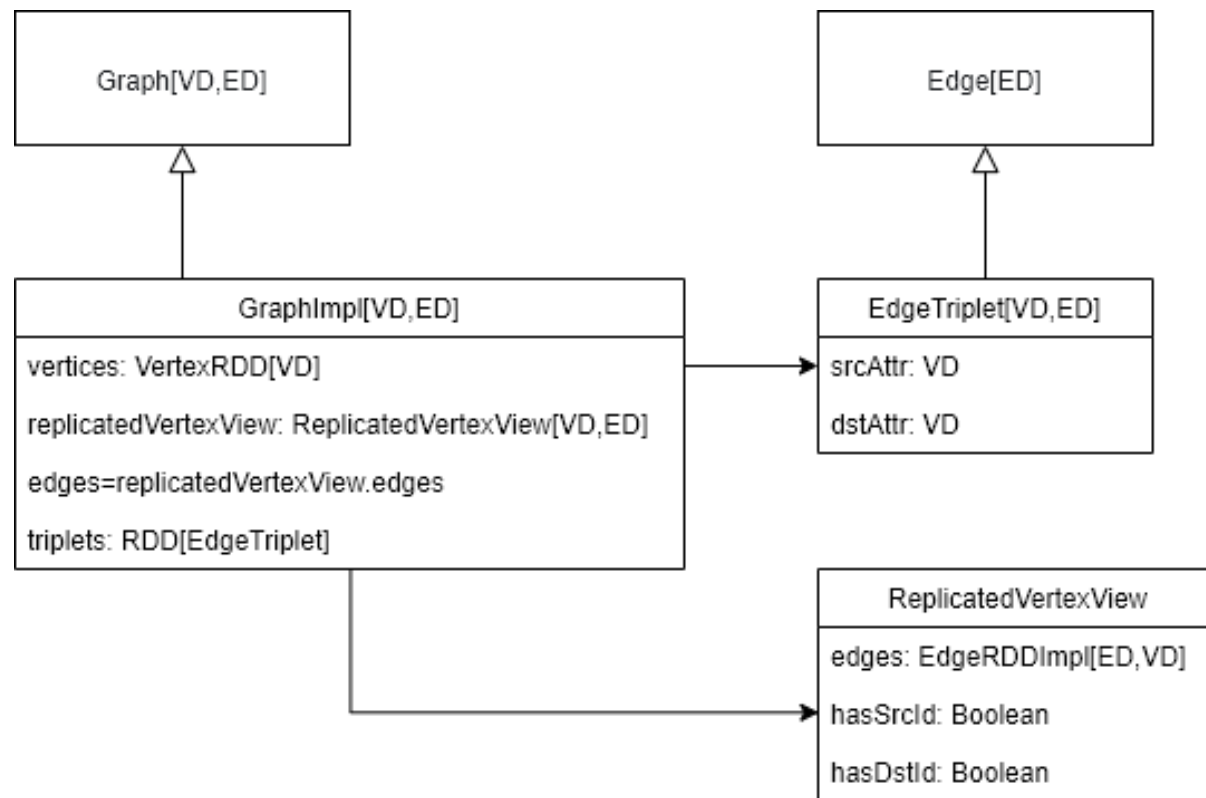
```
def reverse  
def subgraph  
def mask[VD2, ED2]  
def groupEdges
```

Операторы свойств

```
def mapVertices[VD2]  
def mapEdges[ED2]  
def mapTriplets[ED2]
```

Другие операторы

```
def joinVertices[U]  
def aggregateMessages[Msg: ClassTag]  
def pregel[A]
```



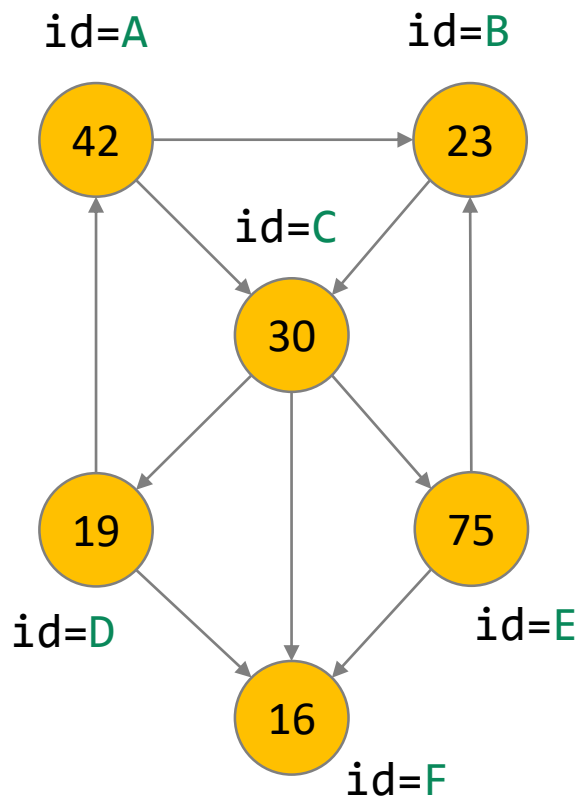
Операторы

Оператор **mapReduceTriplets** берет определенную пользователем функцию **map** и применяет её к каждому triplet'у и получает в результате агрегированные сообщения с использованием функции **reduce**

```
class Graph[VD, ED] {  
  def mapReduceTriplets[Msg](  
    map: EdgeTriplet[VD, ED] => Iterator[(VertexId, Msg)],  
    reduce: (Msg, Msg) => Msg)  
    : VertexRDD[Msg]  
}
```

- Данный оператор устарел
- Используемый в **mapReduceTriplets** итератор ограничивает возможности по оптимизации
- Вместо него используется **aggregateMessages**

Пример mapReduceTriplets



```
SELECT t.dst, reduceUDF(mapUDF)
FROM triplets AS t
GROUP BY t.dstId
```

```
def mapUDF(t: EdgeTriplet[User, Double])
  : Iterator[(VertexId, Int)] =
  if (t.src.age > t.dst.age) (t.dstId, 1)
  else (t.dstId, 0)

def reduceUDF(a: Int, b: Int): Int = a + b

val seniors: VertexRDD[(VertexId, Int)] =
  graph.mapReduceTriplets(mapUDF, reduceUDF)
```

Результат

id	value
A	0
B	2
C	1
D	1
E	0
F	3

aggregateMessages

Оператор **aggregateMessages** применяет функцию **sendMsg** к каждому ребру triplet'а в графе и затем использует функцию **mergeMsg** для агрегирования сообщений на вершинах назначения.

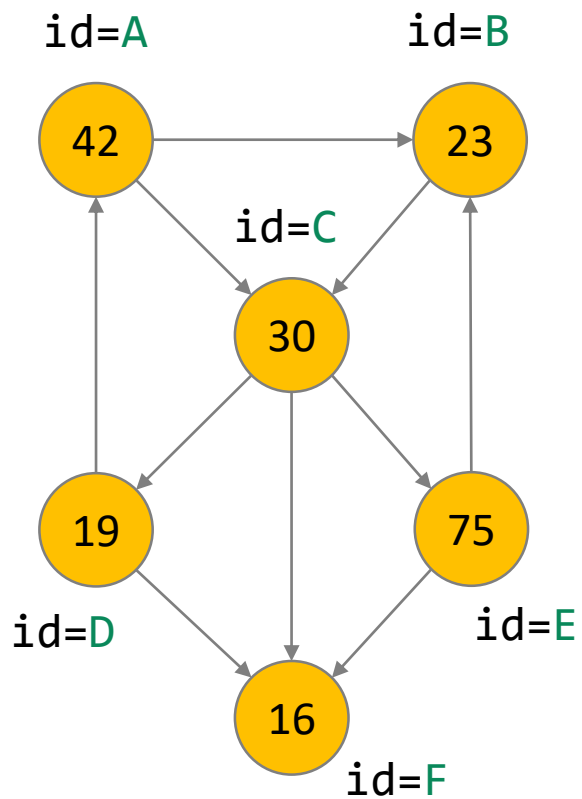
```
class Graph[VD, ED] {  
  def aggregateMessages[Msg: ClassTag](  
    sendMsg: EdgeContext[VD, ED, Msg] => Unit,  
    mergeMsg: (Msg, Msg) => Msg,  
    tripletFields: TripletFields = TripletFields.All)  
    : VertexRDD[Msg]  
  
  def aggregateMessagesWithActiveSet[Msg: ClassTag](  
    sendMsg: EdgeContext[VD, ED, A] => Unit,  
    mergeMsg: (Msg, Msg) => Msg,  
    tripletFields: TripletFields,  
    activeSetOpt: Option[(VertexRDD[_], EdgeDirection)])  
    : VertexRDD[Msg]  
}
```

- None (без значений)
- EdgeOnly (значение ребер)
- Src (значение исходящих вершин)
- Dst (значение входящих вершин)
- All (все значения)

Два вариант доступа к значениям в **edge** частях:

- Полный перебор по ребрам (**edge scan**)
- Использование индекса (**index scan**) (если доля активных вершин < 0.8)

Пример aggregateMessages



```
SELECT t.dst, reduceUDF(mapUDF)
FROM triplets AS t
GROUP BY t.dstId
```

```
val seniors: VertexRDD[(VertexId, Double)] =
  graph.aggregateMessages[(VertexId, Double)](

    t => { // Map Function
      if (t.srcAttr > t.dstAttr) {
        // Send message to destination vertex
        t.sendToDst((1, t.srcAttr))
      }
    },

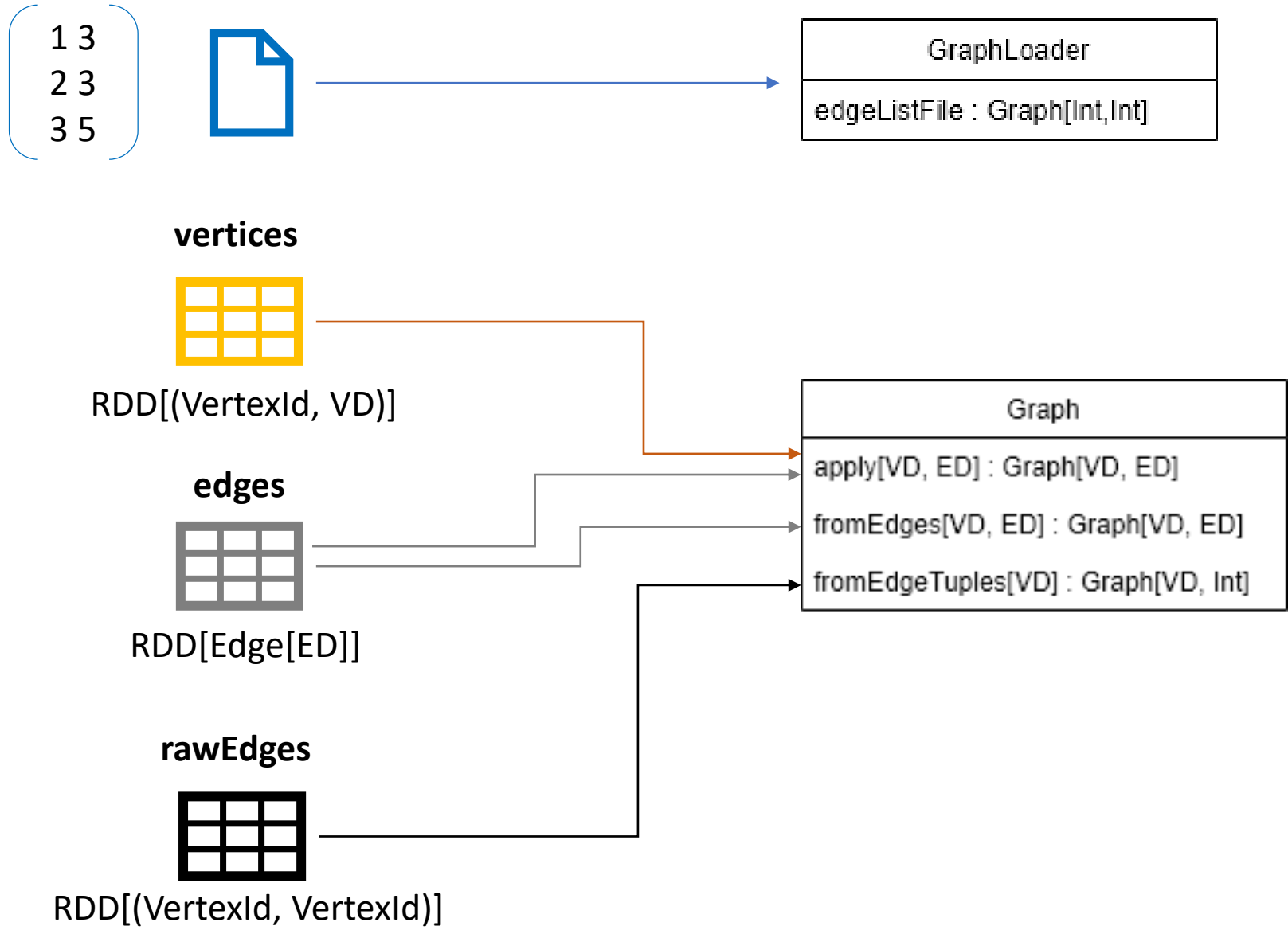
    // Add counter and age
    (a, b) => (a._1 + b._1, a._2 + b._2) // Reduce Function
  )
```

Результат

id	value
A	0
B	2
C	1
D	1
E	0
F	3

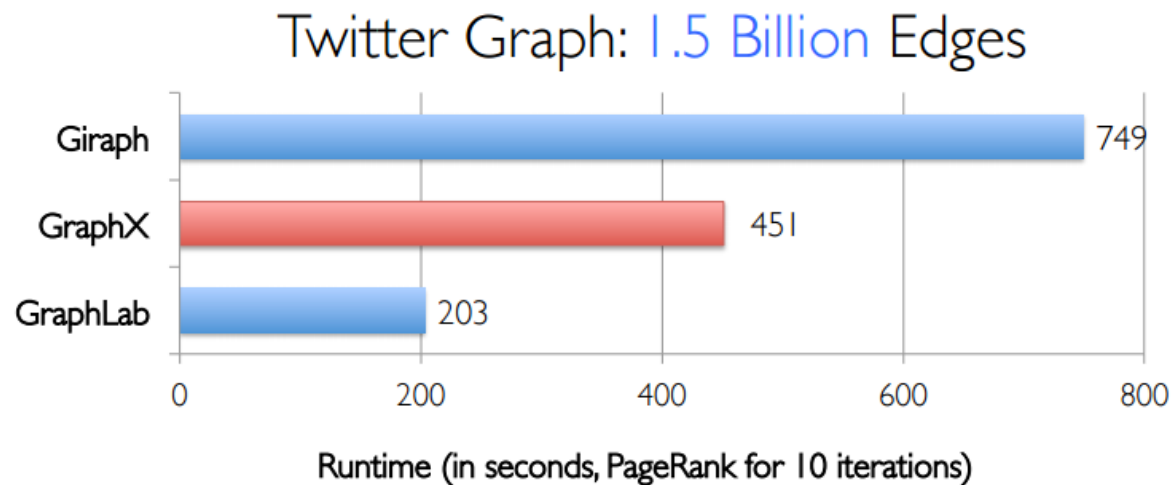
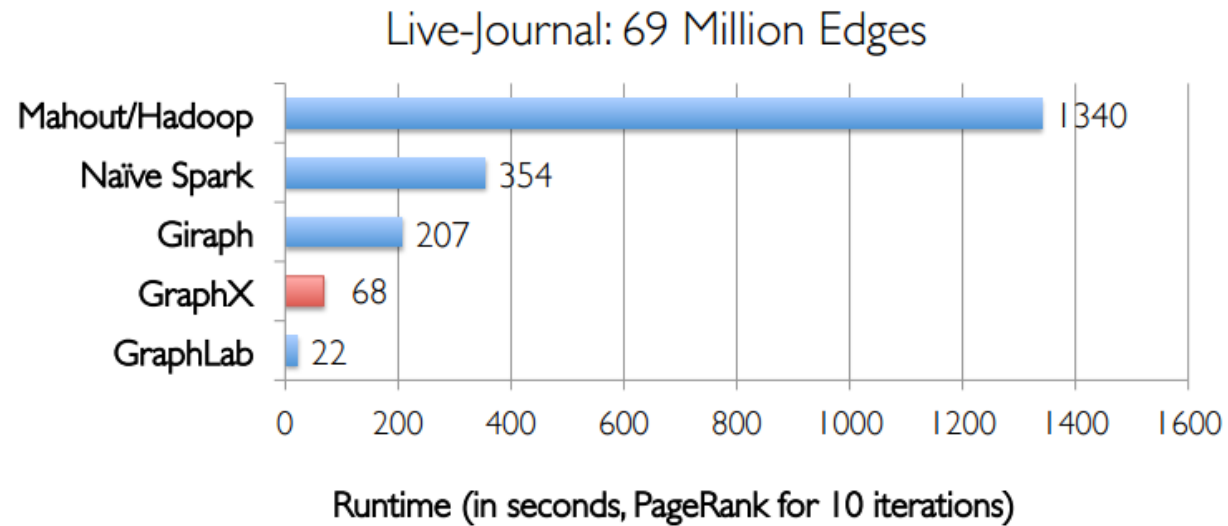
```
def pregel(g: Graph[V,E],
  vprog: (VertexId, VD, A) => VD,
  sendMsg: EdgeTriplet[VD, ED] => Iterator[(VertexId, A)],
  mergeMsg: (A, A) => A) : Graph[VD, ED] {
  // Установить все вершины как активные
  g = g.mapVertices({id, v} => (v, halt=false))
  // Повторить пока есть активные вершины
  while (g.vertices.exists(v => !v.halt)) {
    // Вычислить сообщения
    val msgs = g.subgraph((s,d,sP,eP,dP)=>!sP.halt)
                      .mapReduceTriplets(sendMsg, mergeMsg)
    // Получить сообщения и выполнить vprog
    g = g.leftJoinVertices(msgs).mapVertices(vprog)
  }
  return g
}
```

Создание исходного графа



Сравнение

Сравнение GraphX с другими системами



Источники

[GraphX](#) (github)

[GraphX Programming Guide](#) (official website)

[GraphX: Graph Processing in a Distributed Dataflow Framework](#) (article)