Introduction to Data Mining

Lecture #4: MapReduce-2

U Kang
Seoul National University

Outline

- ☐ **Problems Suited For Map-Reduce**
- ☐ Pointers and Further Reading

Example: Host size

- **Suppose we have a large web corpus**
- Look at the metadata file
  - Lines of the form: (URL, size, date, …)
- **For each host, find the total number of bytes**
  - That is, the sum of the page sizes for all URLs from that particular host

Example: Language Model

- **Statistical machine translation:**
  - Need to count number of times every 5-word sequence occurs in a large corpus of documents

- **Very easy with MapReduce:**
  - **Map:**
    - Extract (5-word sequence, count) from document
  - **Reduce:**
    - Combine the counts

More Examples

- Distributed Grep
  - Map() : emits a line if it matches a supplied pattern

- Reverse Web-Link graph
  - Map() : output <target, source> for each target in a source web page
  - Reduce: output <target, list(source)>

More Examples

- Term-Vector per Host
  - Term vector : summarizes most important words that occur in a given host
  - Map: output <hostname, term vector> for a given document
  - Reduce: output <hostname, term vector> for frequent terms

More Examples

- Inverted index
  - Map(): output <word, document ID>
  - Reduce(): output <word, list(document ID)>

Example: Join By Map-Reduce

- **Compute the natural join *R(A,B)* ⋈ *S(B,C)***
- *R* and *S* are each stored in files
- Tuples are pairs *(a,b)* or *(b,c)*

R (Enrollment)

A (Student id)	B (Course id)
a_1	b_1
a_2	b_1
a_3	b_2
a_4	b_3

⋈

S (Course Info.)

B (Course id)	C (Course name)
b_2	c_1
b_2	c_2
b_3	c_3

=

A (Student id)	C (Course name)
a_3	c_1
a_3	c_2
a_4	c_3

Map-Reduce Join

- **Use a hash function *h* from B-values to *1...k***
- **A Map process turns:**
  - Each input tuple *R(a,b)* into key-value pair *(b,(a,R))*
  - Each input tuple *S(b,c)* into *(b,(c,S))*

- **Map processes** send each key-value pair with key *b* to Reduce process *h(b)*
  - Hadoop does this automatically; just tell it what *k* is.
- Each **Reduce process** matches all the pairs *(b,(a,R))* with all *(b,(c,S))* and outputs *(a,b,c)*.

Cost Measures for Algorithms

- **In MapReduce we quantify the cost of an algorithm using**

1. *Communication cost* = total I/O of all processes
2. *Elapsed communication cost* = max of I/O along any path
3. (*Elapsed*) *computation cost* analogous, but count only running time of processes

Example: Cost Measures

- **For a map-reduce algorithm:**
  - **Communication cost =** input file size + 2 × (sum of the sizes of all files passed from Map processes to Reduce processes) + the sum of the output sizes of the Reduce processes.
  - **Elapsed communication cost** is the sum of the largest input + output for any map process, plus the same for any reduce process

What Cost Measures Mean

- Either the I/O (communication) or processing (computation) cost dominates
  - Ignore one or the other

- Total cost tells what you pay in rent from your friendly neighborhood cloud

- Elapsed cost is wall-clock time using parallelism

Cost of Map-Reduce Join

- **Total communication cost** $= O(|R|+|S|+|R \bowtie S|)$
- **Elapsed communication cost** $= O(s)$
  - We're going to pick ***k*** (# of reducers) and the number of Map processes so that the I/O limit ***s*** is respected
  - We put a limit ***s*** on the amount of input or output that any one process can have. ***s* could be:**
    - What fits in main memory
    - What fits on local disk
- In many cases, computation cost is linear in the input + output size
  - So computation cost is like comm. cost

Outline

- ☑ Problems Suited For Map-Reduce
- ☐ **Pointers and Further Reading**

Implementations

- Google
  - Not available outside Google
- **Hadoop**
  - An open-source implementation in Java
  - Uses HDFS for stable storage
  - Download: http://lucene.apache.org/hadoop/

Cloud Computing

- Ability to rent computing by the hour
  - Additional services e.g., persistent storage
- Amazon's "Elastic Compute Cloud" (EC2)

Reading

- Jeffrey Dean and Sanjay Ghemawat: MapReduce: Simplified Data Processing on Large Clusters
  - http://static.googleusercontent.com/media/research.google.com/ko//archive/mapreduce-osdi04.pdf
  - Must Read!
- Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung: The Google File System
  - http://static.googleusercontent.com/media/research.google.com/ko//archive/gfs-sosp2003.pdf

Resources

- Hadoop Wiki
  - Introduction
    - http://wiki.apache.org/lucene-hadoop/
  - Getting Started
    - http://wiki.apache.org/lucene-hadoop/GettingStartedWithHadoop
  - Map/Reduce Overview
    - http://wiki.apache.org/lucene-hadoop/HadoopMapReduce
    - http://wiki.apache.org/lucene-hadoop/HadoopMapRedClasses
  - Eclipse Environment
    - http://wiki.apache.org/lucene-hadoop/EclipseEnvironment
- Javadoc
  - http://lucene.apache.org/hadoop/docs/api/

Resources

- Releases from Apache download mirrors
  - http://www.apache.org/dyn/closer.cgi/lucene/hadoop/
- Nightly builds of source
  - http://people.apache.org/dist/lucene/hadoop/nightly/
- Source code from subversion
  - http://lucene.apache.org/hadoop/version_control.html

Further Reading

- Programming model inspired by functional language primitives
- Partitioning/shuffling similar to many large-scale sorting systems
  - NOW-Sort ['97]
- Re-execution for fault tolerance
  - BAD-FS ['04] and TACC ['97]
- Locality optimization has parallels with Active Disks/Diamond work
  - Active Disks ['01], Diamond ['04]
- Backup tasks similar to Eager Scheduling in Charlotte system
  - Charlotte ['96]
- Dynamic load balancing solves similar problem as River's distributed queues
  - River ['99]

Questions?