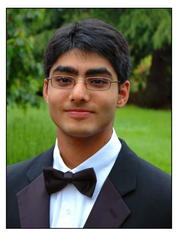
Bagel

Large-Scale Graph Processing on Spark

Ankur Dave, Matei Zaharia, Scott Shenker, Ion Stoica

amplab



Motivation

- Google's Pregel is a convenient programming model for graph problems
- *Can we build a concise, efficient implementation of Pregel on top of Spark?*

Why Spark?

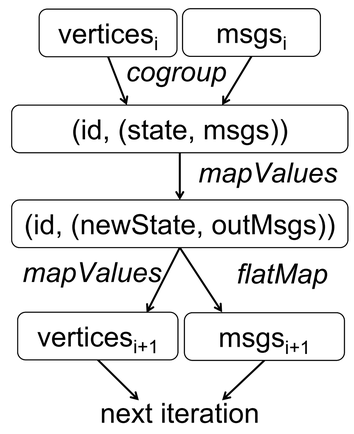
- *Ease of programming* using Scala and Spark: Bagel is only ~150 lines of Scala
- *Efficiency* from controllable partitioning and in-memory working sets: Bagel is 2.6× faster than Hadoop for PageRank

Pregel Model

- Programs run as a series of iterations called *supersteps*
- On each superstep, *vertices* run user code that can update vertex state and pass *messages* to others for next superstep
- *Combiners* reduce communication by merging messages to the same vertex
- *Aggregators* reduce vertex values and send result to all vertices in next superstep
- Natural for graph algorithms: shortest path, bipartite matching, PageRank

Architecture

- Vertex states and sent messages stored as RDDs
- Message passing done with cogroup on vertex ID (establishes partitioning)
- Computation done with mapValues (preserves partitioning)

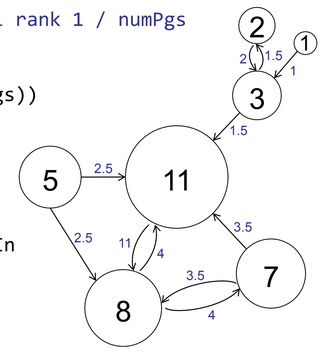


PageRank Example

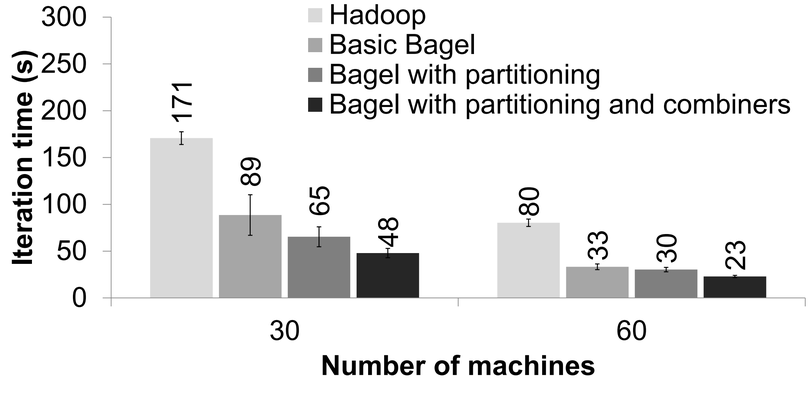
```
// Load the pages into an RDD with initial rank 1 / numPgs
file = spark.textFile("hdfs://...")
numPgs = file.count()
pages = file.map(new PRVertex(_, 1 / numPgs))

// Run PageRank for 30 iterations
ranked = Pregel.run(pages) {
  (pg, msgs, step) =>
    rankIn = msgs.map(_.value).sum
    newRank = 0.15 / numPgs + 0.85 * rankIn
    rankOut = newRank / pg.links.size
    outMsgs = pg.links.map(
      new Message(_, rankOut))
    newPage = pg.copy(rank = newRank,
                      halt = step >= 30)
    (newPage, outMsgs)
}

// Collect the results as a mapping from page ID to rank
result = ranked.map(_.id -> _.rank).collect
```



PageRank Results



Shortest Path Example

```
// Load the vertices into an RDD with initial distance infinity
file = spark.textFile("hdfs://...")
verts = file.map(new SPVertex(_, PositiveInfinity))

// Send a message from the source vertex to start the computation
initMsg = spark.parallelize(List(new SPMessage(sourceVert, 0)))

// Run Shortest Path until no messages are sent
dists = Pregel.run(verts, initMsg)(min(_, _.value)){
  (vert, minMsg, step) =>
    newVal = min(vert.value, minMsg)
    outMsgs =
      if (newVal != vert.value)
        vert.edges.map(edge =>
          new SPMessage(edge.target, newVal + edge.val))
      else
        List()
    (vert.copy(value = newVal), outMsgs)
}

// Collect the results as a mapping from vertex ID to distance
result = dists.map(_.id -> _.value).collect
```

Future Plans

1. *Improved efficiency* through user-specified partitioning to reduce communication
2. *Checkpointing* for mutable vertex values
3. *Complete Pregel support* including mutation of graph topology