Resilient distributed datasets

Łukasz Dudziak

Motivation

Computational frameworks were inefficient when handling iterative algorithms. Two main problems were identified by the authors (all referred frameworks had at least one):

- Bad support for applications which would like to *reuse* intermediate results
  - Either no mechanism for efficient reuse at all (*i.e*. only by using external storage),
  - Or possible only for specific computation patterns (*e.g*. support only for iterative MapReduce)
- Very costly fault tolerance due to fine-grained nature of a framework

Proposed solution

- Inefficient data reuse → provide user with option to specify which data should be cached in memory + later schedule tasks taking data-locality into consideration
- Inefficient fault recovery → represent memory in terms of data source and coarse-grained transformations, *i.e.* care not about data itself but *how to get it*

- Resilient distributed dataset (RDD) is an abstraction designed to implement both

Proposed solution

- RDDs:
  - Are immutable
  - Can be created from fault-tolerant data storage (*e.g.* HDFS) or by applying coarse-grained transformation to another RDD
  - Store list of their dependencies (other RDDs) and data partitioning information
    - Dependencies can be either wide or narrow
  - Can be used to recover data in case of node failure
  - Can be viewed as a DAG where each node is an intermediate result and edges represent transformations
  - Are executed lazily
  - Are lightweight

Proposed solution

Example of PageRank code written in spark and resulting DAG

```
val links = spark.textFile(...).map(...).persist()
var ranks = // RDD of (URL, rank) pairs
for (i <- 1 to ITERATIONS) {
  // Build an RDD of (targetURL, float) pairs
  // with the contributions sent by each page
  val contribs = links.join(ranks).flatMap {
    (url, (links, rank)) =>
      links.map(dest => (dest, rank/links.size))
  }
  // Sum contributions by URL and get new ranks
  ranks = contribs.reduceByKey((x,y) => x+y)
             .mapValues(sum => a/N + (1-a)*sum)
}
```

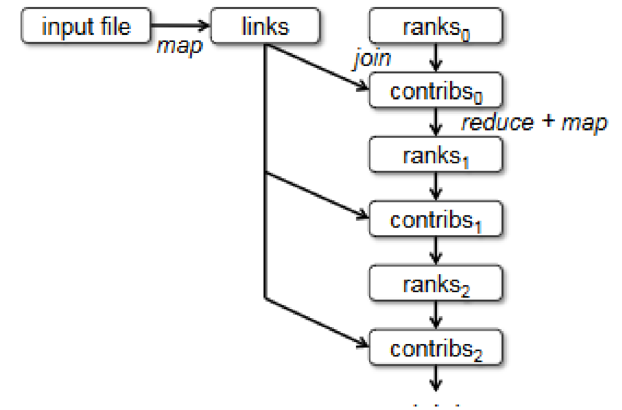

Figure 3: Lineage graph for datasets in PageRank.

Source: *Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing*, M. Zaharia, et al.

Evaluation

- The authors have shown that their system achieves significant speedup comparing to Hadoop when running iterative algorithms
  - The goal seems to be achieved
- It has also been shown that RDD abstraction is generic enough to express many programming models
  - So the criticism of existing frameworks has been addressed as well
- It has been shown that system based on RDDs can relatively quickly recover in case of node failure
  - Seems good too

However…

- When it comes to the recovery and fault-tolerance it is not clear if RDDs really have met all requirements
- Although it has been shown that they are sufficient, efficiency of the recovery depends on the actual DAG structure
  - Section 6.3 does not provide any information whether presented recovery time is average/best/worst case
  - Recovery from RDD **can** be fast but it's not guaranteed
  - Authors have admitted that checkpointing can still be desired in cases when recovery solely from RDD's lineage may be expensive
    - On the other hand, it may be enough to checkpoint only specific RDDs so still better than saving global state of whole system

- In general: for me fault-tolerance could have been described more in-detail since many things are not obvious

The End

Thank you for attention.