PageRank and recommenders on very large scale

A Big Data perspective through Stratosphere

Márton Balassi
Data Mining and Search Group[1]

[1]Computer and Automation Research Institute of the Hungarian Academy of Sciences

May 8, 2014

DMS

Table of Contents

Distributing data-intensive algorithms

Stratosphere Input Contracts

PageRank and recommender systems

Reference

DMS

Table of contents

Distributing data-intensive algorithms

Stratosphere Input Contracts

PageRank and recommender systems

Reference

Motivation

Let's do a PageRank on this graph...

- The soc-LiveJournal1 provided by Stanford LNDC[1]
- $4.8 \cdot 10^6$ nodes
- $6.9 \cdot 10^7$ edges
- 250 MB of compressed data
- „Conventional" single machine solution seems sufficient

[1] Stanford Large Network Dataset Collection

DMS

Motivation

Let's do a PageRank on this graph...

- The soc-LiveJournal1 provided by Stanford LNDC[1]
- $4.8 \cdot 10^6$ nodes
- $6.9 \cdot 10^7$ edges
- 250 MB of compressed data
- „Conventional" single machine solution seems sufficient

[1] Stanford Large Network Dataset Collection

Motivation

Let's do a PageRank on this graph...

- The soc-LiveJournal1 provided by Stanford LNDC[1]
- $4.8 \cdot 10^6$ nodes
- $6.9 \cdot 10^7$ edges
- 250 MB of compressed data
- „Conventional" single machine solution seems sufficient

[1] Stanford Large Network Dataset Collection

Motivation

Let's do a PageRank on this graph...

- The soc-LiveJournal1 provided by Stanford LNDC[1]
- $4.8 \cdot 10^6$ nodes
- $6.9 \cdot 10^7$ edges
- 250 MB of compressed data
- „Conventional" single machine solution seems sufficient

[1] Stanford Large Network Dataset Collection

Motivation

Let's do a PageRank on this graph...

- The soc-LiveJournal1 provided by Stanford LNDC[1]
- $4.8 \cdot 10^6$ nodes
- $6.9 \cdot 10^7$ edges
- 250 MB of compressed data
- „Conventional" single machine solution seems sufficient

[1] Stanford Large Network Dataset Collection

Motivation

Let's do a PageRank on this graph...

- A large Portugese webcrawl[1]
- $3.1 \cdot 10^9$ nodes
- $1.1 \cdot 10^{11}$ edges
- 80 GB of compressed data
- Divide and conquer is almost mandatory

[1] a large Portuguese crawl of the Portuguese Web Archive obtained from Daniel Gomes

Motivation

Let's do a PageRank on this graph...

- A large Portugese webcrawl[1]
- $3.1 \cdot 10^9$ nodes
- $1.1 \cdot 10^{11}$ edges
- 80 GB of compressed data
- Divide and conquer is almost mandatory

[1] a large Portuguese crawl of the Portuguese Web Archive obtained from Daniel Gomes

Motivation

Let's do a PageRank on this graph...

- A large Portugese webcrawl[1]
- $3.1 \cdot 10^9$ nodes
- $1.1 \cdot 10^{11}$ edges
- 80 GB of compressed data
- Divide and conquer is almost mandatory

[1] a large Portuguese crawl of the Portuguese Web Archive obtained from Daniel Gomes

Motivation

Let's do a PageRank on this graph...

- A large Portugese webcrawl[1]
- $3.1 \cdot 10^9$ nodes
- $1.1 \cdot 10^{11}$ edges
- 80 GB of compressed data
- Divide and conquer is almost mandatory

[1] a large Portuguese crawl of the Portuguese Web Archive obtained from Daniel Gomes

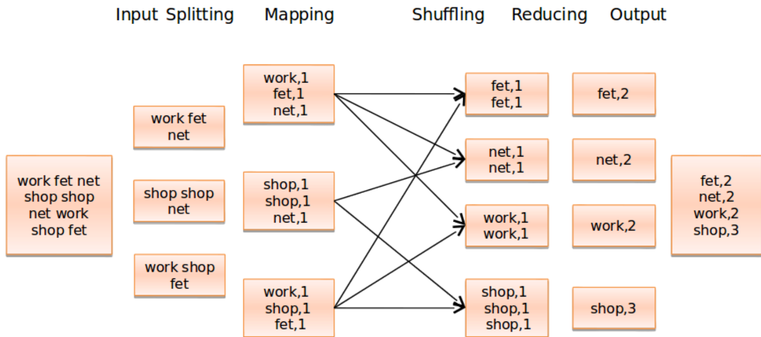
Motivation

Let's do a PageRank on this graph...

- A large Portugese webcrawl[1]
- $3.1 \cdot 10^9$ nodes
- $1.1 \cdot 10^{11}$ edges
- 80 GB of compressed data
- Divide and conquer is almost mandatory

[1] a large Portuguese crawl of the Portuguese Web Archive obtained from Daniel Gomes

MapReduce



Pregel

Traits

- Bulk Synchronous Parallel
- „Think like a vertex"
- Graph kept in memory

Processors

Local Computation

Communication

Barrier Synchronisation

Scheme of the BSP system
Wikipedia, public domain

Pregel

Traits

- Bulk Synchronous Parallel
- „Think like a vertex”
- Graph kept in memory

Processors

Local
Computation

Communication

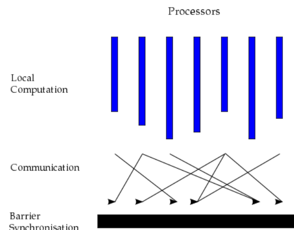
Barrier
Synchronisation

Scheme of the BSP system
Wikipedia, public domain

Pregel

Traits

- Bulk Synchronous Parallel
- „Think like a vertex”
- Graph kept in memory



Scheme of the BSP system
Wikipedia, public domain

Triangle Counter – Sequential algorithm

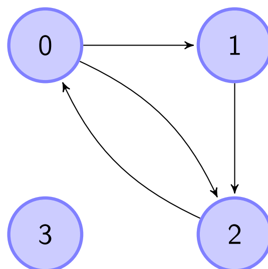
Sequential algorithm

Every vertex executes a search of itself bounded in depth of three. Thus every triangle is counted three times.

Triangle Counter – MapReduce algorithm

Representation

0 1 2
1 2
2 0
3



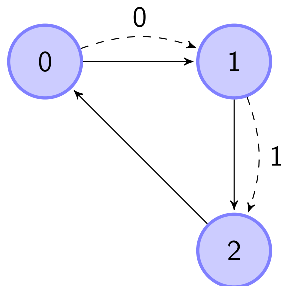
Triangle Counter – MapReduce algorithm

First Map

Let's send our `ID` to all of our neighbours possessing a higher `ID` than ours. Let's send our neighbours to ourselves.

First Reduce

Let's write out the information received.



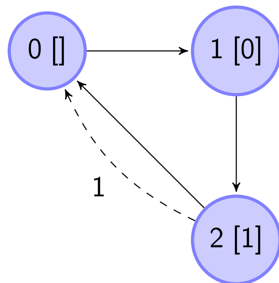
Triangle Counter – MapReduce algorithm

Second Map

If the `ID` received is smaller then ours let's pass it on to our neighbours.
Let's send our neighbours to ourselves.

Second Reduce

If the `ID` received is our neighbour then let's increment a global counter.



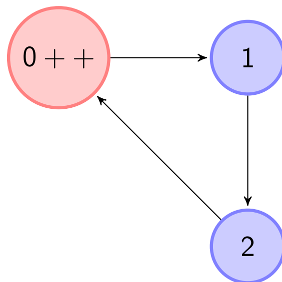
Triangle Counter – MapReduce algorithm

Second `Map`

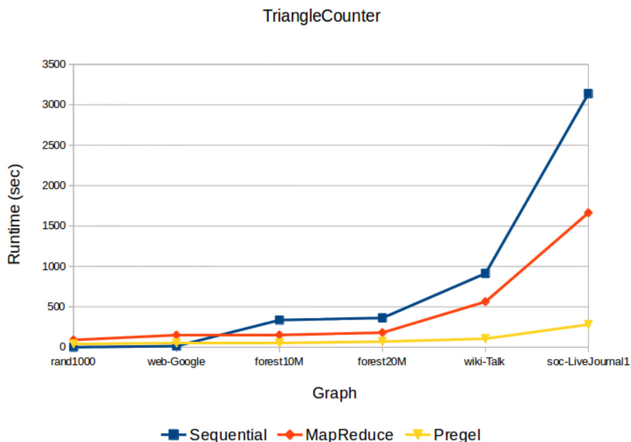
If the `ID` received is smaller then ours let's pass it on to our neighbours.
Let's send our neighbours to ourselves.

Second `Reduce`

If the `ID` received is our neighbour then let's increment a global counter.



Runtime of the three solutions



DMS

Table of contents

Distributing data-intensive algorithms

Stratosphere Input Contracts

PageRank and recommender systems

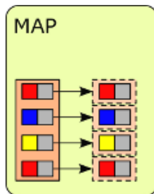
Reference

Map

Wordcount Map

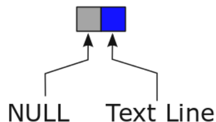
For lines of input text emit $(word, 1)$ for each word.

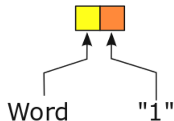
Map

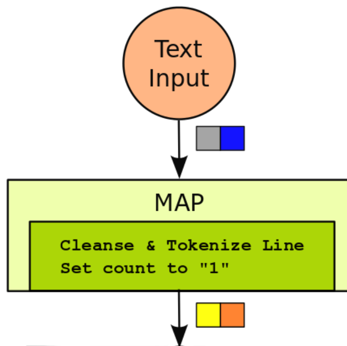



```java
public static class TokenizeLine extends MapStub implements Serializable {
  private static final long serialVersionUID = 1L;

  // initialize reusable mutable objects
  private final PactRecord outputRecord = new PactRecord();
  private final PactString word = new PactString();
  private final PactInteger one = new PactInteger(1);

  @Override
  public void map(PactRecord record, Collector<PactRecord> collector) {
    // get the first field (as type PactString) from the record
    PactString line = record.getField(0, PactString.class);

    // normalize the line with AsciiUtils ...

    // tokenize the line
    this.tokenizer.setStringToTokenize(line);
    while (tokenizer.next(this.word)){
      // emit a (word, 1) pair
      this.outputRecord.setField(0, this.word);
      this.outputRecord.setField(1, this.one);
      collector.collect(this.outputRecord);
    }
  }
}
```

Map







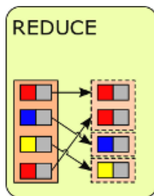
Reduce

Wordcount Reduce

For multiple instances of $(word, 1)$ count frequency of each word.

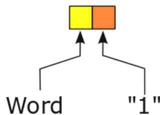
DMS

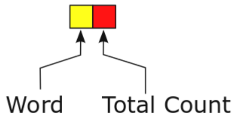
Reduce

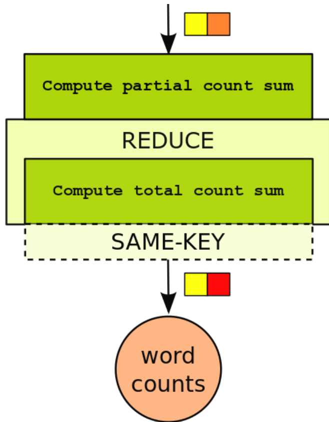



```
public static class CountWords extends ReduceStub implements Serializable {
  private final PactInteger cnt = new PactInteger();

  @Override
  public void reduce(Iterator<PactRecord> records, Collector<PactRecord> out)
         throws Exception {
    PactRecord element = null;
    int sum = 0;
    while (records.hasNext()) {
      element = records.next();
      PactInteger i = element.getField(1, PactInteger.class);
      sum += i.getValue();
    }
    this.cnt.setValue(sum);
    element.setField(1, this.cnt);
    out.collect(element);
  }

  @Override
  public void combine(Iterator<PactRecord> records, Collector<PactRecord> out)
         throws Exception {
  // same logic as reduce so simply a call to it
  this.reduce(records, out);
  }
}
```

Reduce





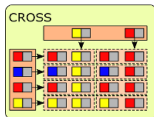


Cross

K-Means Cross

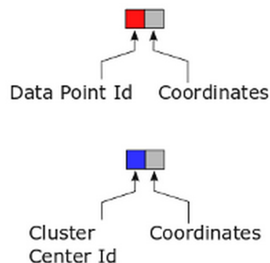
Given data points and cluster centers compute the distance between each data point and cluster center.

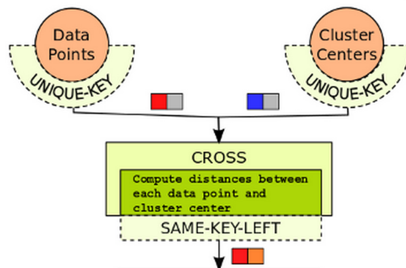
Cross




```
public class ComputeDistance extends CrossStub implements Serializable {
  private static final long serialVersionUID = 1L;
  private final PactDouble distance = new PactDouble();

  //Output Format: (pointID, pointVector, clusterID, distance)
  @Override
  public void cross(PactRecord dataPointRecord, PactRecord clusterCenterRecord,
         Collector<PactRecord> out) {

    CoordVector dataPoint = dataPointRecord.getField(1, CoordVector.class);

    PactInteger clusterCenterId = clusterCenterRecord.getField(0,
          PactInteger.class);
    CoordVector clusterPoint = clusterCenterRecord.getField(1,
          CoordVector.class);

    this.distance.setValue(dataPoint.computeEuclidianDistance(clusterPoint));

    // add cluster center id and distance to the data point record
    dataPointRecord.setField(2, clusterCenterId);
    dataPointRecord.setField(3, this.distance);

  out.collect(dataPointRecord);
  }
}
```

Cross





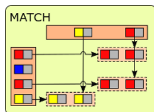
DMS

Match

Path Match

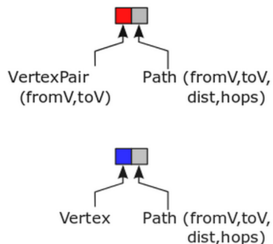
Given edges (e, f) and (f, g) of a graph construct (e, g) paths.

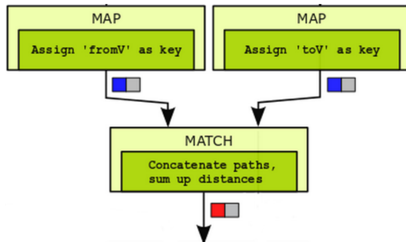
Match




```java
public static class ConcatPaths extends MatchStub implements Serializable {

  //define outputRecord, length, hopCnt, hopList...

  @Override
  public void match(PactRecord rec1, PactRecord rec2, Collector<PactRecord>
         out) throws Exception {
    // rec1 has matching start, rec2 matching end
    final PactString fromNode = rec2.getField(0, PactString.class);
    final PactString toNode = rec1.getField(1, PactString.class);
    if (fromNode.equals(toNode)) return; //circle prevention

    // Create new path
    outputRecord.setField(0, fromNode);
    outputRecord.setField(1, toNode);

    // Compute length of new path & hop count ...
    // Concatenate hops lists and insert matching node...
    // Append the whole path in a Stringbuilder...

    hopList.setValue(sb.toString().trim());
    outputRecord.setField(4, hopList);
    out.collect(outputRecord);
  }
}
```

Match



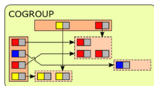


CoGroup

Floyd CoGroup

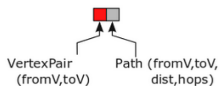
Given shortest paths to inneighbours of a vertex in a directed graph and the edges of the graph compute the shortest path to the vertex.

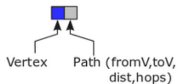
CoGroup

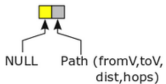



```
public static class FindShortestPath extends CoGroupStub implements
      Serializable {
  // define outputRecord, shortestPaths, hopCnts, minLength ...

  @Override
  public void coGroup(Iterator<PactRecord> inputRecords, Iterator<PactRecord>
         concatRecords, Collector<PactRecord> out) {
    // init minimum length and minimum path ...
    // find shortest path of all input paths...
    // find shortest path of all input and concatenated paths...

    outputRecord.setField(0, fromNode);
    outputRecord.setField(1, toNode);
    outputRecord.setField(2, minLength);

    // emit all shortest paths
    for(PactString shortestPath : shortestPaths) {
      outputRecord.setField(3, hopCnts.get(shortestPath));
      outputRecord.setField(4, shortestPath);
      out.collect(outputRecord);
    }
    hopCnts.clear();
    shortestPaths.clear();
  }
}
```

CoGroup







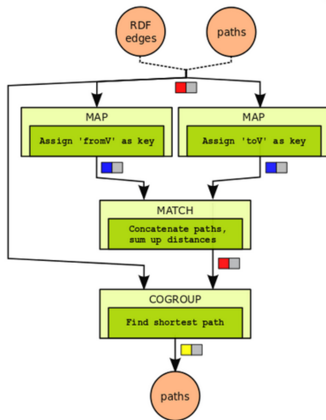


Table of contents

Distributing data-intensive algorithms

Stratosphere Input Contracts

PageRank and recommender systems

Reference

Iterations in Stratosphere

Denotation

- S is a partitioned dataset
- f is a Stratosphere program
- $<$ is a termination criterion

Iterations in Stratosphere

Denotation

- S is a partitioned dataset
- f is a Stratosphere program
- $<$ is a termination criterion

Iterations in Stratosphere

Denotation

- S is a partitioned dataset
- f is a Stratosphere program
- $<$ is a termination criterion

Iterations in Stratosphere

Denotation

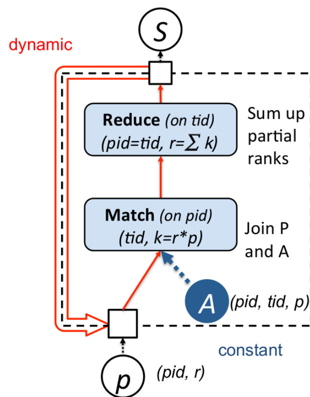
- S is a partitioned dataset
- f is a Stratosphere program
- $<$ is a termination criterion

1: **while** $S < f(S)$ **do**
2: do $S := f(S)$

Bulk iterations

Traits

- Each iteration is a synchronization point (superstep)
- Optimizer weighs costs of dynamic data path with iterations
- Caches where data paths meet
- Pushes repeated work to constant data path

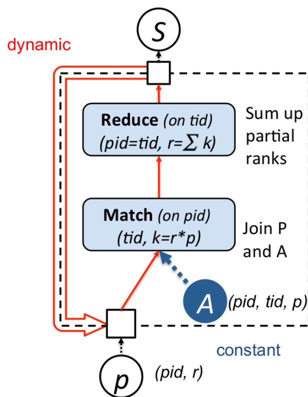


PageRank scheme

Bulk iterations

Traits

- Each iteration is a synchronization point (superstep)
- Optimizer weighs costs of dynamic data path with iterations
- Caches where data paths meet
- Pushes repeated work to constant data path



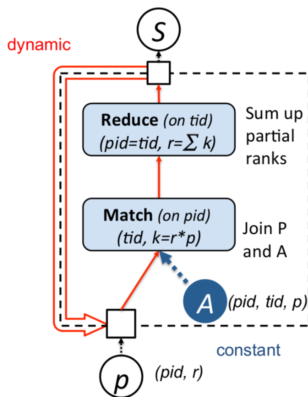
PageRank scheme

DMS

Bulk iterations

Traits

- Each iteration is a synchronization point (superstep)
- Optimizer weighs costs of dynamic data path with iterations
- Caches where data paths meet
- Pushes repeated work to constant data path



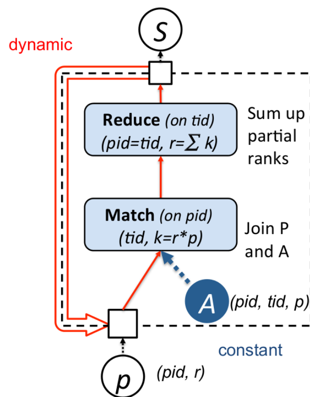
PageRank scheme

DMS

Bulk iterations

Traits

- Each iteration is a synchronization point (superstep)
- Optimizer weighs costs of dynamic data path with iterations
- Caches where data paths meet
- Pushes repeated work to constant data path



PageRank scheme

Incremental iterations

Rationale

- New construct: incremental (workset) iteration
- W contains elements from S that may change in the next iteration
- D computed from S, W and efficiently merged with prior S
  Workset W recomputed from D

Incremental iterations

Rationale

- New construct: incremental (workset) iteration
- W contains elements from S that may change in the next iteration
- D computed from S, W and efficiently merged with prior S
  Workset W recomputed from D

Incremental iterations

Rationale

- New construct: incremental (workset) iteration
- W contains elements from S that may change in the next iteration
- D computed from S, W and efficiently merged with prior S
  Workset W recomputed from D

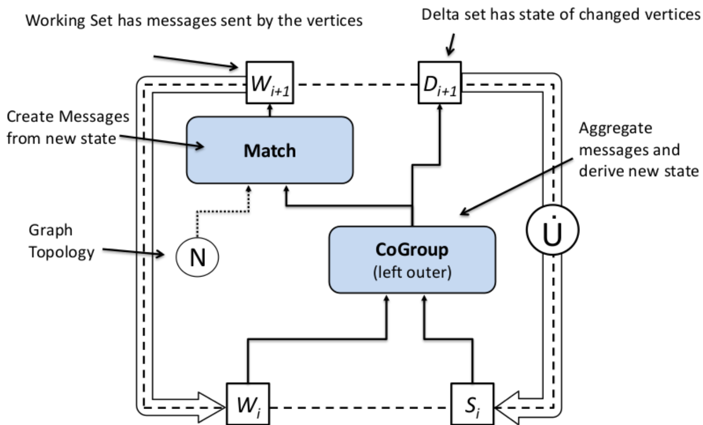
Incremental iterations

Rationale

- New construct: incremental (workset) iteration
- W contains elements from S that may change in the next iteration
- D computed from S, W and efficiently merged with prior S
  Workset W recomputed from D

1: $S := I,\ W := S$
2: **while** $W \neq \emptyset$ **do**
3: $\quad D := u(S, W)$
4: $\quad W := \delta(D, S, W)$
5: $\quad S := S \uplus W$

Pregel as a Stratosphere job



DMS

Recommender systems

Alternating Least Squares (ALS)

- We have a U user and an I itemset
- The users rating are stored in $\mathbf{R} \in \mathbb{R}^{|U| \times |I|}$
- But $|U|$ and $|I|$ can easily be at the range of millions...
- Let's find $\mathbf{P}$ and $\mathbf{Q}$ such that $\mathbf{PQ} \approx \mathbf{R}$
- Let $\mathbf{P} \in \mathbb{R}^{|U| \times k}$ and $\mathbf{Q} \in \mathbb{R}^{k \times |I|}$, where k is a small constant
- The algorithm uses least squares to estimate, alternating for $\mathbf{P}$ and $\mathbf{Q}$

DMS

Recommender systems

Alternating Least Squares (ALS)

- We have a U user and an I itemset
- The users rating are stored in $\mathbf{R} \in \mathbb{R}^{|U| \times |I|}$
- But $|U|$ and $|I|$ can easily be at the range of millions…
- Let's find $\mathbf{P}$ and $\mathbf{Q}$ such that $\mathbf{PQ} \approx \mathbf{R}$
- Let $\mathbf{P} \in \mathbb{R}^{|U| \times k}$ and $\mathbf{Q} \in \mathbb{R}^{k \times |I|}$, where k is a small constant
- The algorithm uses least squares to estimate, alternating for $\mathbf{P}$ and $\mathbf{Q}$

DMS

Recommender systems

Alternating Least Squares (ALS)

- We have a U user and an I itemset
- The users rating are stored in $\mathbf{R} \in \mathbb{R}^{|U| \times |I|}$
- But $|U|$ and $|I|$ can easily be at the range of millions...
- Let's find $\mathbf{P}$ and $\mathbf{Q}$ such that $\mathbf{PQ} \approx \mathbf{R}$
- Let $\mathbf{P} \in \mathbb{R}^{|U| \times k}$ and $\mathbf{Q} \in \mathbb{R}^{k \times |I|}$, where k is a small constant
- The algorithm uses least squares to estimate, alternating for $\mathbf{P}$ and $\mathbf{Q}$

Recommender systems

Alternating Least Squares (ALS)

- We have a U user and an I itemset
- The users rating are stored in $\mathbf{R} \in \mathbb{R}^{|U| \times |I|}$
- But $|U|$ and $|I|$ can easily be at the range of millions...
- Let's find $\mathbf{P}$ and $\mathbf{Q}$ such that $\mathbf{PQ} \approx \mathbf{R}$
- Let $\mathbf{P} \in \mathbb{R}^{|U| \times k}$ and $\mathbf{Q} \in \mathbb{R}^{k \times |I|}$, where k is a small constant
- The algorithm uses least squares to estimate, alternating for $\mathbf{P}$ and $\mathbf{Q}$

Recommender systems

Alternating Least Squares (ALS)

- We have a U user and an I itemset
- The users rating are stored in $\mathbf{R} \in \mathbb{R}^{|U| \times |I|}$
- But $|U|$ and $|I|$ can easily be at the range of millions…
- Let's find $\mathbf{P}$ and $\mathbf{Q}$ such that $\mathbf{PQ} \approx \mathbf{R}$
- Let $\mathbf{P} \in \mathbb{R}^{|U| \times k}$ and $\mathbf{Q} \in \mathbb{R}^{k \times |I|}$, where k is a small constant
- The algorithm uses least squares to estimate, alternating for $\mathbf{P}$ and $\mathbf{Q}$

Recommender systems

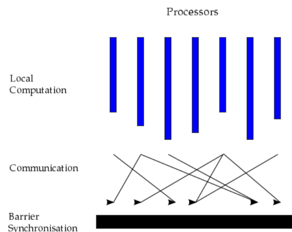
Alternating Least Squares (ALS)

- We have a U user and an I itemset
- The users rating are stored in $\mathbf{R} \in \mathbb{R}^{|U| \times |I|}$
- But $|U|$ and $|I|$ can easily be at the range of millions...
- Let's find $\mathbf{P}$ and $\mathbf{Q}$ such that $\mathbf{PQ} \approx \mathbf{R}$
- Let $\mathbf{P} \in \mathbb{R}^{|U| \times k}$ and $\mathbf{Q} \in \mathbb{R}^{k \times |I|}$, where k is a small constant
- The algorithm uses least squares to estimate, alternating for $\mathbf{P}$ and $\mathbf{Q}$

Limitations of BSP

Challenge

- Algorithmic and physical partitions are different to utilize cpus
- In PageRank its OK to send the same rank multiple times
- In ALS it means duplicating the matrix each time!

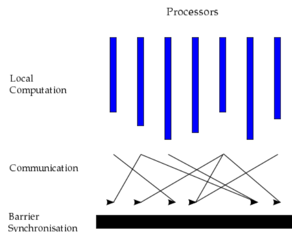


Scheme of the BSP system
Wikipedia, public domain

Limitations of BSP

Challenge

- Algorithmic and physical partitions are different to utilize cpus
- In PageRank its OK to send the same rank multiple times
- In ALS it means duplicating the matrix each time!

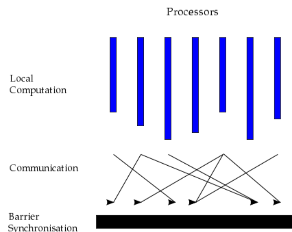


Scheme of the BSP system
Wikipedia, public domain

Limitations of BSP

Challenge

- Algorithmic and physical partitions are different to utilize cpus
- In PageRank its OK to send the same rank multiple times
- In ALS it means duplicating the matrix each time!



Scheme of the BSP system
Wikipedia, public domain

Possible solution

Proposed new Stratosphere input contract

Given a set of values p_i indexed by i, and a relation R_{ij} over the index set, form the co-group $\forall j$ as:

$$j : p_i \text{ for } R_{ij}$$

Possible solution

Proposed new Stratosphere input contract

Given a set of values p_i indexed by i, and a relation R_{ij} over the index set, form the co-group $\forall j$ as:

$$j : p_i \text{ for } R_{ij}$$

In other words, a directed graph defines the values p_i that have to be aggregated at nodes j.

DMS

Possible solution

Proposed new Stratosphere input contract

Given a set of values p_i indexed by i, and a relation R_{ij} over the index set, form the co-group $\forall j$ as:

$$j : p_i \text{ for } R_{ij}$$

In other words, a directed graph defines the values p_i that have to be aggregated at nodes j.
Both ALS and PageRank (and I guess may more) use this Input Contract.

Table of contents

Distributing data-intensive algorithms

Stratosphere Input Contracts

PageRank and recommender systems

Reference

Literature

„TriangleCounter”

Englert et al. (2014): Efficiency Issues of Computing Graph Properties of Social Networks, *Presented at The 9th International Conference on Applied Informatics, Eger,* proceedings are under publish.

Stratosphere PACTs

Battré et al. (2010): Nephele/PACTs: a programming model and execution framework for web-scale analytical processing, *Proceedings of the 1st ACM symposium on Cloud computing,* p119-130.

DMS

On the web

Data Mining and Search & Big Data BI Groups

Our research groups can be found at `dms.sztaki.hu` and at `bigdatabi.sztaki.hu`.

Stratosphere project homepage

The project can be found at `stratosphere.eu`.
The homepage served as a source for all the images and code presented on these slides.

DMS