Practical problems in distributed dataflow frameworks

Lijie Xu

2015-03-12

Outline

- Inconsistent results of user programs
  - *Shared variables*
  - *Shared intermediate data*
  - *Transformation error*
- Fault-tolerant problems in iterative applications
  - *Fault tolerance vs. Performance*
- Consistency model of Parameter Server (optional)

Topic 1

- Inconsistent results of user programs
  - *Shared variables*
  - *Shared intermediate data*
  - *Transformation error*

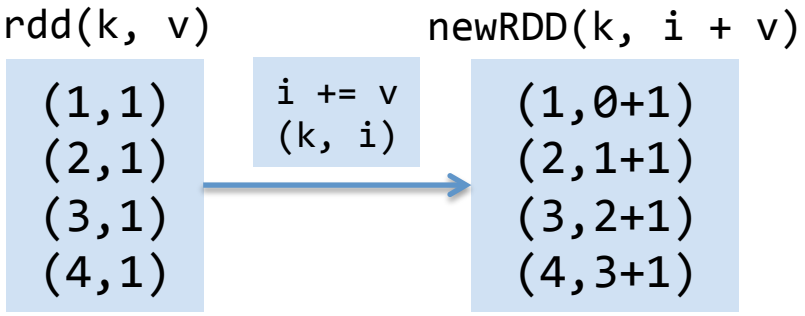
Case 1.1: Shared variables

```scala
object SimpleSharedVariables {
  def main(args: Array[String]) {
    val spConf = new SparkConf().setMaster("local[2]")
    val sc = new SparkContext(spConf)

    var i = 0
    val rdd = sc.makeRDD(List((1, 1), (2, 1), (3, 1), (4, 1)))

    val newRDD = rdd.map{case (k, v) =>
      i += v
      (k, i)
    }

    newRDD.foreach(println)
    println("i = " + i)
  }
}
```

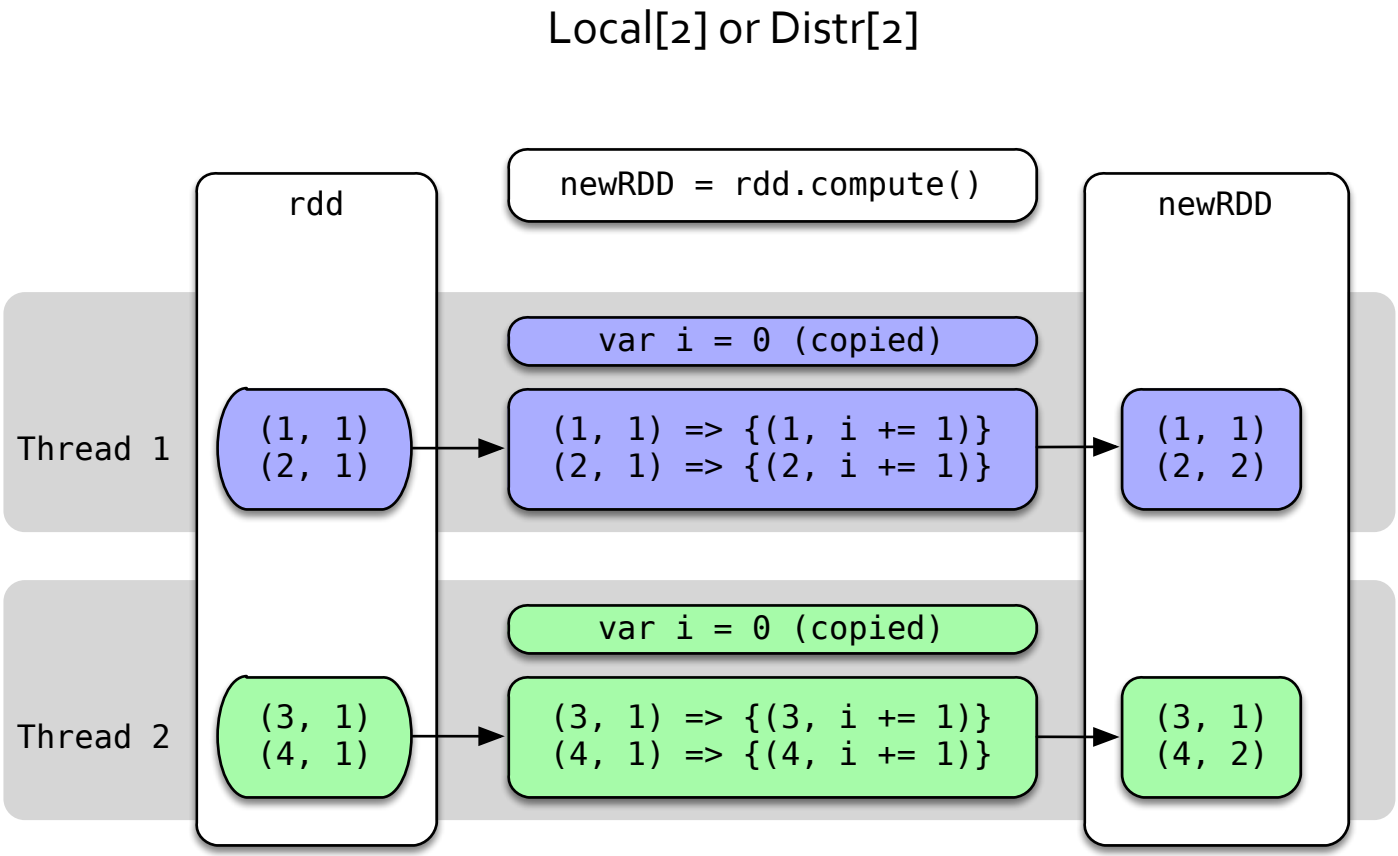
Outputs:

Local[1] Distr[1]	Local[2] Distr[2]
(1,1) (2,2) (3,3) (4,4) i = 0	(1,1) (2,2) (3,1) (4,2) i = 0

Local[4] Distr[4]
(1,1) (2,1) (3,1) (4,1) i = 0



Case 1.1: Shared variables



Case 1.2: Shared variables (array)

```
object ReCalculationModified {

  def main(args: Array[String]) {
    val spConf = new SparkConf().setMaster("local")
    val sc = new SparkContext(spConf)

    val array = Array(0, 0, 0, 0)

    val vertexRDD = sc.makeRDD(List(0, 1, 2, 3))

    val newVertexRDD = vertexRDD.map(vid => {array(vid) += 1; (vid, array)})

    newVertexRDD.foreach(x => println(x._1 + ":" + x._2.toBuffer))
    println("Array is: " + array.toBuffer)
  }
}
```

Local/Distr[1]

0:ArrayBuffer(1, 0, 0, 0)
1:ArrayBuffer(1, 1, 0, 0)
2:ArrayBuffer(1, 1, 1, 0)
3:ArrayBuffer(1, 1, 1, 1)

collect() [1]

0:ArrayBuffer(1, 1, 1, 1)
1:ArrayBuffer(1, 1, 1, 1)
2:ArrayBuffer(1, 1, 1, 1)
3:ArrayBuffer(1, 1, 1, 1)

Local/Distr[2]

0:ArrayBuffer(1, 0, 0, 0)
1:ArrayBuffer(1, 1, 0, 0)
2:ArrayBuffer(0, 0, 1, 0)
3:ArrayBuffer(0, 0, 1, 1)

collect() [2]

0:ArrayBuffer(1, 1, 0, 0)
1:ArrayBuffer(1, 1, 0, 0)
2:ArrayBuffer(0, 0, 1, 1)
3:ArrayBuffer(0, 0, 1, 1)

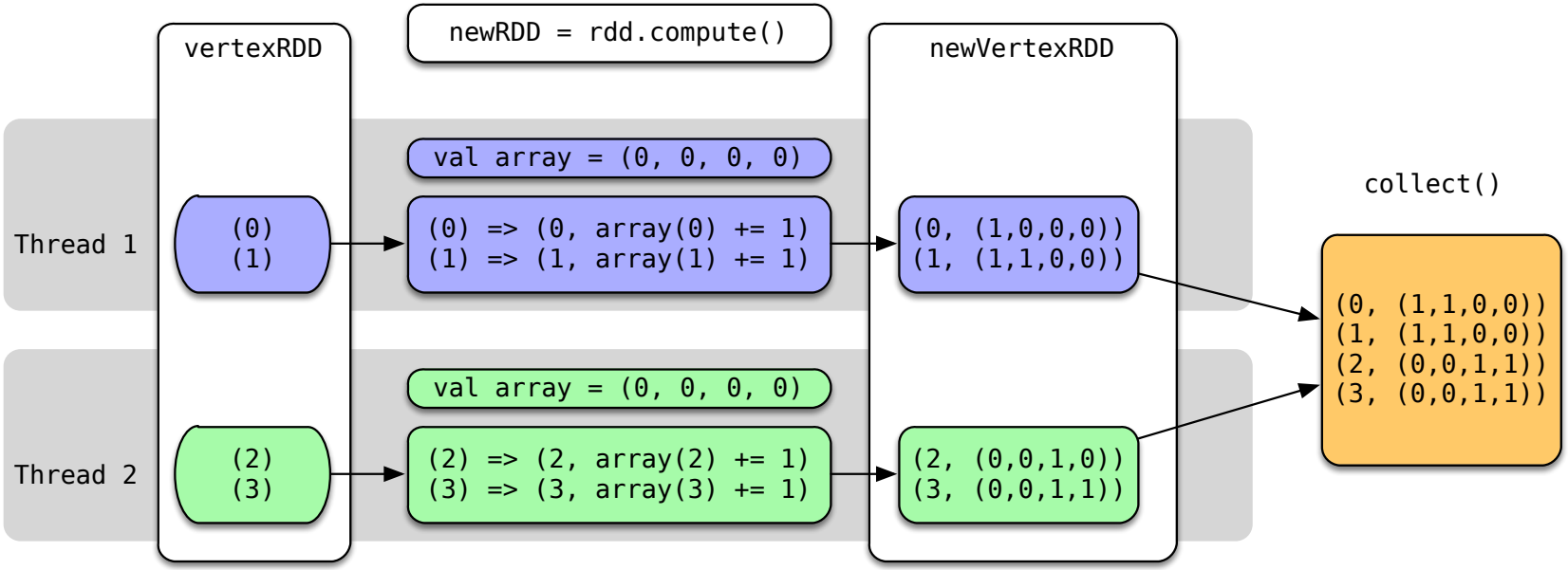
Local/Distr[4]

0:ArrayBuffer(1, 0, 0, 0)
1:ArrayBuffer(0, 1, 0, 0)
2:ArrayBuffer(0, 0, 1, 0)
3:ArrayBuffer(0, 0, 0, 1)

collect() [4]

0:ArrayBuffer(1, 0, 0, 0)
1:ArrayBuffer(0, 1, 0, 0)
2:ArrayBuffer(0, 0, 1, 0)
3:ArrayBuffer(0, 0, 0, 1)

Case 1.2: Shared variables (array)



Local/Distr[2]

0:ArrayBuffer(1, 0, 0, 0)
1:ArrayBuffer(1, 1, 0, 0)
2:ArrayBuffer(0, 0, 1, 0)
3:ArrayBuffer(0, 0, 1, 1)

collect() [2]

0:ArrayBuffer(1, 1, 0, 0)
1:ArrayBuffer(1, 1, 0, 0)
2:ArrayBuffer(0, 0, 1, 1)
3:ArrayBuffer(0, 0, 1, 1)

Case 2: Shared intermediate data

```
// graph
val vertex = Array(
  (0, Array(0, 0, 0, 0)),
  (1, Array(0, 0, 0, 0)),
  (2, Array(0, 0, 0, 0)),
  (3, Array(0, 0, 0, 0)))

val vertexRDD = sc.makeRDD(vertex).map(x => x)

val newVert1 = vertexRDD.map{case (vid, array) =>
  array(vid) += 1
  (vid, array)
}

newVert1.foreach(x => println(x._1 + ":" + x._2.toBuffer))
vertexRDD.foreach(x => println(x._1 + ":" + x._2.toBuffer))

val newVert2 = vertexRDD.map{case (vid, array) =>
  array(vid) += 1
  (vid, array)
}

newVert2.foreach(x => println(x._1 + ":" + x._2.toBuffer))
vertexRDD.foreach(x => println(x._1 + ":" + x._2.toBuffer))
```

newVert1:

0:ArrayBuffer(1, 0, 0, 0)
1:ArrayBuffer(0, 1, 0, 0)
2:ArrayBuffer(0, 0, 1, 0)
3:ArrayBuffer(0, 0, 0, 1)

newVert2:

0:ArrayBuffer(1, 0, 0, 0)
1:ArrayBuffer(0, 1, 0, 0)
2:ArrayBuffer(0, 0, 1, 0)
3:ArrayBuffer(0, 0, 0, 1)

Case 2: Shared intermediate data

```
// graph
val vertex = Array(
  (0, Array(0, 0, 0, 0)),
  (1, Array(0, 0, 0, 0)),
  (2, Array(0, 0, 0, 0)),
  (3, Array(0, 0, 0, 0)))

val vertexRDD = sc.makeRDD(vertex).map(x => x).cache()

val newVert1 = vertexRDD.map{case (vid, array) =>
  array(vid) += 1
  (vid, array)
}

newVert1.foreach(x => println(x._1 + ":" + x._2.toBuffer))
vertexRDD.foreach(x => println(x._1 + ":" + x._2.toBuffer))

val newVert2 = vertexRDD.map{case (vid, array) =>
  array(vid) += 1
  (vid, array)
}

newVert2.foreach(x => println(x._1 + ":" + x._2.toBuffer))
vertexRDD.foreach(x => println(x._1 + ":" + x._2.toBuffer))
```

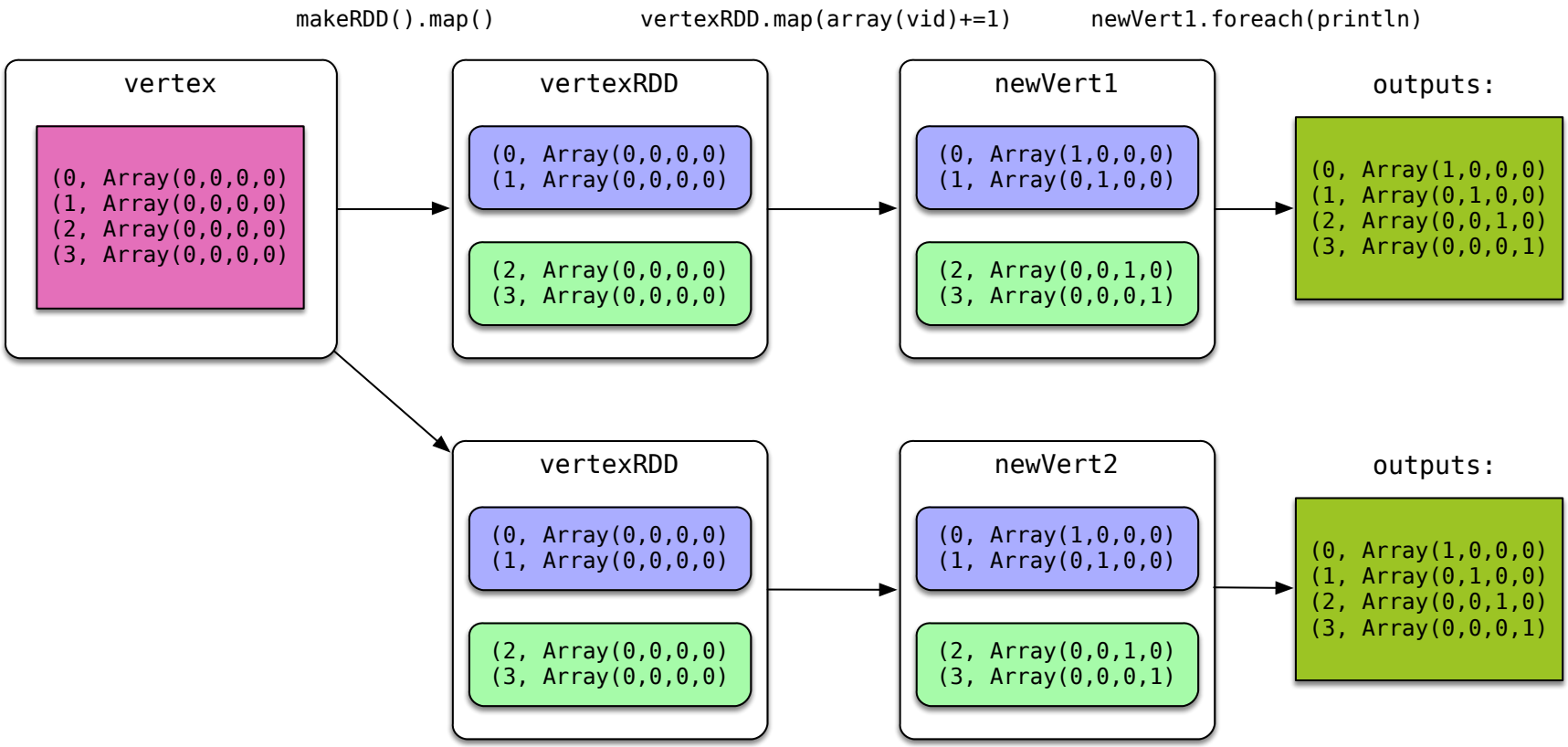
newVert1:

0:ArrayBuffer(1, 0, 0, 0)
1:ArrayBuffer(0, 1, 0, 0)
2:ArrayBuffer(0, 0, 1, 0)
3:ArrayBuffer(0, 0, 0, 1)

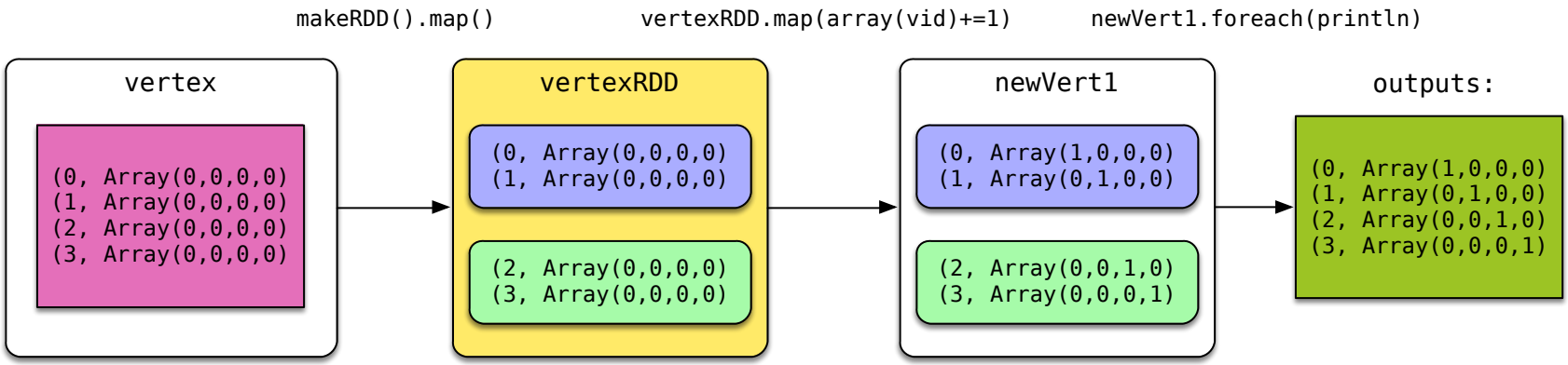
newVert2:

0:ArrayBuffer(2, 0, 0, 0)
1:ArrayBuffer(0, 2, 0, 0)
2:ArrayBuffer(0, 0, 2, 0)
3:ArrayBuffer(0, 0, 0, 2)

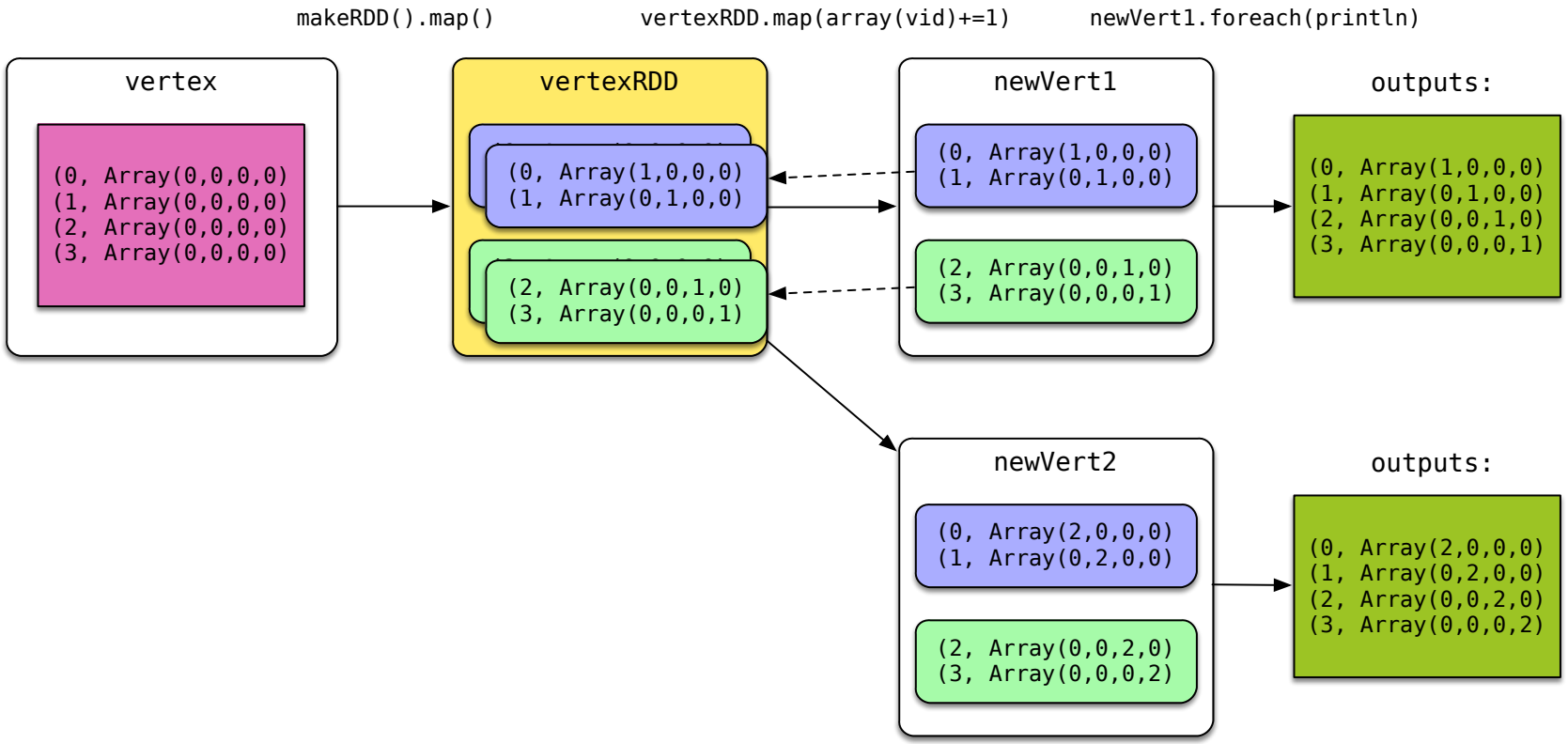
Case 2: Shared intermediate data without cache()



Case 2: Shared intermediate data with cache()



Case 2: Shared intermediate data with cache()



Case 3: Transformation error

```scala
object ItersectionList {

  def main(args: Array[String]) {
    val spConf = new SparkConf().setMaster("local[2]")
    val sc = new SparkContext(spConf)

    val a = Array(
      (1, List(1596, 1617, 1929, 2399, 2674)),
      (2, List(1702, 1785, 1933, 2054, 2583, 2913)),
      (3, List(1982, 2002, 2048, 2341, 2666))
    )

    val t2 = List(2002, 2399)

    val t1 = sc.makeRDD(a).map(x => (x._1, (List(x._2)))).reduceByKey(_ ++ _)
    val t3intersect = t1.map(x => (x._1, (x._2.intersect(t2))))
    val t3union = t1.map(x => (x._1, (x._2.union(t2))))

    t3intersect.foreach(println)
    t3union.foreach(println)
  }
}
```

```
Intersection:
(2,List())
(1,List())
(3,List())

Union:
(2,List(List(1702, 1785, 1933, 2054, 2583, 2913), 2002, 2399))
(1,List(List(1596, 1617, 1929, 2399, 2674), 2002, 2399))
(3,List(List(1982, 2002, 2048, 2341, 2666), 2002, 2399))
```

This should return intersection:

```
(1, List(2399))
(2, List())
(3, List(2002))
```

and union:

```
(1, List(1596, 1617, 1929, 2399, 2674, 2002))
(2, List(1702, 1785, 1933, 2054, 2583, 2913, 2002, 2399))
(3, List(1982, 2002, 2048, 2341, 2666, 2399))
```

http://stackoverflow.com/questions/29000384/spark-intersection-of-lists-not-working/

Requirements of user code

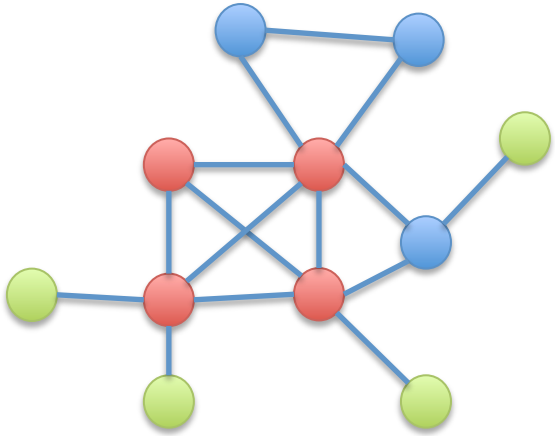
- Deterministic
  - *Asynchronous algorithms (e.g., delta PageRank, asynchronous MapReduce)*
  - *Branch-and-bound algorithms (NP-hard, update shared bounds)*
  - *Adaptive algorithms*
  - *Non-deterministic parallelism considered useful, HotOS 2011*
- Idempotent
  - $a * a = a$
- Commutative
  - $a + b = b + a$
  - *Nondeterminism in MapReduce Considered Harmful? An Empirical Study on Non-commutative Aggregators in MapReduce Programs, ICSE 2014*
- Associative
  - $(a + b) + c = a + (b + c)$
  - *Mean(20, 10, 25, 15) vs. Mean(mean(20, 10), mean(25, 15))*

Topic 2

- Fault-tolerant problems in iterative applications
  - *Fault tolerance vs. Performance*

A StackOverflow error in K-Core

```
//K-Core Algorithm
val kNum = 5

var degreeGraph = graph.outerJoinVertices(graph.degrees) {
  (vid, vd, degree) => degree.getOrElse(0)
}.cache()

do {
  val subGraph = degreeGraph.subgraph(
    vpred = (vid, degree) => degree >= KNum
  ).cache()

  val newDegreeGraph = subGraph.degrees

  degreeGraph = subGraph.outerJoinVertices(newDegreeGraph) {
    (vid, vd, degree) => degree.getOrElse(0)
  }.cache()

  isConverged = check(degreeGraph)
} while(isConverged == false)
```



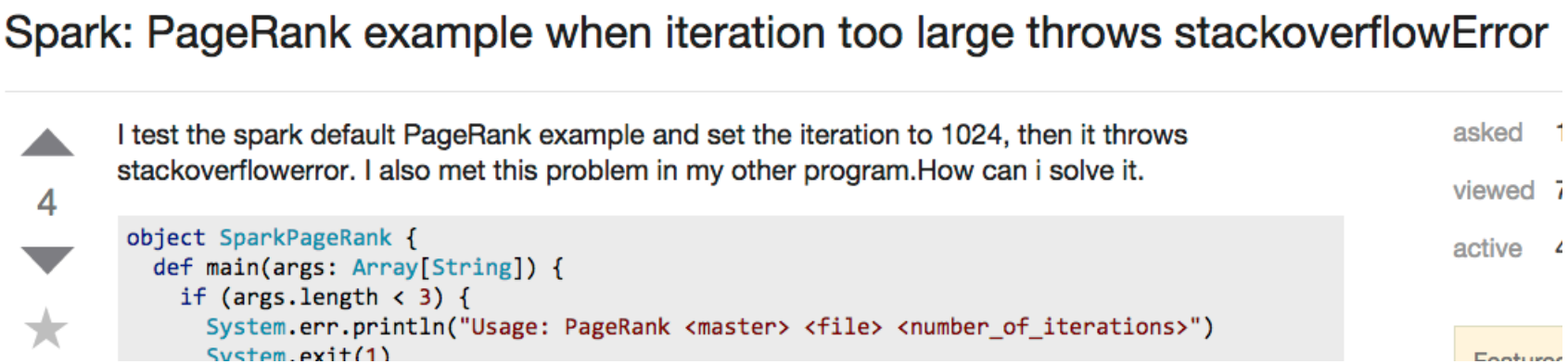
A k-core of G can be obtained by recursively removing all the vertices of degree less than k, until all vertices in the remaining graph have degree at least k.

StackOverflow error

- After 300+ iterations

```
ERROR Executor: Exception in task 1.0 in stage 339993.0 (TID 3341)
java.lang.StackOverflowError

at java.lang.StringBuilder.append(StringBuilder.java:204)
at java.io.ObjectInputStream$BlockDataInputStream.readUTFSpan(ObjectInputStream.java:3143)
...
java.io.ObjectInputStream.readObject0(ObjectInputStream.java:1350)
java.io.ObjectInputStream.readObject(ObjectInputStream.java:370)
scala.collection.immutable.$colon$colon.readObject(List.scala:362)
sun.reflect.GeneratedMethodAccessor2.invoke(Unknown Source)
sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
java.lang.reflect.Method.invoke(Method.java:606)
java.io.ObjectStreamClass.invokeReadObject(ObjectStreamClass.java:1017)
```

Similar StackOverflow error



Spark: PageRank example when iteration too large throws stackoverflowError

4

I test the spark default PageRank example and set the iteration to 1024, then it throws stackoverflowerror. I also met this problem in my other program.How can i solve it.

```
object SparkPageRank {
  def main(args: Array[String]) {
    if (args.length < 3) {
      System.err.println("Usage: PageRank <master> <file> <number_of_iterations>")
      System.exit(1)
```

asked 1

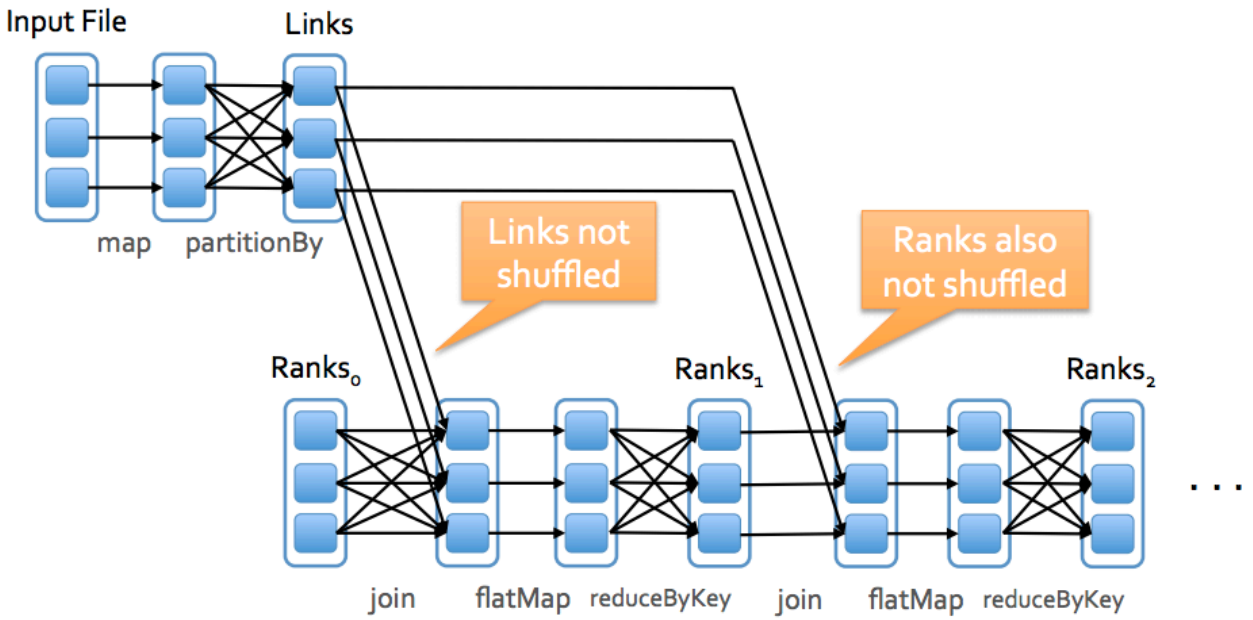
viewed 7

active 4

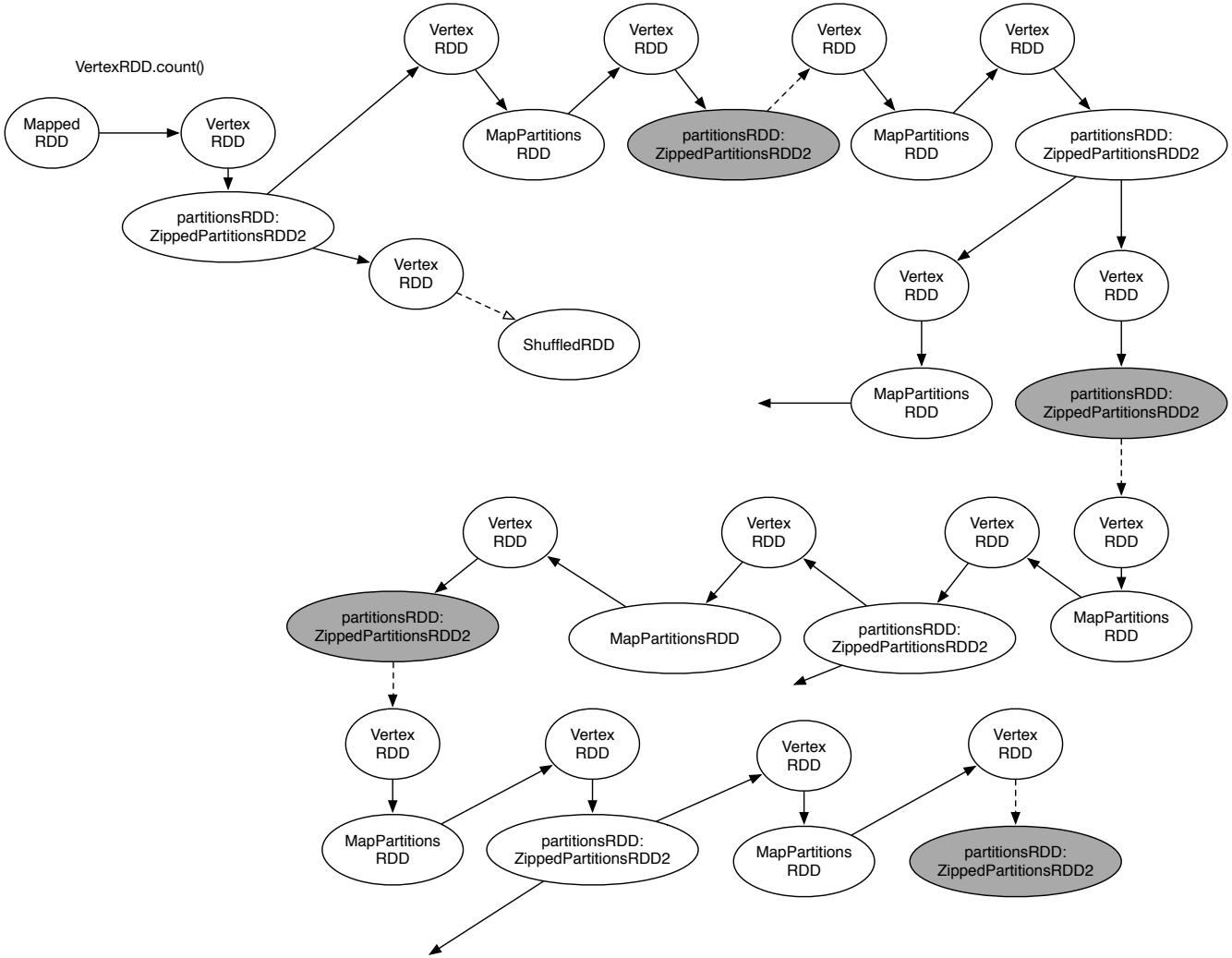
http://stackoverflow.com/questions/22321202/spark-pagerank-example-when-iteration-too-large-throws-stackoverflowerror

Iterative apps: PageRank

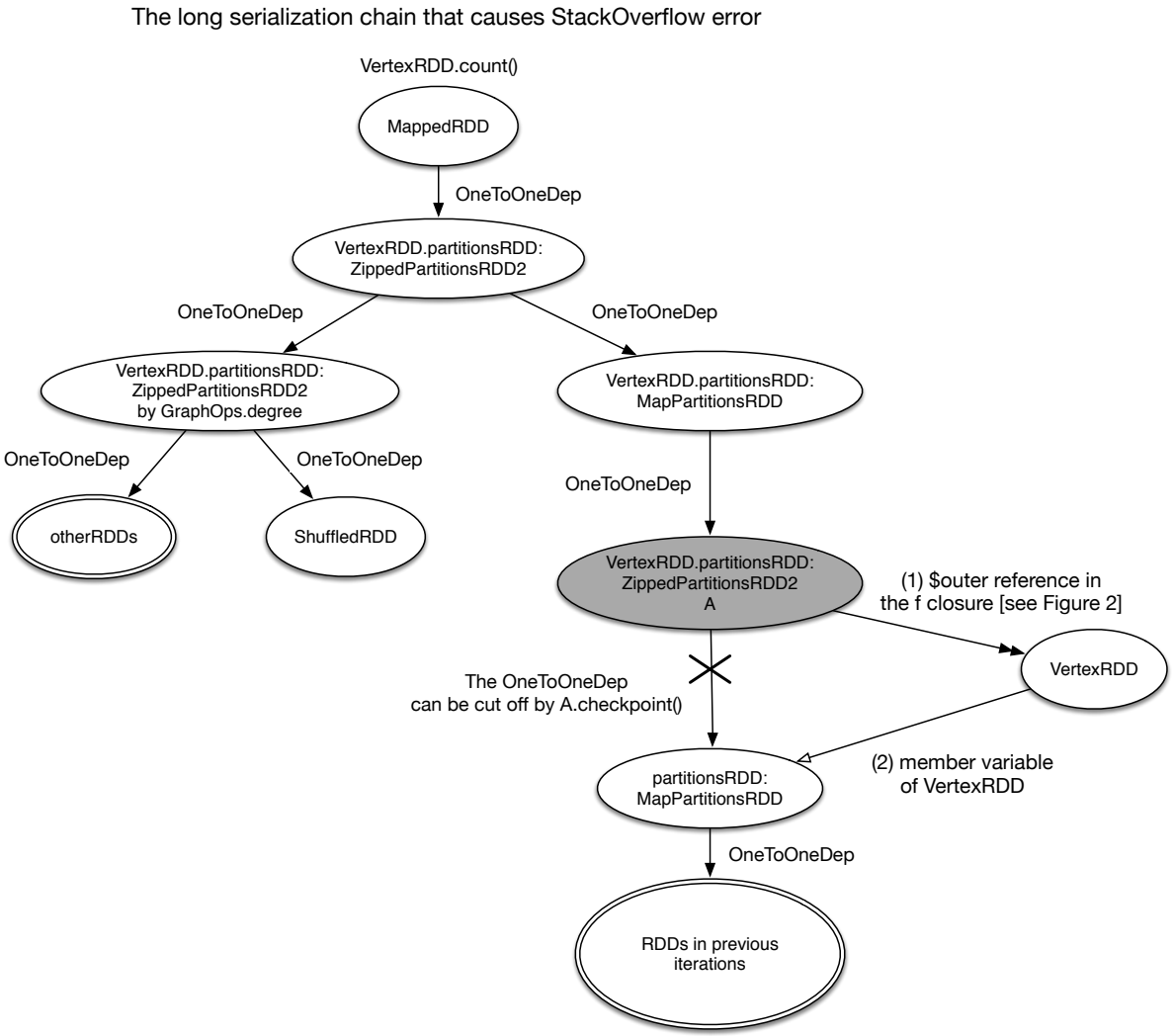
```
val links = // RDD of (url, neighbors) pairs
var ranks = // RDD of (url, rank) pairs
for (i <- 1 to ITERATIONS) {
  val contribs = links.join(ranks).flatMap {
    case (url, (links, rank)) =>
      links.map(dest => (dest, rank / links.size))
  }
  ranks = contribs.reduceByKey(_ + _).mapValues(.15 +.85 * _)
}
```



Lineage of K-Core (intermediate data)



Causes of Stackoverflow error

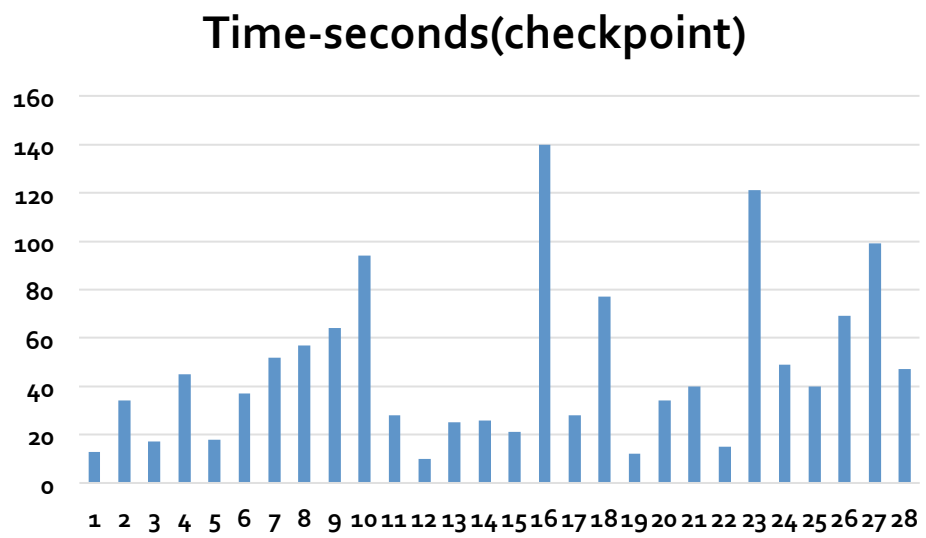


Problems of Checkpointing

- Which intermediate data need checkpointing?
  - *(rdd1 => (rdd2.1, rdd2.2, …, rdd2.n) => rdd3)*
- When to checkpoint them?
  - *Iteration 1, 2, .., m?*
- How to improve checkpointing performance?

Time cost of checkpoint in K-Core

T(checkpoint)
13
34
17
45
18
37
52
57
64
94
28
10
25
26
21
140
28
77
12
34
40
15
121
49
40
69
99
47
Total: 1312

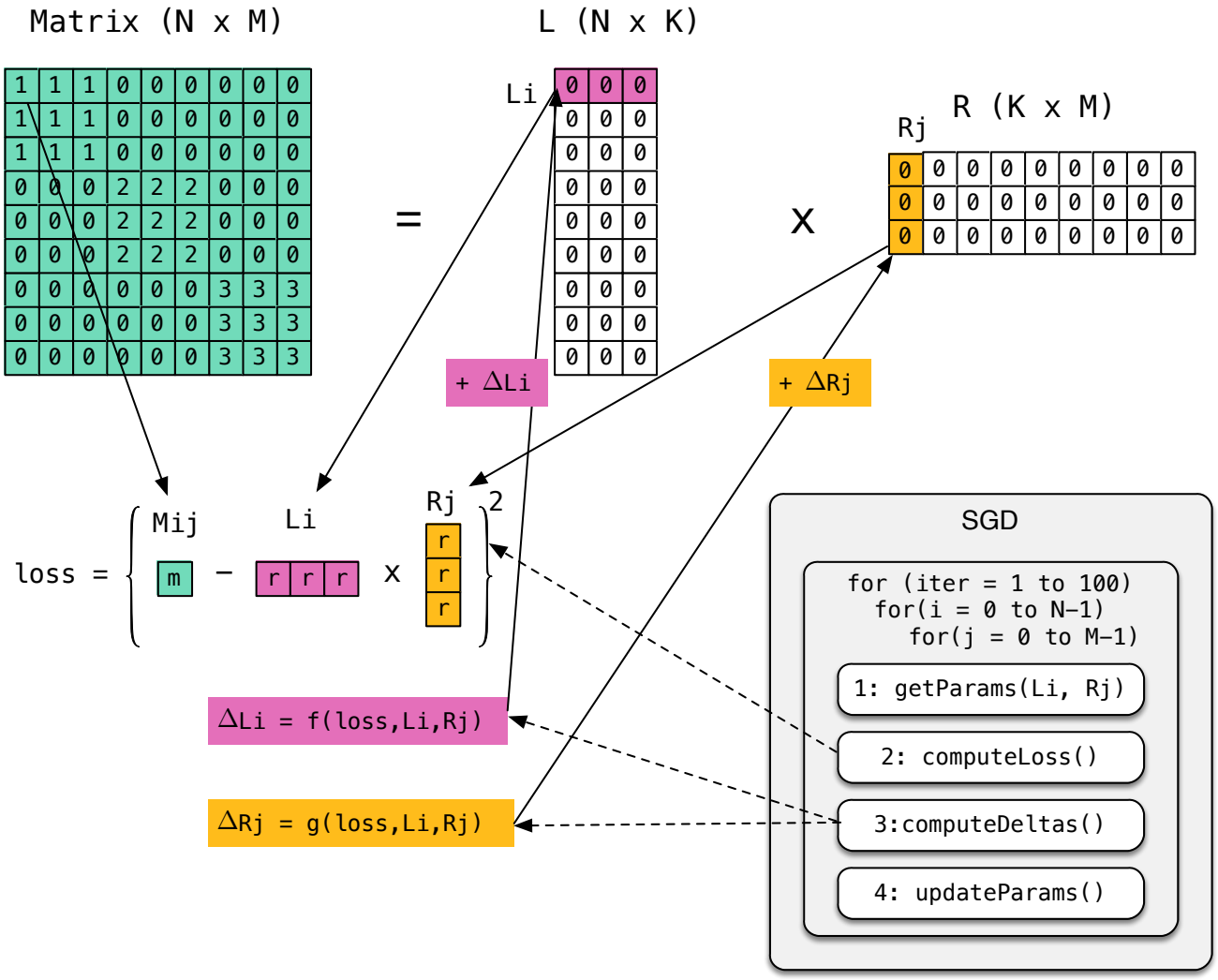


Total execution time	Total(Checkpoint)
2 hours, 30 minutes	20+ minutes

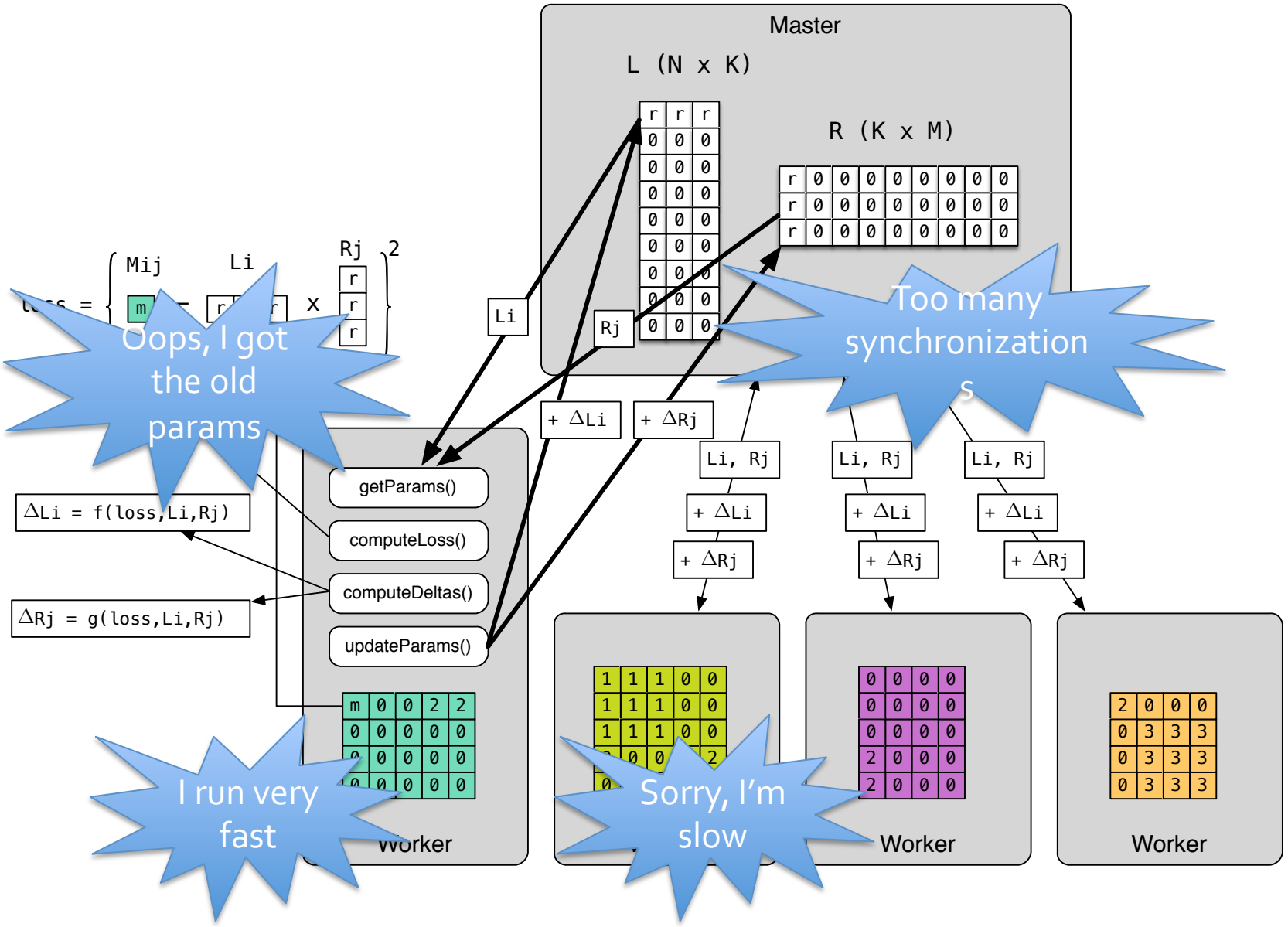
Topic 3

- Consistency model of Parameter Server (optional)

Iterative apps: Matrix Factorization



Problems


Master
L (N x K)
R (K x M)
Li
Rj
+ ΔLi
+ ΔRj
Too many synchronization s
Oops, I got the old params
ΔLi = f(loss,Li,Rj)
ΔRj = g(loss,Li,Rj)
getParams()
computeLoss()
computeDeltas()
updateParams()
Li, Rj
I run very fast
Sorry, I'm slow
Worker
Worker
Worker

Parameter Server