面试常见的大数据问题之 TopK Question

赵敏 老师

扫描二维码关注微信/微博
获取最新IT面试情报及权威解答

微信: ninechapter
知乎专栏: http://zhuanlan.zhihu.com/jiuzhang
微博: http://www.weibo.com/ninechapter
官网: www.jiuzhang.com

Outline

- TopK on single node
- TopK on multiple nodes
- Realtime topK with low frequency
- Realtime topK with high frequency
- Bonus: Approx TopK
- Bonus: MapReduce

TopK on Single Node(Order by Value)

九章算法

Question:

Given a list of unsorted Integer, find top k based on value.

list = [3, 10, 1000, -99, 98, 99], k = 3

topK = ?

TopK on Single Node(Order by Value)

Which data structure we will use?

TopK on Single Node(Order by Value)

PriorityQueue

TopK on Single Node(Order by Value)

Priority Queue:

Like a regular queue, but all elements are ordered based on their value.

Function:

- offer()
- peak()
- poll()

TopK on Single Node(Order by Value)

PriorityQueue:

- Ascending?
- Descending?

TopK on Single Node(Order by Value)

九章算法

Why Ascending?

list = [3, 10, 1000, -99, 98, 99], k = 3

Workflow: nextNum > smallest_number_in_queue? replace: continue

- [3, 10, 1000]
- -99<3, continue ⟶ [3, 10, 1000]
- 98>3, replace ⟶ [10, 98, 1000]
- 99>10, replace ⟶ [98, 99, 1000]

TopK on Single Node(Order by Frequency)

Question:

Given a list of Twitter Hashtag, find hottest K hashtag.

TopK on Single Node(Order by Frequency)

九章算法

Overview:

- Data preprocessor
- Priority Queue

TopK on Single Node(Order by Frequency)

九章算法

Step1: Data Preprocessor

Which data structure to use?

TopK on Single Node(Order by Frequency)

九章算法

Step1: Data Preprocessor

```
HashMap<String, Integer> counter = new HashMap<>();
for (String word : words) {
    if (counter.containsKey(word)) {
        counter.put(word, counter.get(word) + 1);
    } else {
        counter.put(word, 1);
    }
}
```

TopK on Single Node(Order by Frequency)

Step 2: What data to be stored in PQ?

- Word?
- Frequency?

TopK on Single Node

Key + Frequency:

```
class Pair {
        String key;
        int frequency;
        Pair(String key, int frequency) {
                this.key = key;
                this.frequency = frequency;
        }
}
```

TopK on Single Node


```
private Comparator<Pair> pairComparator = new Comparator<Pair>() {
        public int compare(Pair left, Pair right) {
            return left.frequency - right.frequency
 };
```

TopK on Single Node

九章算法

http://www.jiuzhang.com/solutions/top-k-frequent-words/

TopK on Single Node(Order by Frequency)

Time Complexity:

O(n +n*lgk) = O(nlgk)

Space Complexity:

O(|n|+k)

(|n| = number of unique words)

TopK on multiple nodes

场景一：

假设给一组10T文件，文件内容是10million用户，当天的微博搜索记录，求微博今日热搜？

TopK on multiple nodes

能否继续用single node处理？

TopK on multiple nodes

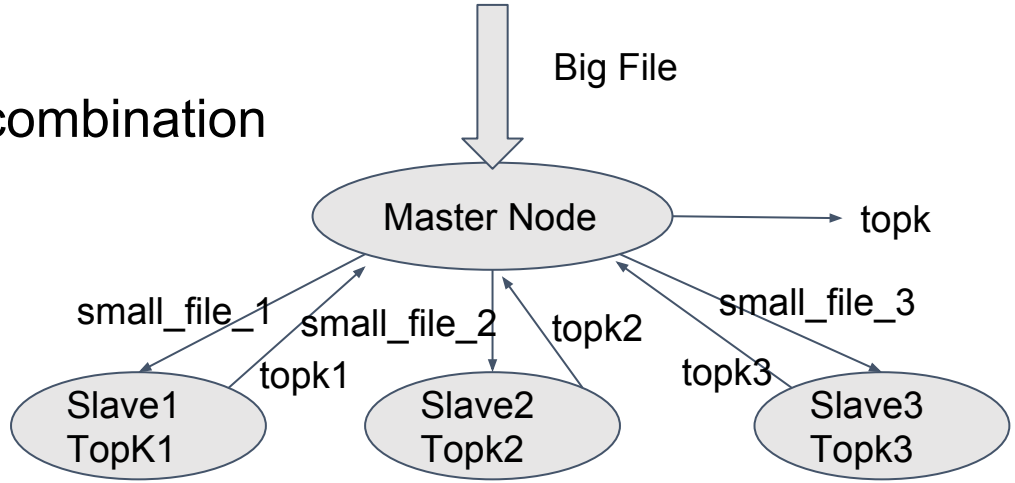
不可以:

- 文件太大，单机无法处理
- 处理速度太慢

TopK on multiple nodes

Overview: 分 & 合

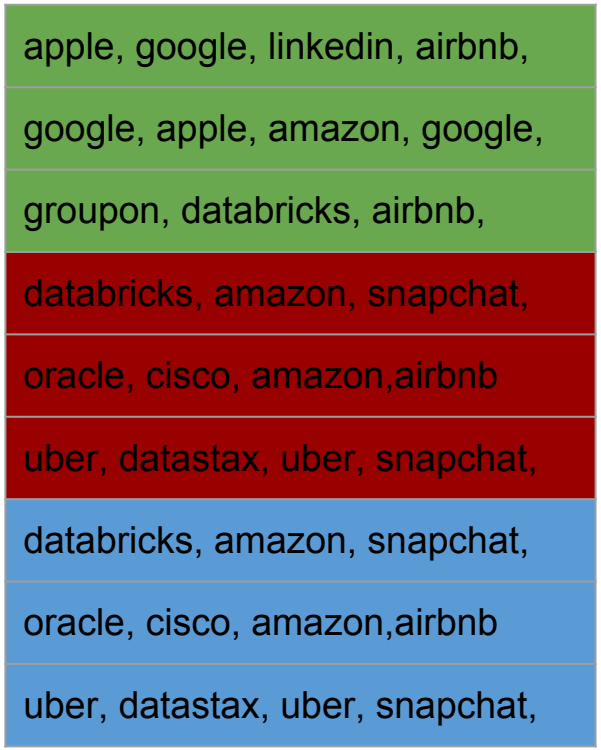
- Cut into small files
- Dispatch small files to different machines(node)
- Get topk from each machine
- Calculate topk from the topk combination



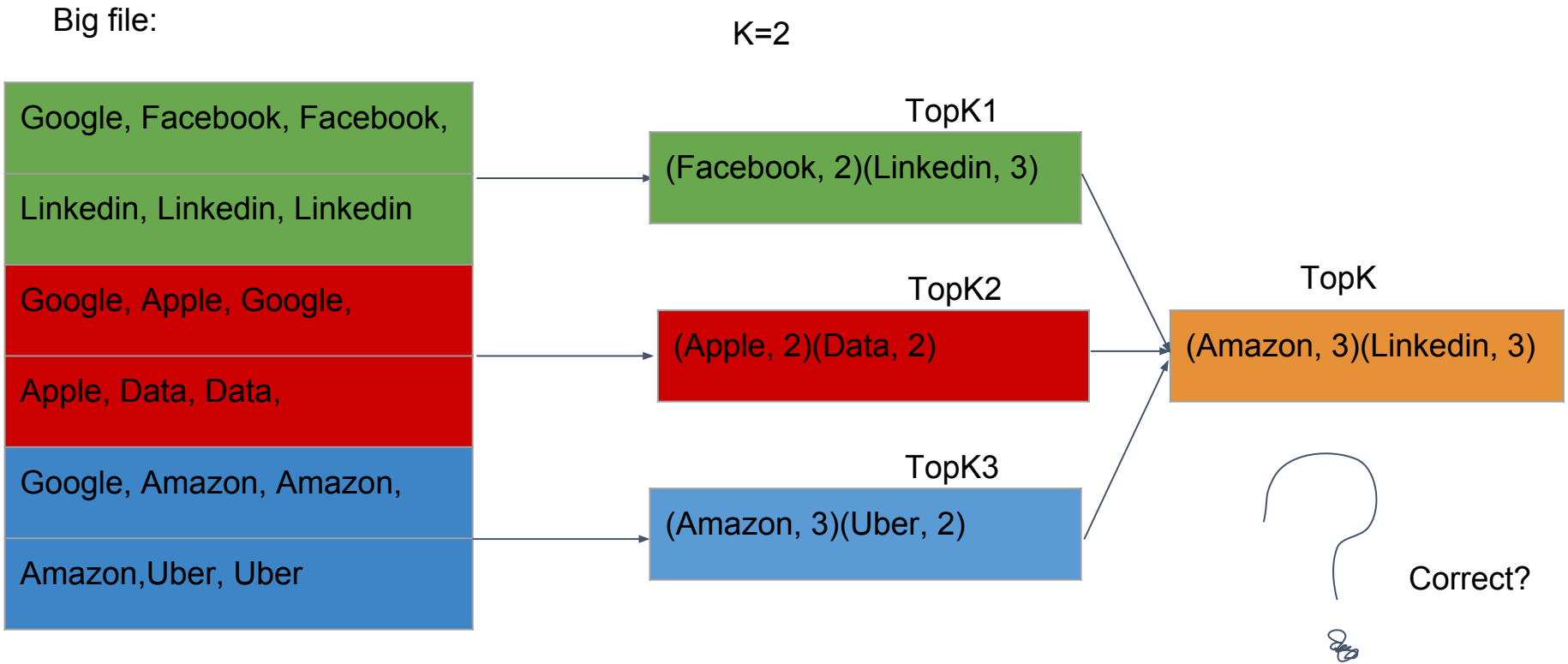
TopK on multiple nodes: 分

Divide Solution:

- Divide by file order?



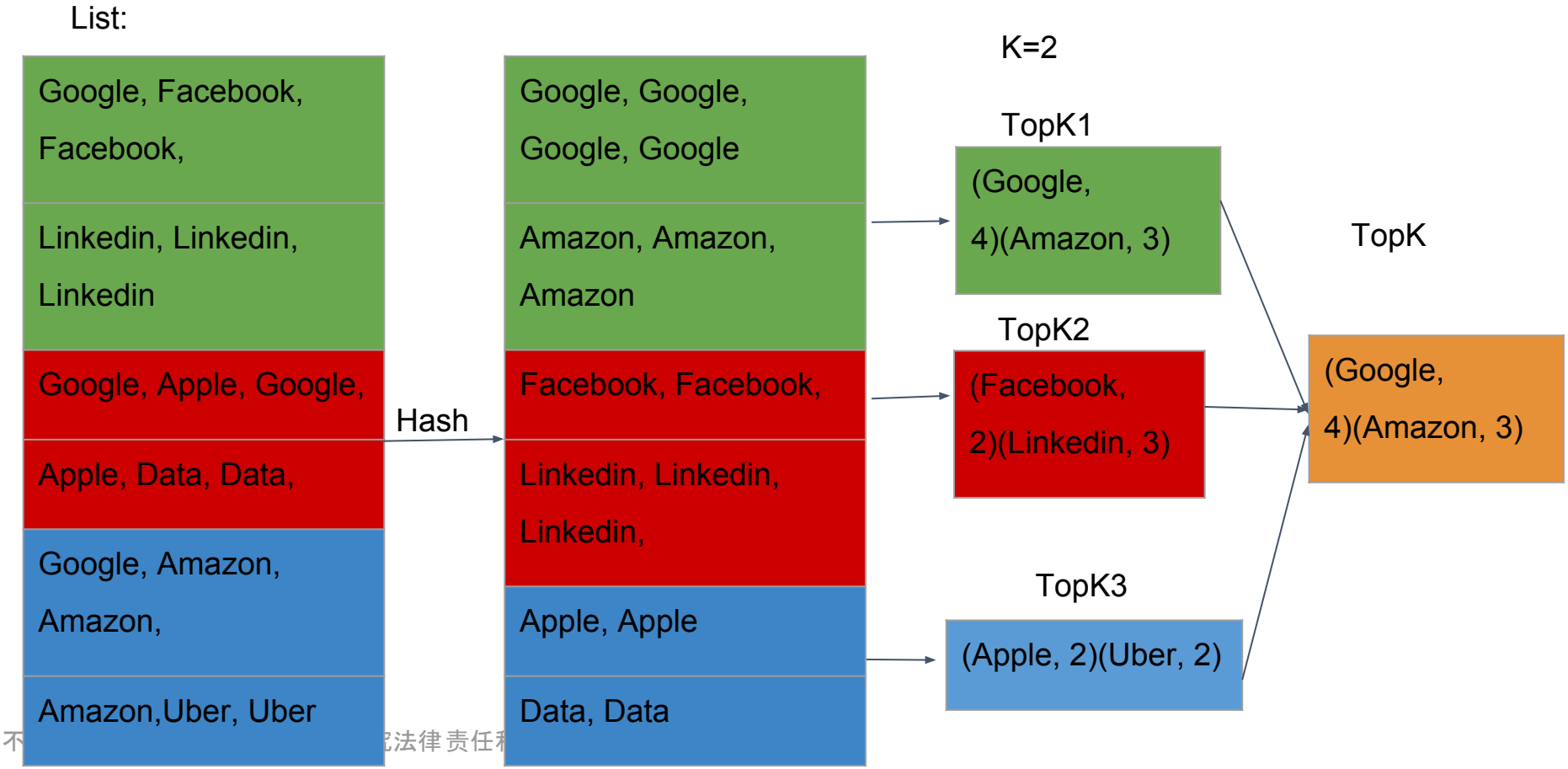
TopK on multiple nodes: 分



TopK on multiple nodes: 分

- Divide by hash value
  - SHA1
  - MD5

TopK on multiple nodes: 分



TopK on multiple nodes: 合

- Get list of topk: {topk1, topk2, topk3,...}
- Get final topk.

TopK on multiple nodes

场景二：

有N台机器，每台机器各自存储单词文件，求所有单词出现频率的topK

TopK on multiple nodes

Solution：

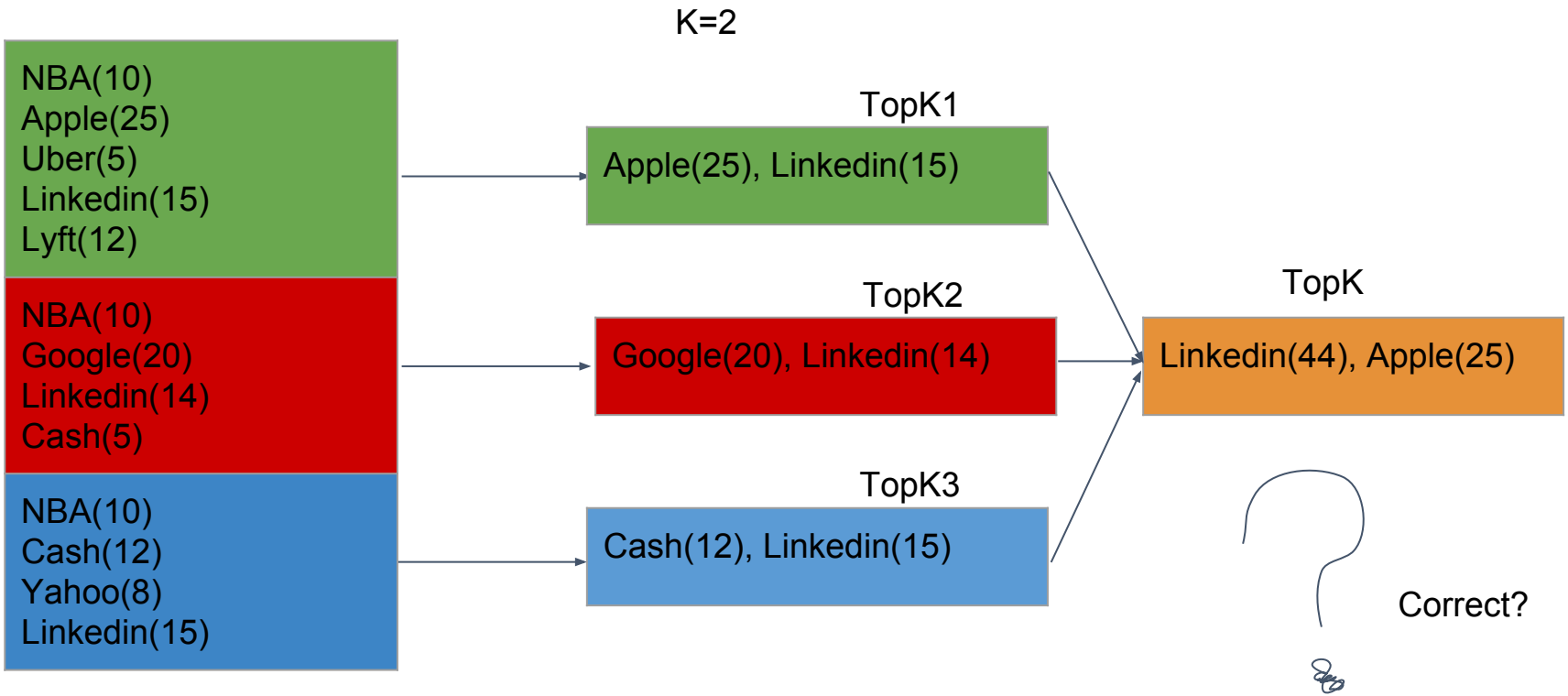
- Get topK from each machine
- Merge {topk1, topk2...}, get final topK

TopK on multiple nodes

Wrong!

TopK on multiple nodes: Divide

File on each disk:

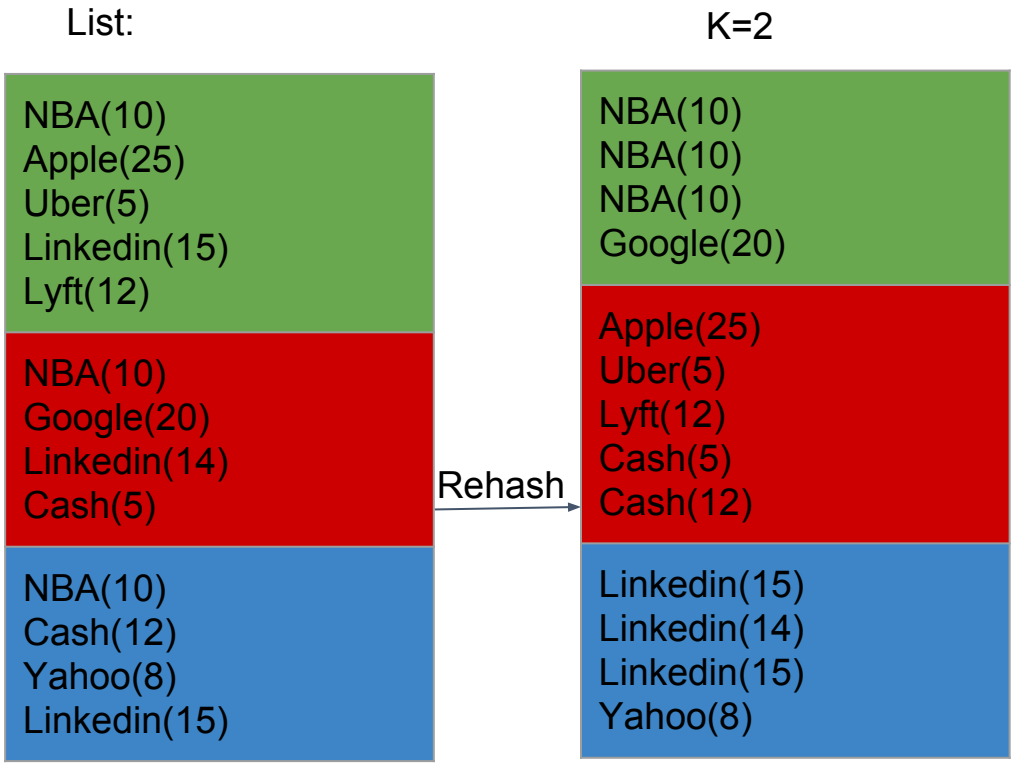


TopK on multiple nodes

九章算法

Rehash!

TopK on multiple nodes: Divide



Realtime TopK with Low Frequency

场景：有实时数据流进入，需要实时计算topK

Realtime TopK with Low Frequency

Solution:

- When new data comes in, write it to disk file
- When server request for topK, run the algorithm on disk file
- Get topK

Realtime TopK with Low Frequency

Disadvantage:

- Too slow!

Realtime TopK with Low Frequency

Solution:

- When new data comes in, write it to hashmap
- When hashmap is updated, trigger the PQ at the same time
- Get topK

Realtime TopK with Low Frequency

Disadvantage:

- OOM(out of memory)
- Data loss when node failure or power off.

Happens in which stage?

- Hashmap?
- Priority Queue?

Realtime TopK with Low Frequency

Replacement for HashMap:

- Store data in database
- Update counter in database

ID	Hashtag	Counter
1	Hello	2
2	World	5
...	...	...

Realtime TopK with Low Frequency

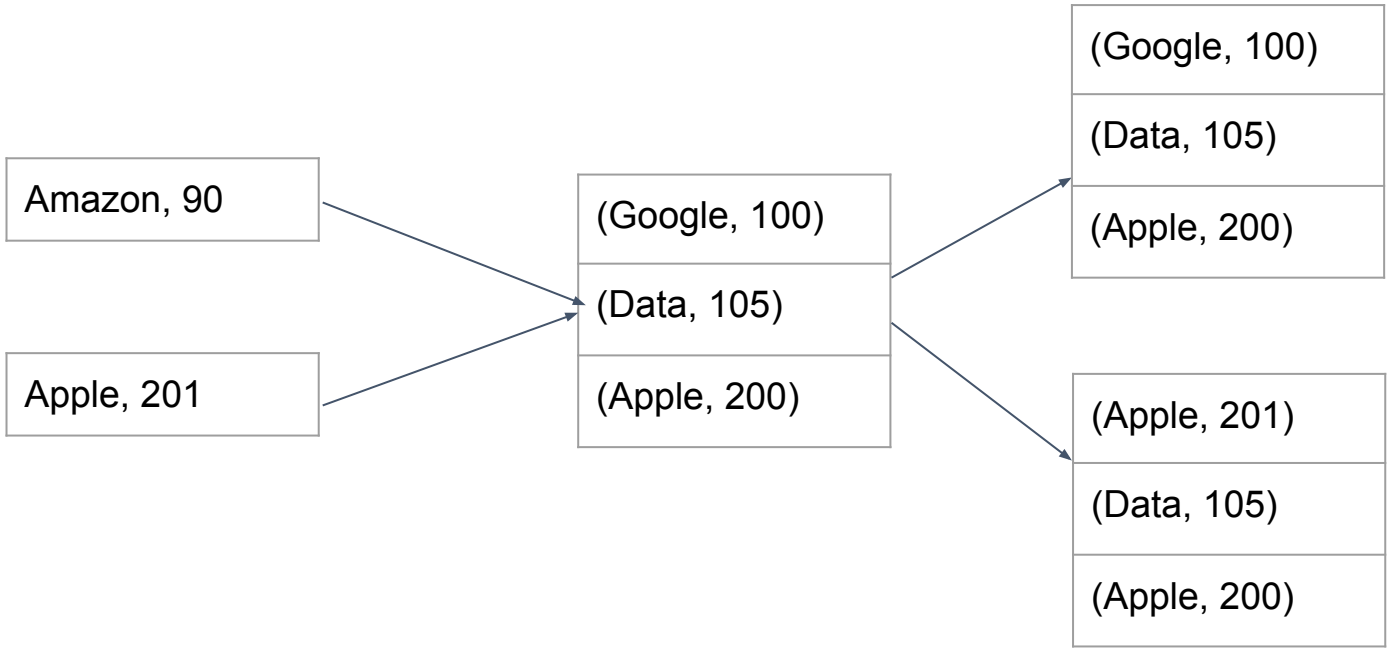
Does PQ have the same function as before?

- Store topK
- When new data comes in, compare new data with smallest data
- Replace or continue

Realtime TopK with Low Frequency

九章算法

No!!



Realtime TopK with Low Frequency

Solution → Use treemap to replace PQ

- Order by value
- Support all the functions in PQ
- Support find and delete by key

Realtime TopK with High Frequency

Follow-up:

What if the input stream has high frequency?

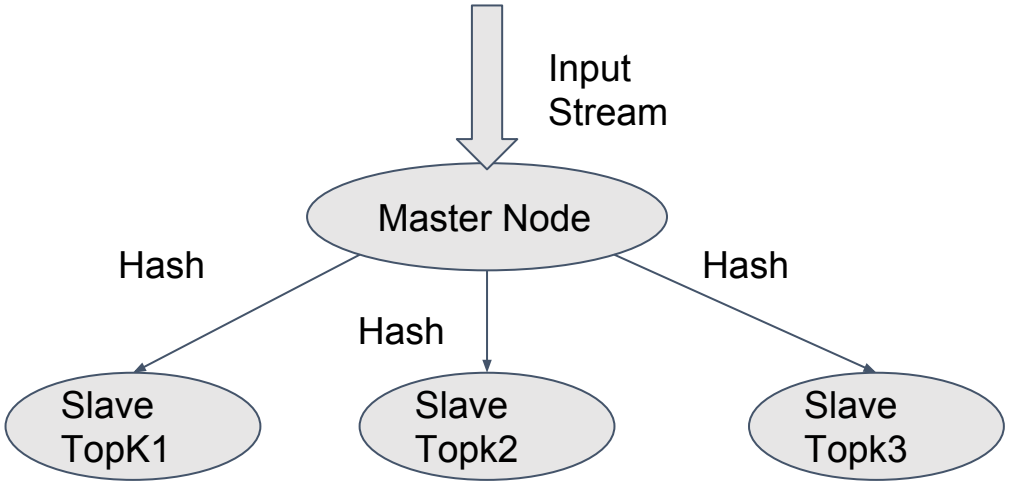
Realtime TopK with High Frequency

Qps too high → database couldn’t respond immediately → high latency

Realtime TopK with High Frequency

Solution:

分:



Realtime TopK with High Frequency

合:

- Get topK from each machine
- Merge {topk1, topk2...}, get final topK

Realtime TopK with High Frequency

Follow-up:

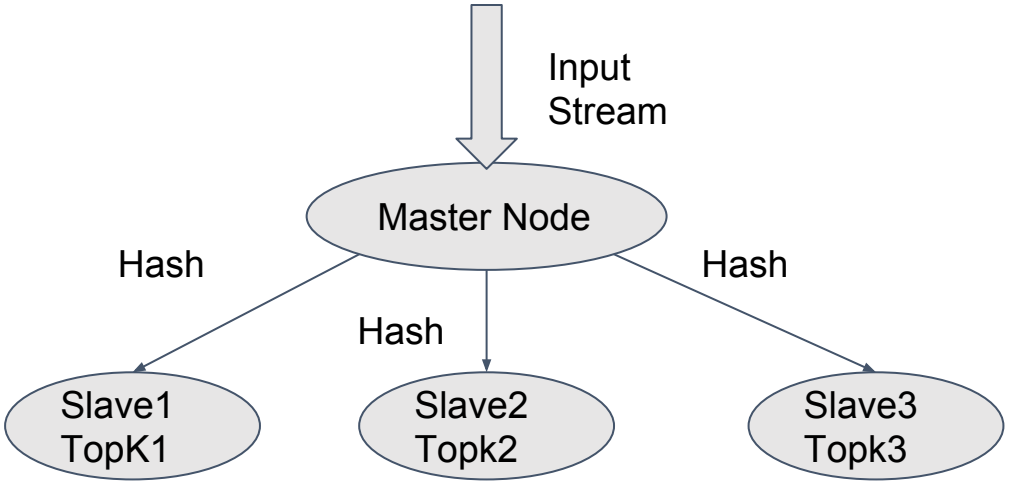
What if one key is too hot, writing frequency is very heavy on one node?

What will happen?

Realtime TopK with High Frequency

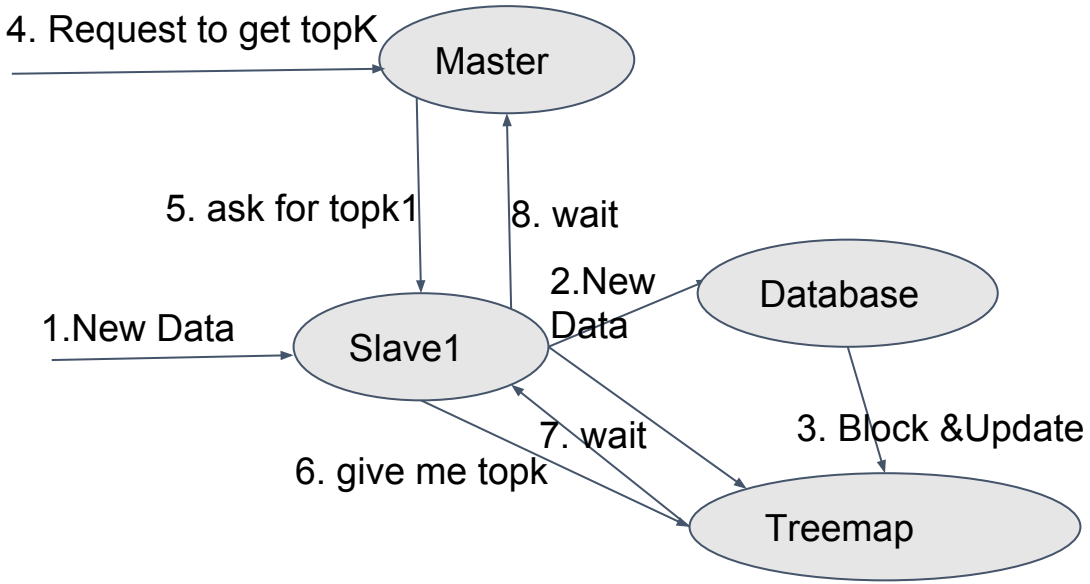
High Latency!

Realtime TopK with High Frequency



Slave1(HashValue: 0-10)
Slave2(HashValue: 11-20)
Slave3(HashValue: 21-30)
…

Realtime TopK with High Frequency



Slave1 receive new data

Slave1 update counter in data base

Slave1 update topk inside PriorityQueue (PriorityQueue will be locked for writing)

Slave1 reading topK operation will be waiting

Realtime TopK with High Frequency

Confliction:

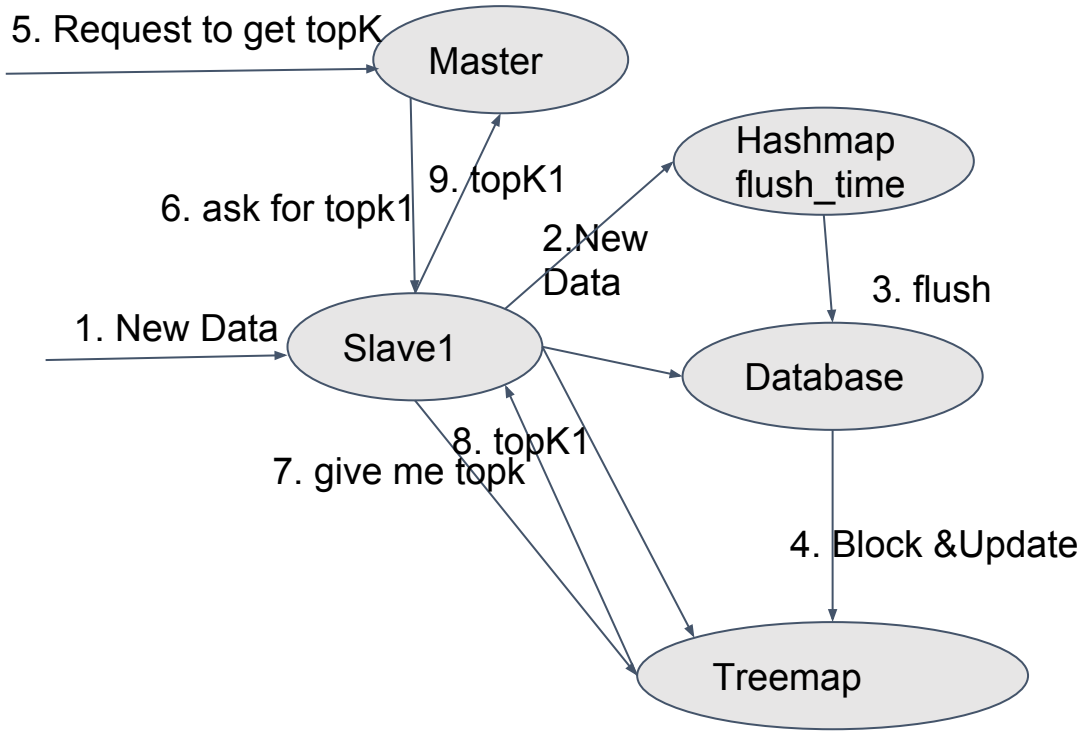
Acurracy VS Latency

Realtime TopK with High Frequency

Solution:

- Cache

Realtime TopK with High Frequency



What to cache?
Keyword-count pair.

Where to cache?
Slave node.

Slave1 receive new data

Store new data into hashmap

Slave1 receive new data

Store new data into hashmap

After flush_time, flush data into database

Question

Store all words on disk:

Low frequency words take up so much space.

Approx TopK Algorithm

Solution:

Tradeoff: Sacrifice accuracy for space

- Flexible space!
- O(logK) time complexity
- 牛逼！

Approx TopK Algorithm

Overview:

- When a new word comes in, update its count in hashmap
- Update topK in treemap

Approx TopK Algorithm: Hashmap

Hashmap:

- Key = word_hashvalue
- Value = frequency
- size can be customized

Approx TopK Algorithm

Inputstream

Apple(1)
Google(4)
Facebook(4)
Google(4)
Apple(1)
Linkedin(2)
Uber(6)
Google(4)
Uber(6)
Google(4)
Apple(1)
Amazon(6)
Snapchat(2)
Uber(6)
Apple(1)

Hashmap

Key	Value
1	2
2	1
4	3
6	2

TreeMap

Key	Value
Apple	2
Google	3
Facebook	2

Approx TopK Algorithm

Disadvantages:

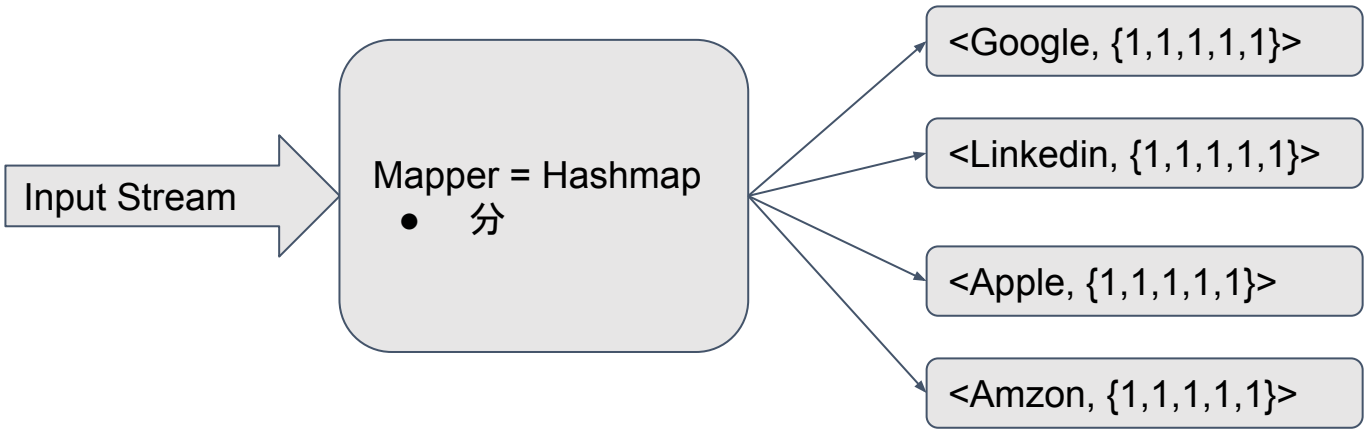
- All low frequency words will be hashed to same value, which will result in incoerrect result. (low possibility)
- Some low frequency words will come later, which will have a great count, then replace other high frequency words.(bloom filter)

Approx TopK Algorithm

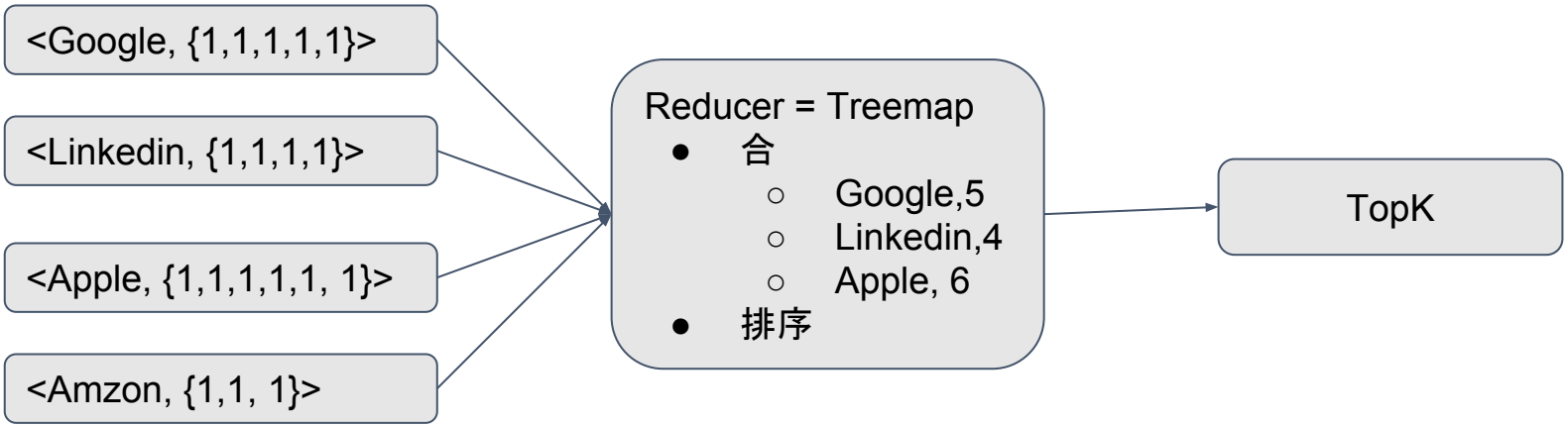
Bloom Filter:

- Hashmap will have 3 different hash functions
- Choose the lowest count from hashmap.
- https://en.wikipedia.org/wiki/Bloom_filter

Use Mapreduce to Solve it: Mapper



Use Mapreduce to Solve it: Reducer



Use Mapreduce to Solve it

九章算法

http://www.lintcode.com/en/problem/top-k-frequent-words-map-reduce/

What you learned

- Use PQ and Hashmap to calculate topK
- When the file is too big → 先分再合
- Real time data analytics system design
  - Latency vs Accuracy
  - Cache
- Approx TopK Algorithm
- Mapreduce to implement topK