PART 1. BATCH COMPUTING MODEL FOR BIG DATA ANALYTICS

2. DISTRIBUTED BATCH COMPUTING FRAMEWORKS: MAPREDUCE

Sangmi Lee Pallickara
Computer Science, Colorado State University
http://www.cs.colostate.edu/~cs535

FAQs

- Programming Assignment 1
  - We will discuss link analysis today and week 3
  - Installation/configuration guidelines for Hadoop and Spark are uploaded
  - Port assignment has been posted

- TP teams
  - Any change? Please let me know

Today’s topics

- YARN
- MapReduce with examples

YARN (MapReduce 2)

- To provide the scalability to MapReduce
  - Splitting responsibility of the jobtracker
    - Scheduling
    - Task progress monitoring

- MapReduce is one type of YARN application

YARN (MapReduce 2)

- Resource manager
  - Manages the use of resources across the cluster
- Node manager
  - Launches and monitors the compute containers on machines in the cluster
- Application master
  - Manages the lifecycle of applications running on the cluster
  - Application master negotiates with the resource manager for cluster resources
  - Number of container and certain memory limit
  - Node managers oversees containers not to use more resources than allocated

A MapReduce job using YARN
Task assignment [1/2]

- Application Master requests container for all the map and reduce tasks in the job
  - From the resource manager (Step 8)

- All the requests includes information about each map tasks’ locality
  - Host and corresponding racks that the input split resides on

- Scheduler attempts to place tasks on data-local nodes in the ideal case
  - If it is not possible, the scheduler prefers rack-local placement
  - Job is running on a node in the same rack

What is a small job?
A small job is one that has less than 10 mappers, only one reducer, and an input size that is less than the size of one HDFS block

Task assignment [2/2]

- Requests specify required memory
  - 1024MB (by default)
    - This is configurable
      - mapreduce.map.memory.mb
      - mapreduce.reduce.memory.mb

- In YARN, resources are managed more fine-grained
  - Applications may request a memory capability that is anywhere between the minimum allocation and a maximum allocation
    - yarn.scheduler.capacity.minimized-allocation-mb
    - yarn.scheduler.capacity.maximum-allocation-mb
    - Default minimum: 1024MB
    - Default maximum: 10240 MB
    - Tasks can request any memory allocation between 1 and 10GB (default) in multiple of 1GB
      - mapreduce.map.memory.mb and mapreduce.reduce.memory.mb

Task execution [1/2]

- Application master starts the container by contacting node manager
  - The task is executed by YarnChild
  - YarnChild runs in a dedicated JVM

Progress and status updates [2/3]

- Task reports its progress and status back to its application master
  - Every 3 seconds over the umbilical interface

- The client polls the application master every second
  - mapreduce.client.progressmonitor.pollinterval

Failures [3/3]

 Failures observed in Google Data Centers

“In each cluster’s first year, it’s typical that 1,000 individual machine failures will occur; thousands of hard drive failures will occur; one power distribution unit will fail, bringing down 500 to 1,000 machines for about 6 hours; 20 racks will fail, each time causing 40 to 80 machines to vanish from the network; 5 racks will “go wonky,” with half their network packets missing in action; and the cluster will have to be rewired once, affecting 5 percent of the machines at any given moment over a 2-day span. And there’s about a 50 percent chance that the cluster will overheat, taking down most of the servers in less than 5 minutes and taking 1 to 2 days to recover.”

Task failure in YARN

- Failure of the running task is similar to the classic case
  - Runtime exception and sudden exit of the JVM are propagated back to the application master
  - The task attempt is marked as failed
  - Hanging tasks are noticed by the application manager by the absence of a ping over the umbilical channel

Application master failure in YARN

- No heartbeats to the resource manager from the application master
  - The resource manager will detect the failure and start a new instance of the application master running in a new container
  - All tasks will be rerun (default)
  - Recovery can be enabled
  - Client will access resource manager to get the new address of the application master

Node manager failure in YARN

- Resource manager will stop getting heartbeats
  - Remove the failed node manager from the pool of available nodes

Resource Manager fails in YARN

- Active/Standby architecture
  - Stores state to file or ZooKeeper

1. Active RM writes its states into ZooKeeper
2. Fail-over if the Active RM fails
   - Fail-over can be done by auto/manual

MapReduce Programming with Examples
Example-1: Selecting rows

- Suppose that you have a fileset with a 1TB dataset

From: ~cs535 samplefile.txt

Retrieve all of the lines where “from” and “to” are the same URL

Example-2: Extracting information

- Suppose that you have a file set with 1TB dataset

From: ~cs535 samplefile.txt

Retrieve all unique “from”s

Example-3: Combining datasets

- Suppose that you have two tables with 1TB each

From: ~cs535 A.txt and B.txt

If the tuples agree on an attribute that are common to the two schemas, then produce a new tuple that has components for each of the attributes in either schema

- Equivalent to SQL’s “JOIN” operation
- second url in the A.txt and first attribute in B.txt are common
- JOIN(A, B) = \{127.0.0.1, 192.168.10.102, Alive\}, \{127.0.0.1, 192.168.10.102, Down\}
Example 3: Combining datasets using MapReduce

Map function:
For each line (a, b) in A, produce the key-value pair (b, ("A", a))
For each tuple (b, c) in B, produce the key-value pair (b, ("B", c))

Reduce function:
Construct all pairs comprising one with first component "A" and the other with first component "B". e.g. ("A", a) and ("B", c)
Produce tuple (a, b, c) such that ("A", a) and ("B", c)

This material is built based on,
  - Chapter 5
- http://infolab.stanford.edu/~ullman/mmds.html
**PageRank**

- **Goals**
  - Providing effective summaries for the search results
  - Ordering/Ranking results

- Simulate random Web surfers
  - Pages that would have a large number of surfers were considered more “important” than pages that would rarely be visited
  - The content of a page was judged not only by the terms appearing on that page
    - But by the terms used in or near the links to that page

---

**Definition of PageRank**

A function that assigns a real number to each page in the Web

- The higher the PageRank of a page, the more “important” it is
- There is NOT one fixed algorithm for assignment of PageRank

---

**Example**

- Page A has links to B, C and D
- Page B has links to A and D
- Page C has a link to A
- Page D has links to B and C

---

**Example**

- Suppose that a random surfer starts at page A
  - Page B, C and D will be the next with probability 1/3
  - 0 probability of being at A
**Example [4/5]**

- Transition matrix $M$
  - What happens to random surfers after one step
  - $M$ has $n$ rows and columns
  - What is the transition matrix for this example?

```
Example [5/5]
```

**Example [5/5]**

- The first column
  - A surfer at A has a 1/3 probability of next being at each of the other pages
- The second column
  - A surfer at B has a ½ probability of being next at A and the same for being at D

---

**What does this matrix mean? [1/6]**

- The probability distribution for the location of a random surfer
  - A column vector whose $j$th component is the probability that the surfer is at page $j$

```
What does this matrix mean? [2/6]
```

**What does this matrix mean? [2/6]**

- If we surf at any of the $n$ pages of the Web with equal probability
  - The initial vector $v_0$ will have $1/n$ for each component
  - If $M$ is the transition matrix of the Web
    - After the first one step, the distribution of the surfer will be $Mv_0$
    - After two steps, $M^2v_0$ and so on

```
What does this matrix mean? [3/6]
```

**What does this matrix mean? [3/6]**

- Multiplying the initial vector $v_0$ by $M$ a total of $i$ times
  - The distribution of the surfer after $i$ steps
    - The probability for the next step from the current location
    - The probability for being in the current location

```
What does this matrix mean? [4/6]
```

**What does this matrix mean? [4/6]**

- The probability $x_i$ that a random surfer will be at node $i$ at the next step
  - $x_i = \sum_{j} M_{ij} v_j$
  - $M_{ij}$ is the probability that a surfer at node $j$ will move to node $i$ at the next step
  - $v_j$ is the probability that the surfer was at node $j$ at the previous step
What does this matrix mean? [5/6]

- The distribution of the surfer approaches a limiting distribution \( v \) that satisfies \( v = Mv \) provided two conditions are met:
  1. The graph is strongly connected
     - It is possible to get from any node to any other node
  2. There are no dead ends
     - Nodes that have no arcs out

What does this matrix mean? [6/6]

- The limit is reached when multiplying the distribution by \( M \) another time does not change the distribution
- The limiting \( v \) is an eigenvector of \( M \)
- Since \( M \) is stochastic (its columns each add up to 1), \( v \) is the principle eigenvector
- Its associated eigenvalue is the largest of all eigenvalues
- The principle eigenvector of \( M \)
  - Where the surfer is most likely to be after a long time
- For the Web, 50-75 iterations are sufficient to converge to within the error limits of double-precision arithmetic

Example

\[
M = \begin{bmatrix}
0 & 1/2 & 1 & 0 \\
1/3 & 0 & 0 & 1/2 \\
1/3 & 0 & 0 & 1/2 \\
1/3 & 1/2 & 0 & 0
\end{bmatrix}
\]

- Suppose we apply this process to the matrix \( M \)
  The initial vector \( v_0 \) and \( v_1 \) after multiplying \( M \)

\[
v_1 = Mv_0 = \begin{bmatrix}
1/3 \\
1/3 \\
1/3 \\
1/2
\end{bmatrix}
\]

Example continued

- The sequence of approximations to the limit
  - We get by multiplying at each step by \( M \) is

\[
\begin{bmatrix}
1/3 & 1/3 & 1/3 & 1/3 \\
1/3 & 1/3 & 1/3 & 1/3 \\
1/3 & 1/3 & 1/3 & 1/3 \\
1/2 & 1/2 & 1/2 & 1/2
\end{bmatrix}
\]

- This difference in probability is not noticeable
- In the real Web, there are billions of nodes of greatly varying importance
- The probability of being at a node like www.amazon.com is orders of magnitude greater than others
Problems we need to avoid

- Dead end
  - A page that has no links out
  - Surfers reaching such a page will disappear
  - In the limit, no page that can reach a dead end can have any PageRank at all

- Spider traps
  - Groups of pages that all have outlinks but they never link to any other pages

Avoiding Dead Ends

- If we allow dead ends
  - The transition matrix of the Web is no longer stochastic
  - Some of the columns will sum to 0 rather than 1

- If we compute \( M^v \) for increasing powers of a substochastic matrix
  - Some of all of the components of the vector go to 0
    - substochastic matrix
      - A matrix whose column sums are at most 1
      - Importance "drains out" of the Web
      - No information about the relative importance of pages

- Remove the arc from C to A
  - C becomes a dead end
  - If a random surfer reaches C, they disappear at the next round

- Repeatedly multiplying the vector by \( M \):
  - The probability of a surfer being anywhere goes to 0 as the number of steps increase
Approaches to dealing with dead ends

- **Recursive deletion**
  - Drop the dead ends from the graph
  - Drop their incoming arcs as well
  - Doing so may create more dead ends
  - Drop those new dead ends

- **Taxation**
  - Modify the process by which random surfers are assumed to move about the Web

Example of recursive deletion (1/4)

- The final matrix for the graph is

\[
M = \begin{bmatrix}
0 & 0 & 0 \\
\frac{1}{2} & 0 & 1 \\
0 & \frac{1}{2} & 0 \\
\end{bmatrix}
\]

Example of recursive deletion (2/4)

\[
\begin{array}{cccc}
1/1 & 6/12 & 5/24 & 2/36 \\
1/3 & 2/6 & 5/12 & 11/24 \\
1/3 & 2/6 & 4/12 & 8/24 \\
\end{array}
\]

Example of recursive deletion (3/4)

- We still need to compute deleted nodes (C and E)
  - C was the last to be deleted
    - We know all its predecessors have PageRanks (A and D)
    - Therefore,
      - \( \text{PageRank of } C = \frac{1}{3} \times \frac{2}{9} + \frac{1}{2} \times \frac{3}{9} = \frac{13}{54} \)

Example of recursive deletion (4/4)

- Now, we can compute the PageRank for E
  - Only one predecessor, C
  - The PageRank of E is the same as that of C (13/54)

- The sums of the PageRanks exceeds 1
  - It cannot represent the distribution of a random surfer
  - It provides a good estimate

**Link Analysis**

*Challenges in PageRank Algorithm for the real Web*

(2) *Spider Traps*
Spider Traps and Taxation

- Spider traps
  - A set of nodes with no dead ends but no arcs out (from the set of nodes)
  - This can appear intentionally or unintentionally on the Web

Spider traps causes the PageRank calculation to place all the PageRank within the spider traps

Example 1

- There is a simple spider trap of 4 nodes

\[
M = \begin{bmatrix}
0 & 1/2 & 0 & 0 \\
1/3 & 0 & 0 & 1/2 \\
1/3 & 1/2 & 0 & 1/2 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1/2 \\
1/3 & 0 & 1/2 & 0 \\
1/3 & 1/2 & 0 & 0
\end{bmatrix}
\]

If we perform the usual iteration to compute the PageRank of the nodes, we get

\[
\begin{align*}
1/6 & \quad 1/12 \\
1/6 & \quad 5/36 \\
1/6 & \quad 11/56 \\
1/6 & \quad 4/18 \\
1/6 & \quad 1/12 \\
1/6 & \quad 0 \\
1/6 & \quad 0
\end{align*}
\]

Example 2

- The arc out of C changed to point to C itself

\[
M = \begin{bmatrix}
0 & 1/2 & 0 & 0 \\
1/3 & 0 & 0 & 1/2 \\
1/3 & 1/2 & 0 & 1/2 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1/2 \\
1/3 & 1 & 1/2 \\
1/3 & 1/2 & 0 & 0
\end{bmatrix}
\]

If we perform the usual iteration to compute the PageRank of the nodes, we get

\[
\begin{align*}
1/4 & \quad 5/24 \\
1/4 & \quad 11/24 \\
1/4 & \quad 5/24 \\
1/4 & \quad 0
\end{align*}
\]

To avoid Spider traps, we modify the calculation of PageRank

- All the PageRank is at C
- Once there, a random surfer can never leave
- Allowing each random surfer a small probability of teleporting to a random page
- Rather than following an out-link from their current page
The iterative step, where we compute a new vector estimate of PageRanks \( v' \) from the current PageRank estimate \( v \) and the transition matrix \( M \) is

\[
v' = \beta Mv + (1 - \beta) \frac{e}{n}
\]

- Where \( \beta \) is a chosen constant
  - Usually in the range 0.8 to 0.9

\( v \) is a vector for all 1's with the appropriate number of components

\( e \) is a vector for all 1's with the appropriate number of components

\( n \) is the number of nodes in the Web graph

If the graph has no dead ends
- The probability of introducing a new random surfer is exactly equal to the probability that the random surfer will decide not to follow a link from their current page
- Surfer decides either to follow a link or teleport to a random page

Example of Taxation (1/2)

\[
M = \begin{bmatrix}
0 & 1/2 & 0 & 0 \\
1/3 & 0 & 0 & 1/2 \\
1/3 & 0 & 1/2 & 0 \\
1/3 & 1/2 & 0 & 0
\end{bmatrix}
\]

\[
v' = \beta Mv + (1 - \beta) \frac{e}{n}
\]

\( \beta = 0.8 \)

Example of Taxation (2/2)

- First few iterations:

<table>
<thead>
<tr>
<th>Iteration</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.6</td>
<td>0.4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0.36</td>
<td>0.64</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0.25</td>
<td>0.75</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

- By being a spider trap, C gets more than 1/2 of the PageRank for itself
- Effect is limited
- Each of the nodes gets some of the PageRank

If the graph has dead ends
- The surfer goes nowhere
- The term \( (1 - \beta) \frac{e}{n} \) does not depend on the sum of the components of the vector \( v \), there will be some fraction of a surfer operating on Web
- When there are dead ends, the sum of the components of \( v \) may be less than 1
  - But it will never reach 0
Link Analysis

Examples

Example 1
- Compute the PageRank of each page assuming $\beta=0.8$

$$M = \begin{bmatrix} 0.3 & 0.3 & 0.2 \\ 0.5 & 0.2 & 0 \\ 0.5 & 0.3 & 0 \end{bmatrix}$$

$$v_0 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

$$\beta M v_0 + (1-\beta)e/n = 0.8 \times \begin{bmatrix} 0.3 \\ 0.5 \\ 0.5 \end{bmatrix} + (1-0.8)e/3$$

Example 2
- Determine the PageRank of each page assuming $\beta=0.8$
- Is there a dead end?
- Yes, E
- Is there a spider trap?
- No. Having clique does not mean that there is a spider trap
Example 3
- Suppose that we recursively eliminate dead ends from the Web graph to solve the remaining graph.
- Suppose that the graph is a chain of dead ends, headed by a node with a self-loop.
- What would be the PageRank assigned to each of the nodes?

\[ A \rightarrow B \rightarrow C \rightarrow D \rightarrow Z \]

Example of Taxation (1/2)

\[ M = \begin{bmatrix} 0 & 1/2 & 0 & 0 \\ 0 & 0 & 1/2 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 1/2 & 0 & 0 \end{bmatrix} \]

\[ v' = \beta M v + (1-\beta) \frac{e}{n} \]

\[ \beta = 0.8 \]

\[ v' = \begin{bmatrix} 0 & 2.5 & 0 & 0 \\ 4/5 & 0 & 0 & 2.5 \\ 4/5 & 2.5 & 0 & 0 \end{bmatrix} \]

Example of Taxation (2/2)

For the first few iterations:

\[ \begin{array}{c|c|c|c|c|c} \hline M & 0.96 & 0.936 & 0.936 & 0.936 \hline 1/3 & 5.1/300 & 70/74500 & 70/74500 & 15/140 \hline 2/3 & 53/300 & 70/74500 & 70/74500 & 19/140 \hline \end{array} \]
Hyperlink-Induced Topic Search (HITS)