Streaming
CS199 - ACC

Prof. Robert J. Brunner

Ben Congdon
Tyler Kim
Bhuvan Venkatesh

Streaming

- **Theory**
  - Distribution Algorithms
  - Streaming Algorithms
  - Streaming = Analytics!

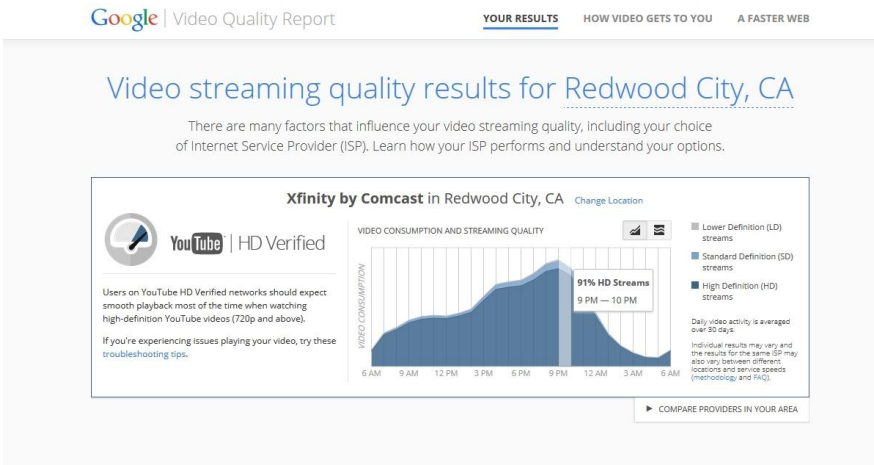
- **Actual Implementation**
  - Apache Storm
  - Spark Streaming
  - Kafka -- Not Exactly Streaming

Streaming Theory!

- Yay Theory!
- To clarify, clients send “events” (meaning either a click, data transmission, etc) and we want to process these “events”. We are not opening a TCP connection to them.
- Apart from fast networking, theory is everything
- We want to route “events” to idle machines as fast and efficiently as possible
- We also want to do it somewhat randomly -- we want some kind of safety from attackers.

Streaming Metrics

- What should we do?
- We want to process data either by batch or one by one and give analytics
  - What is trending
  - What is the average
  - Have consumers changed their interest?
- In the end, we process one at a time



Hashing Algorithms

- We need to think that there is an adversary who wants to crash our server
- If we pick a bad hashing algorithm, then an attacker can exploit that and overload a particular server, causing our entire system to fail
- We want to be careful hashing by
  - Content
  - IP Address
  - Length/Delay “event”
  - Rate
- We want theoretical guarantees of goodness

Hashing Algorithms - Example

- Try to avoid cryptographically secure (they can be broken for small keys)
- Universal/near-universal Hash Families a must
- We need “true” randomness! Any hard coded method can be broken
- (From Wikipedia - Universal Hashing)

```
uint hash(String x, int a, int p)
    uint h = INITIAL_VALUE
    for (uint i=0 ; i < x.length ; ++i)
        h = ((h*a) + x[i]) mod p
    return h
```

Streaming Algorithms

- We want our algorithm to take O(1) per element
- Easy way: Take your non-streaming algorithm and run on the appropriate sized subset of your stream
  - An O(n) algorithm, take sets of size k, every time
  - An O(n^2), take sets of size k, skip the next k^2-k elements
- As you see, the algorithms get inefficient as complexity goes up

Streaming Algorithms - Examples

Top k elements in a stream

```
def topk(stream, k)
    first = sorted([next(stream) for _ in range(k)])
    while stream:
        elem = next(stream)
        if elem > first[0]:
            first.drop(0)
            insertSorted( first, elem)
```

O(k*log(k) + n*log(k)). If k is small relative to stream size, then this runs in O(n)

Distributed Platforms


TWIST

Stream Sources

- File System
- Internet of Things
- Network Traffic
- Embedded devices on a radio frequency

- In storm, you write functions that takes bolts of data (size n subsets)
- You can write whatever algorithms that you want
- Storm will take the data, load balance, make the analytics fault tolerant
- The same data may be reprocessed multiple times (limitation of the CAP theorem)

- Really cool language agnostic tool/framework
- You process “bolts” of data from different “sprouts”
- At the end you grab analytics like what is trending
- This is what twitter uses
- Twitter processes a lot of tweets

Spark!


Kafka
Flume
HDFS/S3
Kinesis
Twitter
Spark
Streaming
HDFS
Databases
Dashboards

- Run spark on batches

Example: MLlib + Spark

```
trainingData = data.load()

model = model.train(trainingData)

stream = socketStream(localhost, 9999)

cleaned = stream.map(cleaner)

predictions = cleaned.transform(lambda rdd:

model.predict(rdd))

predictions.pprint() # Output or send it to another

stream/file/db
```

The Yucky

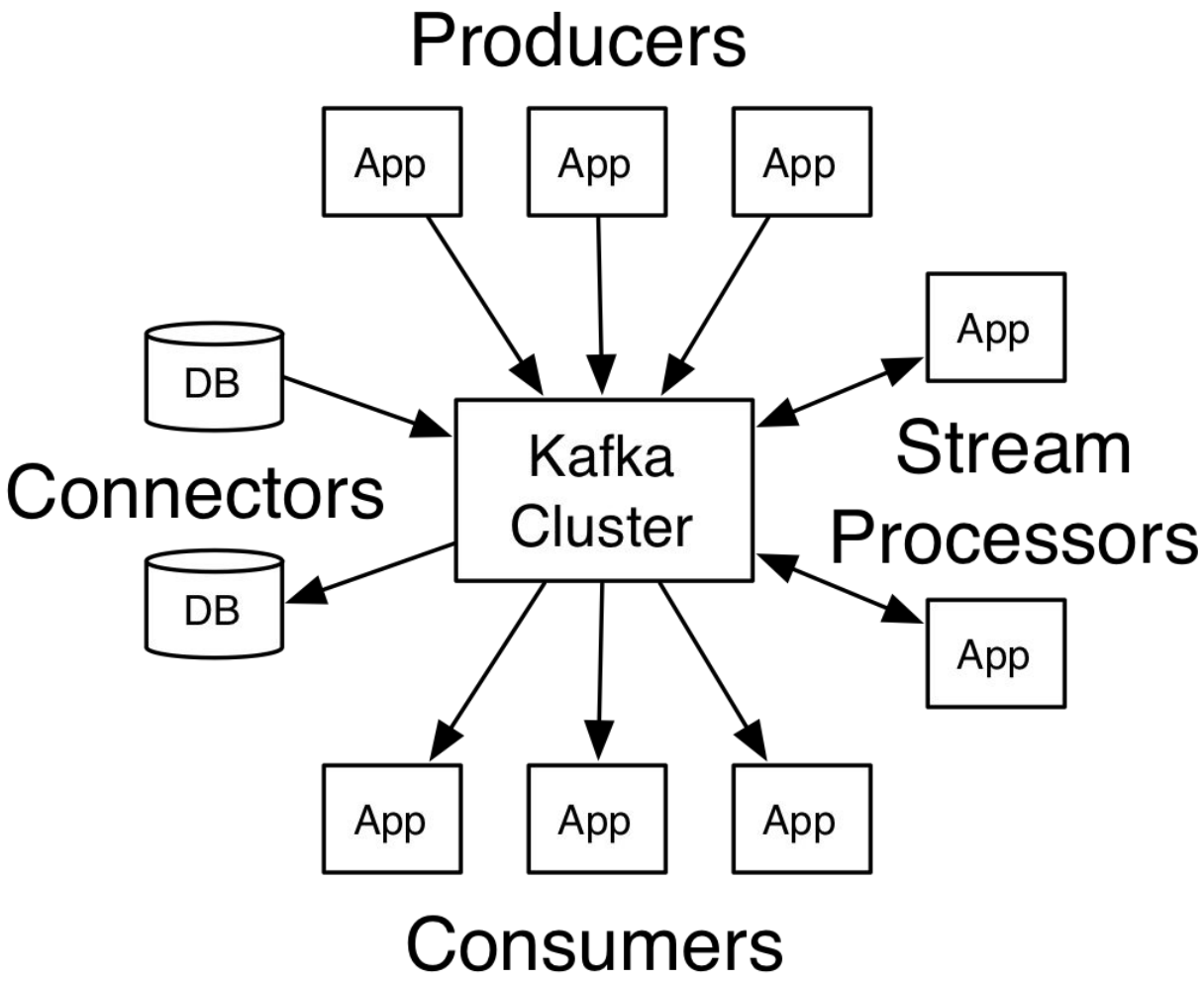
- We are working with dataframes, meaning that if you have a lot of operations and a large replication factor, streaming could be asymptotically efficient but in a practical sense could be overloaded
- Immutability! One cannot just poke out some data values, O(n) operations
- Hacky! It was built on top of an already batch processing
- But we can use our existing algorithms

Apache Kafka

- Not really streaming, but is 30 degrees off. If something is linearizable, that generally denotes that it cannot support streams where linearizability is not logistically possible
- Kafka is a distributed message queue, meaning that there are different queues that consumers can listen and publish on (pub-sub model)
- Not streaming inherently because there is a lack of analytics/linearizable
- Focused on Messages getting delivered in some ordering (linearizable)

Kafka-Esque

- Kafka is commonly used as a streaming service because although it tries to be consistent, it loses messages
- As such, it can be used as a streaming platform or a source for storm to perform its analytics
- If you want to read more about the faults in our kafka https://aphyr.com/posts/293-jepsen-kafka


Producers
App
App
App
App
DB
Connectors
Kafka
Cluster
Stream
Processors
DB
App
App
App
App
Consumers