CS246 Introduction

Essence of the Course
Administrivia
MapReduce
Other Parallel-Computing Systems

Jeffrey D. Ullman
Stanford University

What This Course Is About

- *Data mining* = extraction of actionable information from (usually) very large datasets, is the subject of extreme hype, fear, and interest.
- It's not all about machine learning.
- But some of it is.
- Emphasis in CS246 on algorithms that scale.
  - Parallelization often essential.

Models

- Often, especially for ML-type algorithms, the result is a *model* = a simple representation of the data.
- Example: PageRank is a number Google assigns to each Web page, representing the "importance" of the page.
  - Calculated from the link structure of the Web.
  - Summarizes in one number, all the links leading to one page.
  - Used to help decide which pages Google shows you.

“The Two Cultures”

1. Build a model of the data; answer questions from the model.
   - The classical approach of statisticians.
   - Also used in machine-learning community.
2. Compute answers from data.
   - The computer-science/algorithmic approach.

- Example: given a set of points on a line, a Statistician would try to fit the best Gaussian to the data; a Computer Scientist would compute the average.

Algorithms Vs. Models

- Example: email spam.
- A model of spam might be based on weighted occurrences of words or phrases.
  - Would give high weight to words like “Viagra” or phrases like “Nigerian prince.”
- Problem: when the weights are in favor of spam, there is no obvious reason why it is spam.
- Sometimes, no one cares; other times understanding is vital.
  - Weird but true: EU is outlawing unexplainable models.

Algorithm to Discover Rules

- Rules like "Nigerian prince" -> spam are understandable and actionable.
- But the downside is that every email with that phrase will be considered spam.
- Next lecture will talk about these *association rules*, and how they are used in managing (brick and mortar) stores, where understanding the meaning of a rule is essential.

Administrivia

- Prerequisites
- Requirements
- Staff
- Resources
- Honor Code
- Lateness Policy

Prerequisites

- Programming. Java is best for homeworks.
- Basic Algorithms.
  - CS161 is surely sufficient.
- Probability, e.g., CS109 or Stat116.
  - There will be a review session and a review doc is linked from the class home page.
- Linear algebra.
  - Another review doc + review session is available.
- Multivariable calculus.
- Database systems (SQL, relational algebra).
  - CS145 is sufficient by not necessary.

What If I Don't Know All This Stuff?

- Each of the topics listed is important for a small part of the course.
  - If you are missing an item of background, you could consider just-in-time learning of the needed material.
- The exception is programming.
  - To do well in this course, you really need to be comfortable with writing code in Java or a similar language.

Requirements: Final Exam

- Final Exam (40%). The exam will be held Tues., March 21, 3:30-6:30PM.
  - Place TBD.
- There is no alternative final, but if you truly have a conflict, we can arrange for you to take the exam immediately after the regular final.

Requirements: Gradiance

- Gradiance on-line homework (18%).
  - This automated system lets you try questions as many times as you like, and the goal is that everyone will work until they get all problems right.
  - Sign up at www.gradiance.com/services and enter class **380CE054**
  - Note: your score is based on the most recent submission, not the max.
  - Note: After the due date, you can see the solutions to all problems by looking at one of your submissions, so you must try at least once.

Gradiance – (2)

- You should work each of the implied problems before answering the multiple-choice questions.
- That way, if you have to repeat the work to get 100%, you will have available what you need for the questions you have solved correctly, and the process can go quickly.
- Note: There is a 10-minute delay between submissions, to protect against people who randomly fire off guesses.

Requirements: Ordinary HW

- Written Homework (40%).
  - Four major assignments, involving programming, proofs, algorithm development.
  - Preceded by a “warmup” assignment, called “tutorial,” to introduce everyone to Hadoop.
  - Submission of work will be via gradescope.com and you need to use class code **MBDY2M** for CS246.

Requirements: Piazza

- 2% of your grade will be awarded for participating in Piazza discussions.
- Especially valuable are answers to questions posed by other students.

Staff

- There are 13 great TA’s this year! Naveen Arivazhagan, Rishabh Bhargava, Yixin Cai, Nihit Desai, Anthony Kim, Sachin Padmanabhan, Vinaya Polamreddi, Jessica Su, Yixin Wang, Junwei Yang, Leon Yao (chief TA), Luda Zhao, and Michael Zhu.
- See class page web.stanford.edu/class/cs246 for schedule of office hours.
- cs246-win1617-staff@lists.stanford.edu will contact TA’s + Jure + Jeff.

Resources

- Class textbook: Mining of Massive Datasets by Jure Leskovec, Anand Rajaraman, and U.
  - Sold by Cambridge Univ. Press, but available for free download at www.mmds.org
- Review notes for Probability, Proofs, and Linear Algebra are available.
- MOOC www.youtube.com/channel/UC_Oao2FYkLAUlUVkBfze4jg/videos
- Piazza discussion group (please join) at piazza.com/stanford/winter2017/cs246 (code mmds)

Special Tutorials

- The first two Fridays, we will be holding tutorials in B03 Gates at 3PM.
- Linear Algebra: Jan. 13, 2017.
- Statistics: Jan. 20, 2017.

Honor Code

- We'll follow the standard CS Dept. approach: You can get help, but you *MUST* acknowledge the help on the work you hand in.
- Failure to acknowledge your sources is a *violation of the Honor Code*.
- We use MOSS to check the originality of your code.

Honor Code – (2)

- You can talk to others about the algorithm(s) to be used to solve a homework problem.
  - As long as you then mention their name(s) on the work you submit.
- You should not use code of others or be looking at code of others when you write your own.
  - If you do so, and mention their contributions on your own homework, then it is not an HC violation, although we may deduct points.
  - If you fail to mention your sources, MOSS will catch you, and you will be charged with an HC violation.

Late Homeworks

- There is no possibility of submitting Gradiance homework late.
- For the five written homeworks, you have two *late periods*.
  - If a homework is due on a Thursday (Tuesday), then the late period expires 11:59PM on the following Tuesday (Thursday).
  - You can only use one late period per homework.

The MapReduce Environment

Distributed File Systems
MapReduce and Hadoop
Examples of MR Algorithms

Jeffrey D. Ullman
Stanford University

Distributed File System

- Designed for a *computing cluster* (large collection of loosely connected compute nodes).
- Goal: never lose any data (*resilience*).
- File is split into contiguous *chunks*, typically 64MB.
- Each chunk replicated (usually 3x).
- Master Node for a file.
  - Stores metadata, location of all chunks.
  - Must be replicated.
- Google file system was original; HDFS is used by Hadoop; Colossus now used at Google.

Alternative: Erasure Coding

- More recent approach to resilient file storage.
- Allows reconstruction of a lost chunk.
- Advantage: less redundancy for a given probability of loss.
- Disadvantage: no choice regarding where to obtain a given chunk.

MapReduce

A Quick Introduction
Word-Count Example
Fault-Tolerance

MapReduce and Hadoop

- MapReduce is a style of programming designed for:
  1. Easy parallel programming.
  2. Invisible management of hardware and software failures.
  3. Easy management of very-large-scale data.
- It has several implementations, including Hadoop (used in this class), Spark (becoming dominant), Flink, and the original Google implementation just called "MapReduce."

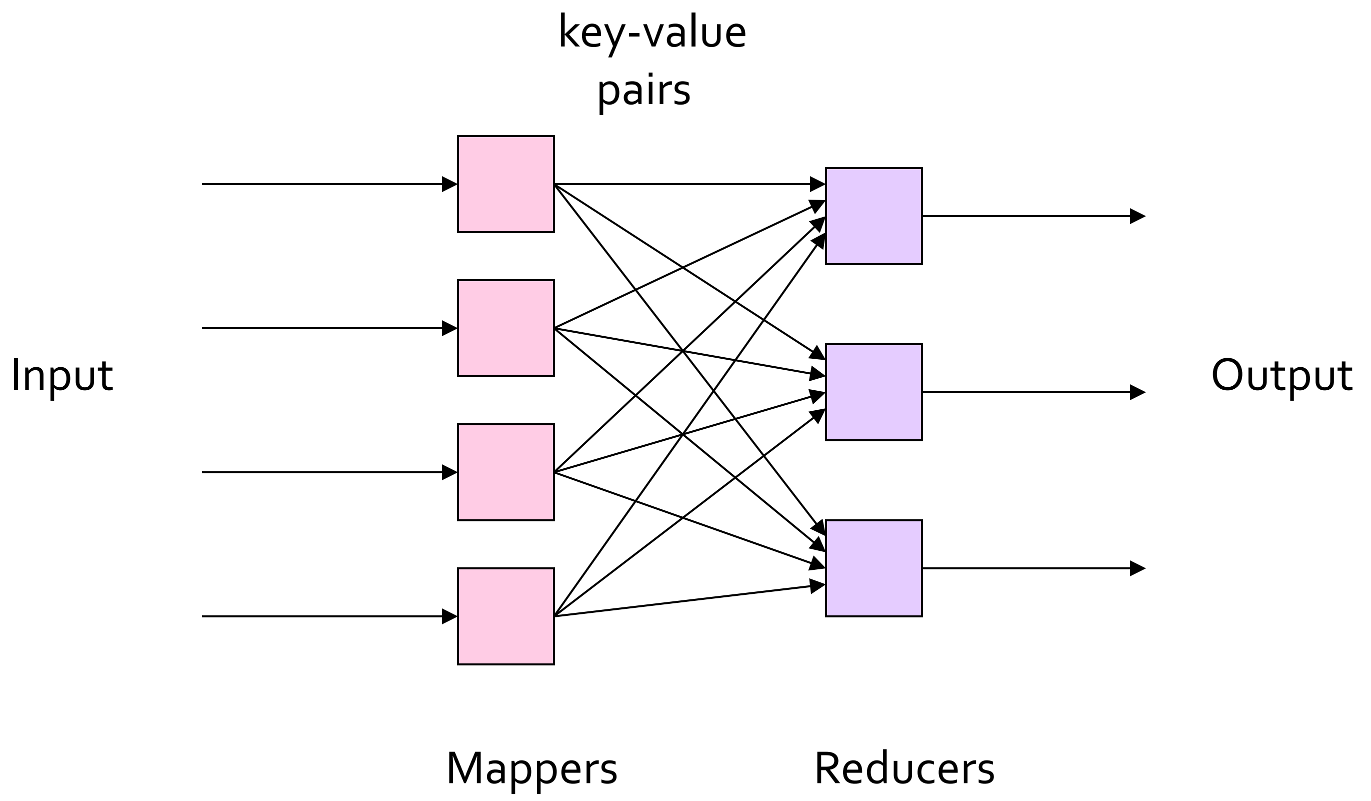
MapReduce in a Nutshell

- A MapReduce job starts with a collection of input elements of a single type.
  - Technically, all types are key-value pairs.
- Apply a user-written *Map function* to each input element, in parallel.
  - *Mapper* applies the Map function to a single element.
    - Many mappers grouped in a *Map task* (the unit of parallelism).
- The output of the Map function is a set of 0, 1, or more *key-value pairs*.
- The system sorts all the key-value pairs by key, forming key-(list of values) pairs.

In a Nutshell – (2)

- Another user-written function, the *Reduce function*, is applied to each key-(list of values).
  - Application of the Reduce function to one key and its list of values is a *reducer*.
    - Often, many reducers are grouped into a *Reduce task*.
- Each reducer produces some output, and the output of the entire job is the union of what is produced by each reducer.

MapReduce Pattern



Example: Word Count

- We have a large file of documents (the input elements).
- Documents are words separated by whitespace.
- Count the number of times each distinct word appears in the file.

Word Count Using MapReduce

```
map(key, value):
// key: document ID; value: text of document
    FOR (each word w IN value)
        emit(w, 1);

reduce(key, value-list):
// key: a word; value-list: a list of integers
        result = 0;
        FOR (each integer v on value-list)
            result += v;
        emit(key, result);
```

Expect to be all 1's, but "combiners" (Sect. 2.2.4) Allow local summing of integers with the same key before passing to reducers.

Coping With Failures

- MapReduce is designed to deal with compute nodes failing to execute a Map task or Reduce task.
- Re-execute failed tasks, not whole jobs.
- Key point: MapReduce tasks have the *blocking property*: no output is used until the task is complete.
- Thus, we can restart a Map task that failed without fear that a Reduce task has already used some output of the failed Map task.

Some MapReduce Algorithms

Relational Join

Matrix Multiplication in One or Two Rounds

Relational Join

- Stick the tuples of two relations together when they agree on common attributes (column names).
- Example: R(A,B) JOIN S(B,C) = {abc | ab is in R and bc is in S}.

A	B
1	2
3	2
4	5

JOIN

B	C
2	6
5	7
5	8
9	10

=

A	B	C
1	2	6
3	2	6
4	5	7
4	5	8

The Map Function

- Each tuple (a,b) in R is mapped to key = b, value = (R,a).
  - Note: "R" in the value is just a bit that means "this value represents a tuple in R, not S."
- Each tuple (b,c) in S is mapped to key = b, value = (S,c).
- After grouping by keys, each reducer gets a key-list that looks like

  (b, [(R,a_1), (R,a_2),…, (S,c_1), (S,c_2),…]).

The Reduce Function

- For each pair (R,a) and (S,c) on the list for key b, emit (a,b,c).
  - Note this process can produce a quadratic number of outputs as a function of the list length.
  - If you took CS245, you may recognize this algorithm as essentially a “parallel hash join.”
  - It’s a really efficient way to join relations, as long as you don’t have too many tuples with a common shared value.

Two-Pass Matrix Multiplication

- Multiply matrix $M = [m_{ij}]$ by $N = [n_{jk}]$.
  - Want: $P = [p_{ik}]$, where $p_{ik} = \Sigma_j\ m_{ij} * n_{jk}$.
- First pass is similar to relational join.
  - Computes each $m_{ij} * n_{jk}$.
- Second pass is a group + aggregate operation.
  - Computes the sum over j.
- Typically, large relations are *sparse* (mostly 0's).
- Assume a nonzero m_{ij} is really a tuple of a relation (i, j, m_{ij}); similarly for n_{jk}.
  - 0 elements are not represented at all.

The Map and Reduce Functions

- The Map function: (i,j,m_{ij}) -> key = j, value = (M,i,m_{ij}); (j,k,n_{jk}) -> key = j, value = (N,k,n_{jk}).
  - As for join, M and N here are bits indicating which relation the value comes from.
- The Reduce function: for key j, pair each (M,i,m_{ij}) on its list with each (N,k,n_{jk}) and produce key = (i,k), value = $m_{ij} * n_{jk}$.

The Second Pass

- The Map function: The identity function.
- Result is that each key (i,k) is paired with the list of products m_{ij} * n_{jk} for all j.
- The Reduce function: sum all the elements on the list, and produce key = (i,k), value = that sum.
  - I.e., each output element ((i,k),s) says that the element p_{ik} of the product matrix P is s.

Single-Pass Matrix Multiplication

- We can use a single pass if:
  1. Keys (reducers) correspond to output elements (i,k).
  2. Map sends input elements to more than one reducer.
- The Map function: m_{ij} -> for all k: key = (i,k), value = (M,j,m_{ij}); n_{jk} -> for all i: key = (i,k), value = (N,j,n_{jk}).
- The Reduce function: for each (M,j,m_{ij}) on the list for key (i,k) find the (N,j,n_{jk}) with the same j. Multiply m_{ij} by n_{jk} and then sum the products.

Extensions to MapReduce

Data-Flow Systems
Bulk-Synchronous Systems
Tyranny of Communication

Data-Flow Systems

- MapReduce uses two ranks of tasks: one for Map the second for Reduce.
  - Data flows from the first rank to the second.
- Generalize in two ways:
  1. Allow any number of ranks.
  2. Allow functions other than Map and Reduce.
- As long as data flow is in one direction only, we can have the blocking property and allow recovery of tasks rather than whole jobs.

Spark

- The most popular implementation of a dataflow system.
- Data passed from one rank of processes to the next forms a *Resilient Distributed Dataset* (RDD).
  - Elements of an RDD are like tuples of a relation.
    - Generalizes (key-value) pairs.
- Built-in operations include Map, Reduce, and also group-aggregate using any components of the RDD type.
  - The thing Hadoop does behind-the-scenes.

Example: 2-Pass MatMult

- The 2-pass MapReduce algorithm had a second Map function that didn’t really do anything.
- We could think of it as a five-rank data-flow algorithm of the form Map-GA-Reduce-GA-Reduce, where the RDD types are:
  1. (j, (M,i,m_{ij})) and (j, (N,k,n_{jk})).
  2. j with list of (M,i,m_{ij})’s and (N,k,n_{jk})’s.
  3. ((i,k), $m_{ij} * n_{jk}$).
  4. (i,k) with list of $m_{ij} * n_{jk}$’s.
  5. ((i,k), p_{ik}).

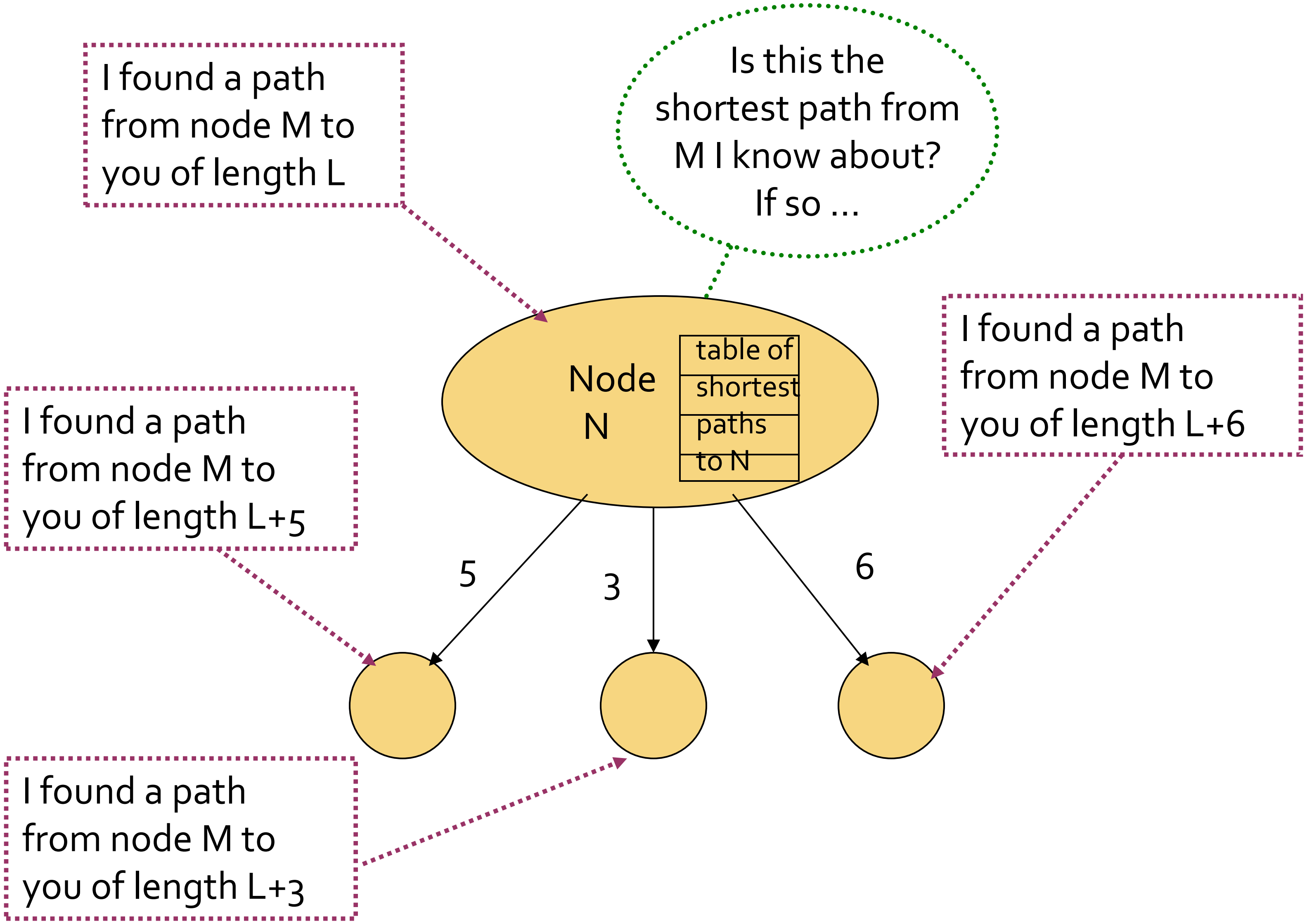
Bulk-Synchronous Systems

Graph Model of Data
Some Systems Using This Model

The Graph Model

- Computation is a recursion on some graph.
- Graph nodes send messages to one another.
  - Messages bunched into *supersteps*, where each graph node processes all data received.
  - Sending individual messages would result in far too much overhead.
- Checkpoint all compute nodes after some fixed number of supersteps.
  - Note blocking property fails to hold.
- On failure, roll all tasks back to previous checkpoint.

Example: Shortest Paths


I found a path from node M to you of length L
Is this the shortest path from M I know about? If so …
Node N
table of shortest paths to N
I found a path from node M to you of length L+6
I found a path from node M to you of length L+5
5
3
6
I found a path from node M to you of length L+3

Some Systems

- *Pregel*: the original, from Google.
- *Giraph*: open-source (Apache) Pregel.
  - Built on Hadoop.
- *GraphX*: a similar front end for Spark.
- *GraphLab*: similar system that deals more effectively with nodes of high degree.
  - Will split the work for such a graph node among several compute nodes.