Parallelism in the WordCount topology

- In our example, we have NOT used any of Storm's parallelism
  - Default setting is a factor of one
- Topology execution flow

Adding workers to a topology

- Through configuration
- Through APIs
  - Passing Config object to the submitTopology() method
- Bolts and spouts do not have to change

```java
Config config = new Config();
config.setNumWorkers(2);
```

Adding executors and tasks

- Specify the number of executors when defining a stream grouping

```java
builder.setSpout(SENTENCE_SPOUT_ID, spout, 2);
```
  - Assigns two tasks and each task is assigned its own executor thread

Two spout tasks (if we are using one worker)
In `SplitSentenceBolt` and `WordCountBolt`,
- Set up the split sentence bolt to execute as 4 tasks and 2 executors
  - Parallelism hint
    - Storm will run 2 tasks per executor (thread)
    - Each executor thread will be assigned two tasks to execute
    - `builder.setBolt(SPLIT_BOLT_ID, splitBolt, 2)`
      - `setNumTasks(4)`
      - `shuffleGrouping(SENTENCE_SPOUT_ID)`
- How many workers will work for this example?
  - **Answer:** 2 workers (JVMs)
  - 2 executors per worker

In `SplitSentenceBolt` and `WordCountBolt`,
- Set up the Word count bolt to execute as 4 tasks each with its own executor thread
  - Parallelism hint
    - Storm will run 2 tasks per executor (thread)
    - Each executor thread will be assigned two tasks to execute
    - `builder.setBolt(COUNT_BOLT_ID, countBolt, 4)`
      - `fieldsGrouping(SPLIT_BOLT_ID, new Fields("word"))`
- How many workers will work for this example?
  - **Answer:** 1 workers (JVMs)
  - This code already specified the number of workers as 2

---

5. Scalable Distributed File Systems: Google File System I and II

**Google File System II Colossus**

**Terminology**

---

http://www.cs.colostate.edu/~cs535
Properties of Reed-Solomon codes

- RS(n,k) with s-bit symbols
  - Encoder takes k data symbols (blocks) of s bits each
  - Adds parity symbols to make n symbol code word
  - There are n-k parity symbols of s bits each
  - A Reed-Solomon decoder can correct up to t symbols that contain errors in a code word, where
    $2t = n-k$
    $t = (n-k)/2$ for n-k even
    $t = (n-k-1)/2$ for n-k odd

Example

- RS(255,223) with 8 bit symbols
  - Each code word contains 255 code word bytes
  - 223 bytes are data and 32 bytes are parity
  - $n=255$, $k=223$, $s=8$, $2t = 32$, $t=16$
  - The decoder can correct any 16 symbol errors in the code word
  - Errors in up to 16 bytes anywhere in the codeword can be automatically corrected.

The Maximum codeword length

- For a symbol size s
  - The maximum codeword length n is
    $n = 2^s-1$
  - For example, the max length of a code with 8-bit symbols (s=8) is 255 bytes

Coding Gain

- The probability of an error remaining in the decoded data is (usually) much lower than the probability of an error if Reed-Solomon is not used

Example

- A digital communication system is designed to operate at a Bit Error Ratio (BER) of $10^{-9}$
  - No more than 1 in $10^9$ bits are received in error
  - This can be achieved by boosting the power of the transmitter or by adding Reed-Solomon (or another type of Forward Error Correction)
  - Reed-Solomon allows the system to achieve this target BER with a lower transmitter output power
  - The power "saving" given by Reed-Solomon (in decibels) is the coding gain.
GEAR Workshop I | Advanced Big Data Analytics Case Study
Workshop 1 Advanced Big Data Analytics Case Study
Workshop 2 Scalable Computing Models
Workshop 3 Large Scale Graph Analysis
Workshop 4 Scalable Data Storage, retrievals and analytics

Workshop 1: Topics for Lectures
- Recommendation Systems
- Anomaly Detection in Network Traffic with K-mean clustering
- Predicting Forest Cover with Decision Trees
- Understanding Wikipedia with Latent Semantic Analysis

Workshop 1: Readings for workshop

This material is built based on
- Sandy Ryza, Uri Laserson, Sean Owen, and Josh Wills, Advanced Analytics with Spark, O'Reilly, 2015

http://www.cs.colostate.edu/~cs535
“What percentage of the top 10,000 titles in any online media store (Netflix, iTunes, Amazon, or any other) will rent or sell at least once a month?”

The long tail phenomenon

Distribution of numbers with a large number of occurrences far from the “head” or central part of the distribution
- The vertical axis represents popularity
- The items are ordered on the horizontal axis according to their popularity
- The long tail phenomenon forces online institutions to recommend items to individual users


Recommendation systems

- Seek to predict the “rating” or “preference” that a user would give to an item

Applications of Recommendation Systems

- Product recommendations
  - Amazon or similar online vendors
- Movie recommendations
  - Netflix offers its customers recommendations of movies they might like
- News articles
  - News services have attempted to identify articles of interest to readers based on the articles that they have read in the past
- Blogs, YouTube

Amazon.com: Item-to-item collaborative filtering

http://www.cs.colostate.edu/~cs535
This material is built based on,

- Greg Linden, Brent Smith, and Jeremy York, “Amazon.com Recommendations, Item-to-Item Collaborative Filtering” IEEE Internet Computing, 2003

Amazon.com uses recommendations as a targeted marketing tool
- Email campaigns
- Most of their web pages

Item-to-item collaborative filtering
- It does NOT match the user to similar customers
- Item-to-item collaborative filtering
  - Matches each of the user’s purchased and rated items to similar items
  - Combines those similar items into a recommendation list

Determining the most-similar match
- The algorithm builds a similar-items table
  - By finding items that customers tend to purchase together
- How about building a product-to-product matrix by iterating through all item pairs and computing a similarity metric for each pair?
  - Many product pairs have no common customer
    - If you already bought a TV today, will you buy another TV again today?

Creating a similar-item table
- Similar-items table is extremely computing intensive
  - Offline computation
  - \( O(N^2M) \) in the worst case
    - Where \( N \) is the number of items and \( M \) is the number of users
  - Average case is closer to \( O(NM) \)
    - Most customers have very few purchases
    - Sampling customers who purchase best-selling titles reduces runtime even more
      - With little reduction in quality

- Calculating the similarity between a single product and all related products:

  For each item in product catalog, I1
  For each customer C who purchased I1
    For each item I2 purchased by customer C
      Record that a customer purchased I1 and I2
  For each item I2
    Compute the similarity between I1 and I2
Computing similarity

**Option 1.** Using co-occurrence matrix
- If an item has been purchased by the same user together many times, it is considered as a “similar” item.

**Option 2.** Using cosine measure
- Each vector corresponds to an item rather than a customer.
- $M$ dimensions correspond to customers who have purchased that item.
- 
  \[ \text{Cosine Similarity}(A,B) = \cos(A,B) = \frac{A \cdot B}{\|A\| \|B\|} \]

Example

**Co-occurrence matrix**

<table>
<thead>
<tr>
<th></th>
<th>b1</th>
<th>b2</th>
<th>b3</th>
<th>b4</th>
<th>b5</th>
<th>b6</th>
</tr>
</thead>
<tbody>
<tr>
<td>b1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>b2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>b3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>b4</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>b5</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>b6</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Purchase record for the user $U = \{b_1, b_2, b_3\}$
Purchase record for the user $U = \{b_4, b_5\}$
Purchase record for the user $U = \{b_6\}$

Co-occurrence matrix

<table>
<thead>
<tr>
<th></th>
<th>b1</th>
<th>b2</th>
<th>b3</th>
<th>b4</th>
<th>b5</th>
<th>b6</th>
</tr>
</thead>
<tbody>
<tr>
<td>b1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>b2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>b3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>b4</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>b5</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>b6</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Cosine similarity $(l_0, l_1)$

\[ l_0 \cdot l_1 = \frac{a \cdot b}{\|a\| \|b\|} \]

Key scalability strategy for Amazon recommendations

- Creating the expensive similar-items table offline.
- Online component
  - Looking up similar items for the user’s purchases and ratings.
  - Scales independently of the catalog size or the total number of customers.
- It is dependent only on how many titles the user has purchased or rated.

Amazon.com has around 110 million active customers (244 million total customers) and several million catalog items.

- Traditional collaborative filtering does little or no offline computation.
- Online computation scales with the number of customers and catalog items.

http://www.cs.colostate.edu/~cs535
Recommendation quality

- The algorithm recommends highly correlated similar items
- Recommendation quality is excellent
- Algorithm performs well with limited user data

Dataset

- Audioscrobbler dataset
  - 2002, Richard Jones
  - Collecting and analyzing user's songs to generate recommendation
  - Started with support for Winamp and XMMS
  - iTunes, Winamp, Windows Media Player, Foobar, iPod, Amarok, Rhythmbox, mpd, Xbox media center, Slimserver, Jnza, mpeg321, Mune, Rhapsody, YME, Soundbridge, VLC…

Netflix Prize

- The Netflix Prize challenge concerned recommender systems for movies (October, 2006)
- Netflix released a training set consisting of data from almost 500,000 customers and their ratings on 18,000 movies.
- More than 100 million ratings
- The task was to use these data to build a model to predict ratings for a hold-out set of 3 million ratings

Dataset

- Confined rating system
  - “Bob rates Coldplay 3.5 stars.”
  - Users rate music far less frequently than they play music
- Audioscrobbler dataset
  - “Bob played Coldplay track”
  - Each individual data carries less information
- Implicit feedback
  - User-artist connections are implied as a side effect of other actions

Dataset

- 141,000 unique users
- 1.6 million unique artists
- 24.2 million user's plays of artist are recorded
  - User_artist_data.txt
- On average, each user has played songs from about 171 artists (out of 1.6 M)
  - Extremely sparse dataset

http://www.cs.colostate.edu/~cs535