# Google Flow Analysis Using MapReduce

Strengths and Limitations

Markus De Shon Sr. Security Engineer

## Agenda

MapReduce What is it?

Case Study Entropy Timeseries

Scaling MapReduces

Other thoughts, Conclusions

Google



A parallel computational method

3 stages

- Map: Apply function(s) to each record, compute a sharding key
- Shuffle: Group data by sharding key
- Reduce: Apply function(s) to records for each key

Optimal for trivially parallelizable problems

• Our problems sometimes are, sometimes not...



This is where the magic happens...

Transport

- Locally: localhost sockets
- Different host: RPC of protocol buffer over TCP socket

There is no free lunch (e.g. count distinct)

- How is data distributed among input shards?
- Ideally, key by input shard (e.g. input filename), but any non-trivial shuffle will defeat this
- Try to optimize (number of keys \* number of emits per key)

#### Case study: Entropy timeseries

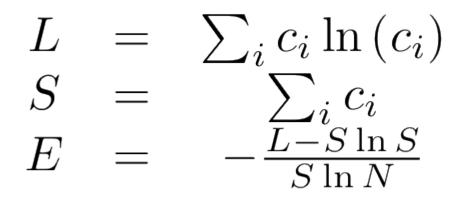


Normalized Shannon Entropy:

$$E = -\frac{1}{\ln(N)} \sum_{i} p_i \ln(p_i)$$

 $p_i$  = probability of each bin (count in bin i/N) N = total count

Single pass version (after binning):



"logsum" L, "sum" S, "entropy" E c<sub>i</sub> = count in each bin

# Case study: Entropy: High-level design



#### Мар

• Only calculate partial sums

Shuffle

• Deliver data for each key to the shard handling that key

Reduce

- Calculate the final sums (L and S)
- Calculate the entropy



Мар

- Calculate the key (e.g. [source ASN, time bin])
- For each key, emit e.g. { source IP, packet count } tuples

Shuffle

- Reorganize data by the [source ASN, time bin] key
- A particular shard receives all the tuples for a particular [source ASN, time bin] key

Reduce

- Iterate through the data calculating a map[source IP] of packet counts
- Finally, iterate through the map and perform the one-pass entropy calculation

## Case study: Entropy: Optimization

Google

Typically, you would be generating multiple such entropy time series

• source IP, dest IP, source port, dest port

perhaps multiple weightings

- by packet count
- by byte count

Optmize by emitting once for each chunk of input records

- data type = enum { sIP, dIP, sPort, dPort }
- e.g. per [ASN, time bin] key do a single emit for a list of all your { data type, packet count, byte count } tuples
  - Advantage: Fewer RPCs
  - Danger: RPC too large

# Scaling MapReduces



#### Мар

- How many unique input sources?
  - Log files processed simultaneously
  - HBase rows
- How is data distributed by sharding key?
  - More grouping is better

Reduce

- How many unique sharding keys?
  - More than that many shards is pointless
- Memory/CPU allocation per shard



Frequent, small MapReduces over recently arrived data

Time windowing vs. latency are critical considerations (cursors)

Need good bookmarking of input files



#### SiLK http://tools.netsa.cert.org/silk

Can SiLK-like analyses be done using MapReduce? Sort of...

rwfilter

- Yes! Just matching, boolean forward or not on per-record basis
- Hard: doing ipsets, tuples efficiently per shard rwsort
  - Done automatically by sharding key, subkeys (depending on output method)

rwcount, rwuniq, rwbag

- Yes, but need to optimize for scalability rwstats
- Yes, rwuniq plus sorting by value rwset
  - Yes, sort of. Not easy, not optimized to IPv4
  - rwsettool: not really, not as elegantly

Quick, iterative analysis: Not really, unless... (cf. SQL/MR)

## Conclusions



Strengths

- Commodity computing platform
- Strong scalability for many problems of interest to us
- Good for ongoing, repeated analyses of large amounts of data
- "Real time" analyses feasible (not as much of a commodity)

Limitations

- Inherent overhead in shuffling phase
  - Irreducible anyway? Remember: no free lunch
- Not so good for iterative, ad hoc analysis (except SQL/MR)