Finding Frequent Item Pairs
Runmin Xu, Lu Lu
Background

- **Basket**: a set of items someone bought together in one time
  - eg. \{apple, milk, coffee, orange\}

- We want to find item pairs that appear together “frequently” in baskets
  - \{a,b,c\}, \{a,b,d\}, \{a,b,e\}, \{a,b,f\}
  - \[a,b\] appears frequently!
Background

- Frequent pair
  - Given threshold \( s \), the pairs whose appearance frequency \( > s \) are called frequent pairs
Brute-force Method

- Count frequency of every possible pair

- n distinct items
  - $n(n-1)/2$ pairs
  - Space complexity: $O(n^2)$

- Suppose $10^5$ items, counts are 4-byte integers
  - $5 \times 10^9$ pairs
  - $2 \times 10^{20}$ (20 GB) memory needed
How to improve?

● If \([a, b]\) are frequent pair,
  ○ \(\text{frequency}([a,b]) > \text{threshold}\)
● Then
  ○ \(\text{frequency}(a) > \text{threshold}\)
  ○ \(\text{AND frequency}(b) > \text{threshold}\)
● Therefore, find frequent individual item first!
Find frequent items

● Read baskets and count the frequency of each individual item
  ○ Space complexity: $O(n)$

● Find the items with frequency $>$ threshold

● Split the dataset into a number of subset and count item frequencies in parallel (MapReduce)
Find frequent pairs

- **Method 1**
  - Generate a list of possible frequent pairs based on results from single count ($O(m^2)$ space)
  - For each basket, iterate through the list to check if each pair exist
  - Time complexity: $O(m^2*L*N)$, $L$ is the length of a basket, $N$ is the number of baskets
Find frequent pairs

● Method 2
  ○ For each basket, generate a list of frequent single items, then generate a list of possible frequent pairs and count
  ○ Iterate through all baskets
  ○ Time complexity: $O(L^2*N)$
  ○ $L$ is usually much smaller than $m^2$
Parallelization
Dataset

- 999,002 transactions
- 41,270 distinct items
Parallelization performance

1.7 GHz Intel Core i5
2 cores
Improvement on Memory Usage

- Based on frequent individual items, we generated a set of possible frequent paris,
  - Define these pairs as "candidate pairs"

- What if the number of candidates pairs are very large?
  - eg. not fit in memory
Hash Table

- Create a hash table with a number of buckets
- For each candidate pair, hash it to one bucket
- We only count the frequency of each bucket, not the candidate pair
- Space Complexity
  - $O(k)$, $k$ is the # of buckets
  - Typically, # of buckets $<<$ # of candidate pair
Hash Table

● Frequent bucket
  ○ Frequency(bucket) > threshold

● If a bucket contains frequent candidate, then it must be frequent bucket

● Only the candidate pairs in frequent buckets need to be considered

● In our test, this method saves about 65% memory
Thank you!

Q & A?