Optimizing MapReduce for Highly-Distributed Environments

Abhishek Chandra
Associate Professor
Department of Computer Science and Engineering
University of Minnesota

http://www.cs.umn.edu/~chandra
Big Data

- **Data-rich** enterprises and communities
  - Both user-facing services and batch data processing
  - Commercial, social, scientific
  - E.g.: Google, Facebook, Amazon, Akamai, LHC, ...

- **Data analysis is key**
  - Search and indexing
  - Ad optimization
  - Account and billing
  - Spam detection and network monitoring
  - Scientific data analysis
Geographically Distributed Data

- Commercial. E.g.: Warehouse, ecommerce data
- Public/social. E.g.: User blogs, traffic data
- Access logs. E.g.: CDNs
- Scientific. E.g.: oceanic, atmospheric data
- Mobile. E.g.: phone pics, sensors
Distributed Computation Resources

• Distributed data centers/clouds
  – E.g.: Amazon EC2 regions

• Edge servers
  – E.g.: Akamai CDN servers

• Computational Grids
  – E.g.: FutureGrid, BOINC
Highly Distributed Environments

- **Question**: How to analyze distributed data **efficiently** in such environments?
Talk Outline

• Motivation
• **Highly-Distributed MapReduce**
• Our Research: MapReduce Optimization
• Concluding Remarks
Highly Distributed Computation

• Data import

• Initial embarrassingly parallel computation

• Grouping / reorganization

• Final summarizing computation
Highly Distributed Computation

• Data import

• Initial embarrassingly parallel computation

• Grouping / reorganization

• Final summarizing computation

  Push

  Map

  Shuffle

  Reduce
Highly Distributed MapReduce

• **Our focus:** Efficient execution of MapReduce in highly-distributed environments

• MapReduce is simple and powerful:
  – Designed for scalability and fault-tolerance
  – Can express several data analysis algorithms

• MapReduce is widely used
  – Popularized by open-source Hadoop project
  – A rich eco-system of higher-level languages, tools
MapReduce Dataflow

Push  Map  Shuffle  Reduce

Input Data
Traditional MapReduce

Network and compute nodes largely homogeneous
Highly-Distributed MapReduce

Push

Map
Shuffle
Reduce

Datacenter 1

Datacenter N

Input Data 1

Input Data N
Problem: Heterogeneity

How can MapReduce handle this *heterogeneity*?
Possible ‘Solutions’

• Centralized Execution
  – Push data over WAN
  – May limit parallelism
  – Problem if large input data

• Local push
  – Shuffle over WAN
  – Poor load balancing
  – Problem if large intermediate data
Experimental Results: Amazon EC2

Amazon EC2: 6 US, 3 EU small instances, 1 data node each

- Data Push cost dominant
- Shuffle cost dominant

Performance depends on network, application characteristics

WordCount (Text) – Large input data
WordCount (Random) – Large intermediate data
Talk Outline

• Motivation
• Highly-Distributed MapReduce
• **Our Research: MapReduce Optimization**
• Concluding Remarks
Optimizing MapReduce: Key Ideas

• **Heterogeneity-aware execution**
  – Data placement and task scheduling should consider network locality, node speeds

• **Application-aware optimization**
  – High data aggregation => Reduce push cost
  – Low data aggregation => Reduce shuffle cost

• **Make globally optimal decisions**
  – Optimize across phase boundaries by factoring in downstream effects
Research Overview

• Approach 1: Model-driven MapReduce optimization
• Approach 2: Cross-phase optimization in Hadoop
Approach 1: Model-Driven Optimization

- **Key idea:** optimize multiple phases to minimize end-to-end execution time
- **Model** MapReduce data flow
- Using model, derive *optimal execution plan*
MapReduce Execution Model

• Parameters
  – $D_i$ – Size of input data at data source $i$
  – $B_{ij}$ – Link bandwidth from node $i$ to node $j$
  – $C_i$ – Mapper/Reducer compute rates
  – $\alpha$ – Ratio of size$_{out}$/size$_{in}$ for map phase

• Execution plan
  – *Each* source: where to push data
  – *Each* mapper: where to shuffle data
Optimization

• **Objective:** minimize *makespan*

• **Constraints**
  
  – Each data source (mapper) must push (shuffle) all of its data

  \[
  \forall (i, j) \in E : 0 \leq x_{ij} \leq 1
  \]

  \[
  \forall i \in V : \sum_{(i, j) \in E} x_{ij} = 1
  \]

  – One-reducer-per-key: \(y_k\) denotes fraction reduced at reducer \(k\)

  \[
  \forall j \in M, k \in R : x_{jk} = y_k
  \]
Obvious ‘Solutions’ Aren’t

PlanetLab measurements: 4 US, 2 Europe, 2 Asia nodes; 1 data source each

α = 0.1

α = 10

Neither purely \{centralized, distributed\} is always better.
Benefit of Optimization

PlanetLab measurements: 4 US, 2 Europe, 2 Asia nodes; 1 data source each

Model-driven optimization performs best under different scenarios

α=0.1 (Data Aggregation)

α=10 (Data Expansion)
Comparison to Hadoop

Emulated PlanetLab, Hadoop 1.0.1 (Modified for model-based execution plans)

Optimized plan outperforms Hadoop for different applications
Approach 2: Cross-phase Optimization in Hadoop

• **Key idea:** factor in downstream effects

• Proactive techniques:
  – Map-aware Push
  – Shuffle-aware Map

• Implemented in Hadoop 1.0.1
Push/Map Barrier

Push, then Map

Push/map barrier:
- Waiting → waste
- Mappers cannot *demand* more or less work
Map-aware Push

• *Pipeline* push and map
  – Hide latency
  – Feedback: mappers *pull* on demand

• *Infer* locality dynamically
  – No model of racks / switches
  – Monitor bandwidth at runtime
  – Choose *nearest* task

• *Proactively optimize data movement, task placement together*
Map/Shuffle Bottlenecks

Shuffle, then Reduce

Map outputs

Slow shuffle links can create bottlenecks.
Shuffle-aware Map

- **Key idea:** do not assign work to mappers that will slow shuffle

- Estimate time $T_m$ for mapper $m$ to finish task
  - Push, map, **and shuffle**
  - Include *accumulated* map outputs
  - Dynamic, based on history, network monitoring

- Refuse work to possible bottleneck mappers
  - Refuse if $T_m > \min_m T_m + \alpha$
  - Large $\alpha \rightarrow$ traditional Hadoop
Benefit of Map-aware Push

- Two PlanetLab data sources (EU, US)
- Four map/reduce workers (2 EU, 2 US)

≈21% reduction in time for push & map
Benefit of Shuffle-aware Map

InvertedIndex on PlanetLab

Worse push & map for better shuffle & reduce

Makespan (s)

Scheduling Approach

Hadoop Default

Shuffle-aware Map

Reduce

Overlapped Push/Map
End-to-end Performance

InvertedIndex on PlanetLab

Push-aware Map, Map-aware Shuffle compose

Makespan (s)

Hadoop Default

End-to-end

Scheduling Approach

<table>
<thead>
<tr>
<th>Scheduling Approach</th>
<th>Hadoop Default</th>
<th>End-to-end</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Push
- Overlapped Push/Map
- Map
- Reduce
Concluding Remarks

• Geographically distributed data, resources
• Many applications fit MapReduce
• Optimizing for highly-distributed environments:
  – Consider multiple phases together
  – Minimize end-to-end execution time
• Acknowledgments:
  – Students: Ben Heintz, Chenyu Wang
  – Ramesh Sitaraman (UMASS), Jon Weissman (UMN)
  – NSF support
Thank You!

http://www.cs.umn.edu/~chandra