Big Data Management

Technische Universiteit **Eindhoven** University of Technology

Milestone 2

Group 6

Tingyuan Zhang, Ruichen Hu, Simin Sun,
Mingzhe Shi, Siyue Chen, Jing Wu

https://youtu.be/K1AB8Pm_m4g

Stream description and processing

- We use the first version of the stream generator, which continuously generates a stream of the following format: $< server\_id, \quad hashed\_ip >$. Each entry of this stream represents a single request from the client with IP address ip to the server with ID sid. The order of the entries in the stream corresponds to the order in which the requests arrived.
- When we get original stream data by using $readStream$, we generate the corresponding hash value by setting five different hash functions generated by five $random\_hash\_seed$, and add it to the original stream dataframe.
- For each 2-second batch, we imply count-min sketch algorithm on a 60-second window. By choosing proper number of hash functions and the number of counters, we can conclude a result that satisfies error bound probability requirements.

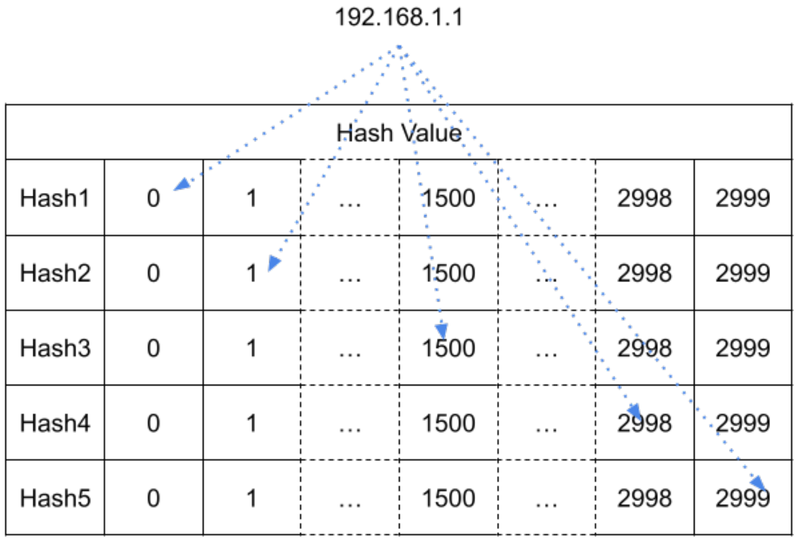
Count-Min Sketch for inner product

Parameters:

- $\tau = 3000$
- $\epsilon = 0.001, \delta = 0.01$
- Number of hash functions $d = \lceil ln\frac{1}{\delta} \rceil = 5$
- Number of counters $w = \lceil \frac{e}{\varepsilon} \rceil = 2719$ (In our project: #counters = 3000)

Our work, for stream processing, implies five hash functions described in Figure 1 on each incoming ip address, meanwhile add a column $batch\_id$ which performs as a window identifier.

For batch processing part, we follow Figure 2 every 2 seconds to calculate the similarity of each pair of sids in a 60-second window and report number of pairs whose similarity exceed threshold τ.

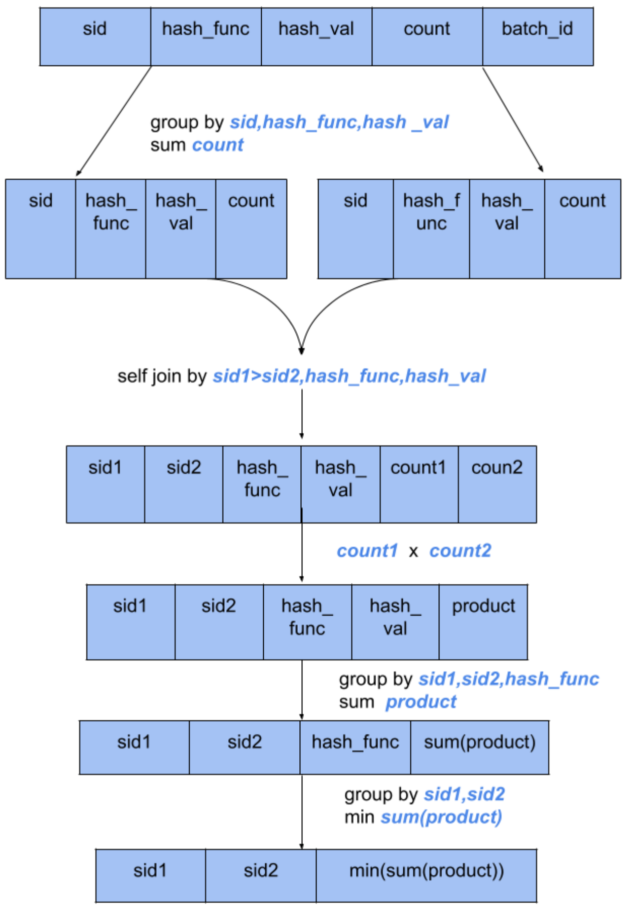


Hash Value							
Hash1	0	1	...	1500	...	2998	2999
Hash2	0	1	...	1500	...	2998	2999
Hash3	0	1	...	1500	...	2998	2999
Hash4	0	1	...	1500	...	2998	2999
Hash5	0	1	...	1500	...	2998	2999

Figure 1: How hash function works for ip address

Optimization

- Use $batch\_id$ as window identifier. Since Batch_id can be accessed from spark context and saved as a simple integer value, we can prevent from filtering on window timestamp and therefore make optimization.
- Use $complete$ output mode, get the latest stream by batch_id from the dynamically updated dataframe of the entire stream.
- For cluster partition setting, try different partition settings to have the best performance: getting maximum of parallelization while reduce network cost on shuffling.



sid	hash_func	hash_val	count	batch_id

group by *sid,hash_func,hash _val*
sum *count*

sid	hash_func	hash_val	count

sid	hash_func	hash_val	count

self join by *sid1>sid2,hash_func,hash_val*

sid1	sid2	hash_func	hash_val	count1	coun2

count1 x *count2*

sid1	sid2	hash_func	hash_val	product

group by *sid1,sid2,hash_func*
sum *product*

sid1	sid2	hash_func	sum(product)

group by *sid1,sid2*
min *sum(product)*

sid1	sid2	min(sum(product))

Figure 2: Process architecture

Results

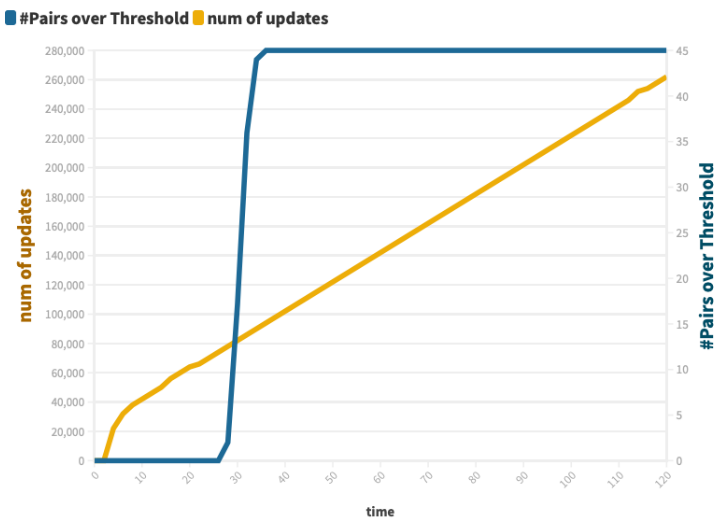


Figure 3: Plot of the results

/ Department of Mathematics and Computer Science