BIG DATA

UNIT - 4

Streaming Analysis

feedback/corrections: vibha@pesu.pes.edu

VIBHA MASTI

Streaming Data

- Sensor data
- Images
- Internet / web traffic
- Real-time processing

STREAMING DATA MODEL

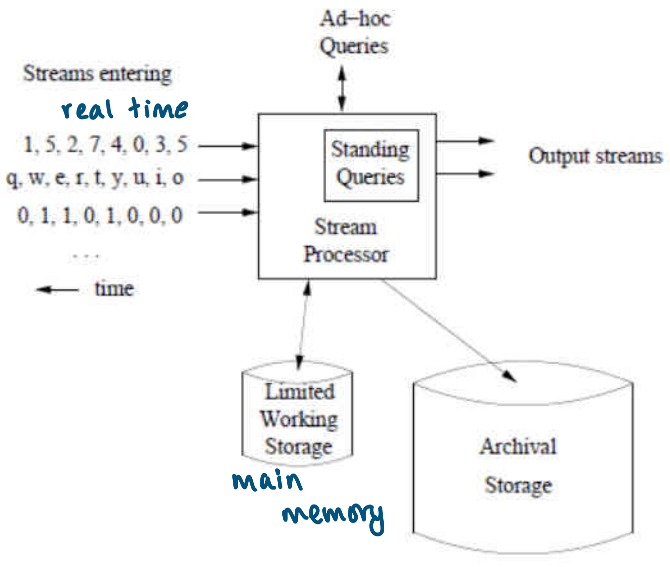


Figure 4.1: A data-stream-management system

- Multiple streams at different rates (not synchronised)
- Archival store: offline analysis
- Working store: real-time analysis
  - disk / memory (usually memory)

Types of Stream Queries

1. Standing Queries

- produce o/p at appropriate time
- query continuously running
- constantly read new data
- query exec can be optimised
- eg: no. of vehicles passing intersection every hour
- eg: max temp ever recorded

2. Ad hoc Queries

- not predetermined ; arbitrary
- need to store stream
- do SQL query
- eg: no. of unique users in the last 30 days

Q: Consider the queries below. Which among them are **STANDING QUERIES** and which are **AD HOC**?

- Alert when temperature > threshold   standing
- Display average of last n temperature readings; n arbitrary
  ad-hoc
- List of countries from which visits have been received over last year   ad-hoc
- Alert if website receives visit from a black-listed country   Standing

Issues in Stream Processing

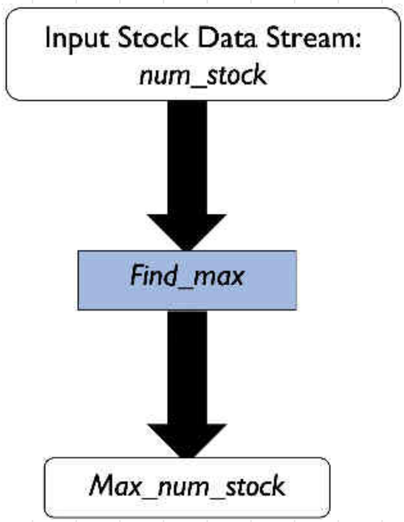
- Velocity (high)
- Volume (high)
- Need to store in memory

Framework Requirements

- Scalable to large clusters
- Second-scale latencies (low latencies)
- Simple programming model
- Integrated with batch & interactive processing
- Efficient fault tolerance

Q: Can Hadoop be used?

- The input is a stream of records from the stock market.
- Each time a stock is sold, a new record is created.
- The record contains a field num_stock which is the number of stocks sold.
- Find_max is a program that updates a variable Max_num_stock which is the maximum of num_stock.



- If Hadoop used, data must be stored and max program must run on entire dataset
- All transactions stored onto file
- Run MR program and share global max variable across nodes - difficult

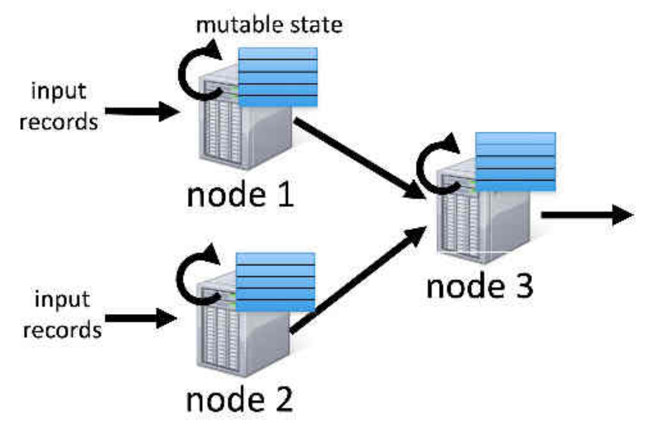
Case Study: Coniva Inc.

- Real-time monitoring of online metadata
- Two processing stacks
  1. Custom-built distributed stream processing system
     - many nodes req
  2. Hadoop backend for offline analysis
     - similar computation as the streaming system
- Twice the effort, bugs

Stateful Stream Processing

- Traditional streaming: event-driven, record-at-a-time processing model
  - each node: state
  - every record: update state

- State lost if node dies
- Hard to make stateful stream processing fault tolerant



- Global State: where to store?

Existing Streaming Systems

1. Storm
    - Processes each record at least once
    - May update mutable state twice
    - Replays record if not processed

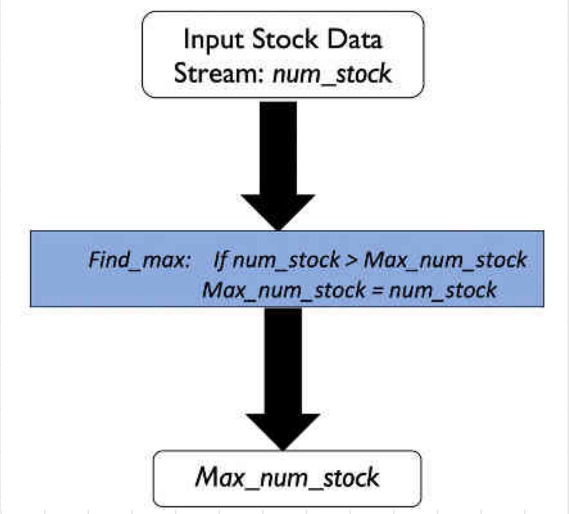
2. Trident
    - Use transactions to update state
    - Process each record exactly once
    - Per state transaction updates slow

SPARK STREAMING

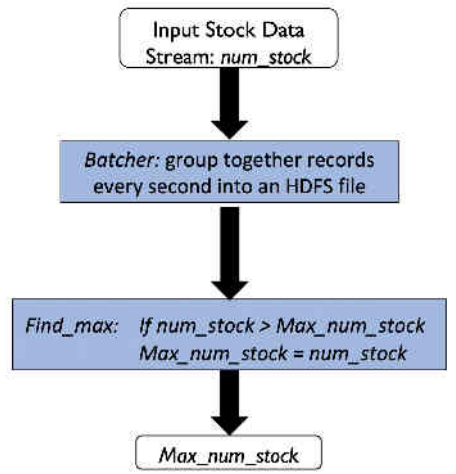
- Framework for large-scale stream processing
- 100s of nodes
- Integrates with Spark's batch and interactive processing
- Provides batch-like API for implementing complex algorithm
- Can absorb live data streams — Kafka, Flume, ZeroMQ etc

Can Hadoop be modified?

- Assume 1 sec updates acceptable
- Hadoop for stream processing?
- Ignore global variable problem

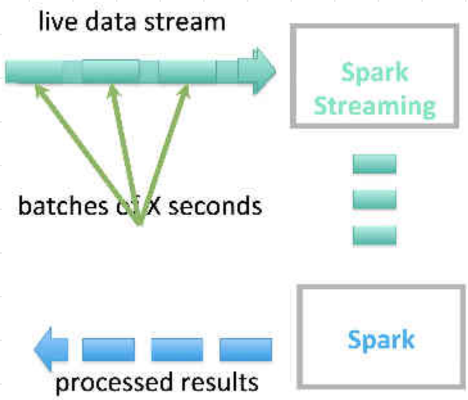


- Batch together input records every 1 sec into single HDFS file
- Every file processed using MR
- Update every second



Discretised Stream Processing

- Chop live stream into batches of X seconds
- Each batch treated as RDD by Spark
- Processed results of RDD operations returned in batches
- Batch sizes as low as ½ second, latencies as low as 1 sec
- Potential: combine batch processing and stream processing

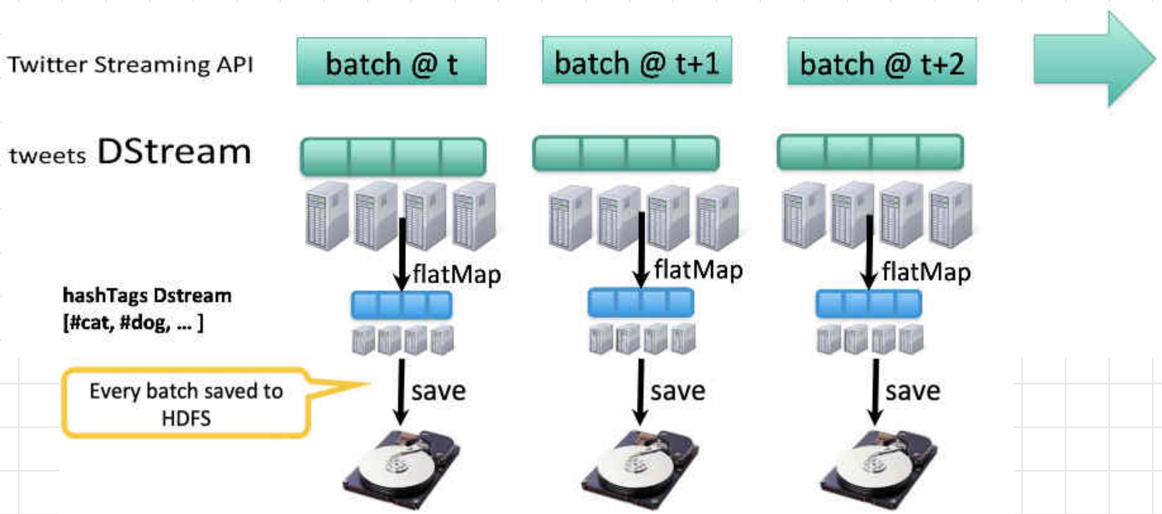


DStreams

- In Spark (not Streaming spark)
  - every variable — RDD
  - Pair RDDs — key-value pairs

Example: Get hashtags from Twitter

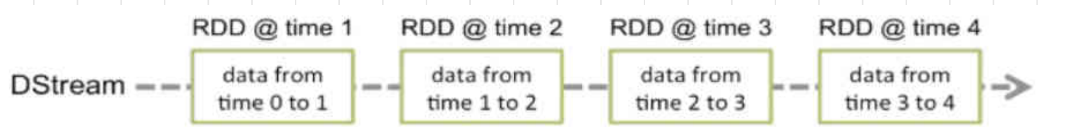
```
// twitterStream returns variable of type Dstream
// Dstream: sequence of RDD representing a stream of data
val tweets = ssc.twitterStream(<Twitter username>, <Twitter
password>)

// hashTags is new object of type Dstream
// flatMap transformation
// Dstream is sequence of RDDs
val hashTags = tweets.flatMap (status => getTags(status))

// Push data to external storage (HDFS)
hashTags.saveAsHadoopFiles("hdfs://...")
```



Spark Streaming- Execution of Jobs

1. DStreams and Receivers

- Streaming spark- batch processing
- Every DStream associated with receiver
  - read data from source
  - store into Spark memory for processing
  - types
    (i) Basic – file systems, sockets
    (ii) Advanced – Kafka, flume
- Relationship between DStream and RDD



- Streaming spark processes job
- Starts receiver on an executor as a long running job
- Driver starts tasks to process blocks on every interval

2. Transformations in Spark

- Stateless
- Stateful

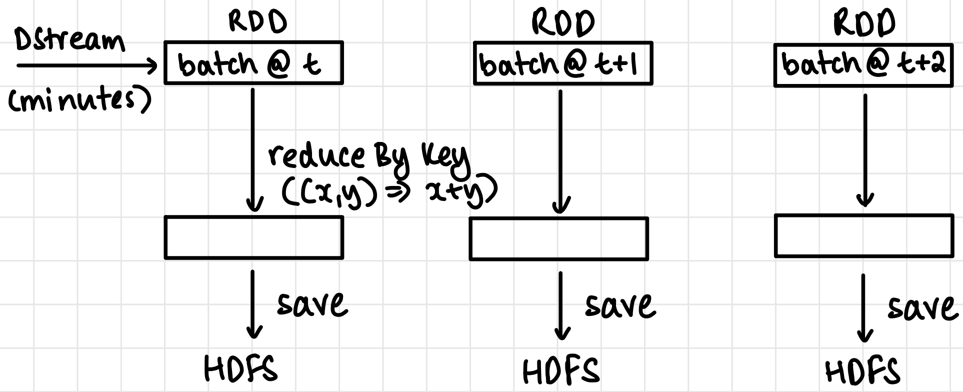
(a) Stateless Transformations

- Transformation applied independently to every batch
- No info carried forward from one batch to next
- Examples
  - Map()
  - FlatMap()
  - Filter()
  - Repartition()
  - reduceByKey()
  - groupByKey()

Q: Consider a Dstream on stock quotes generated similar to earlier that contains

- A sequence of tuples that contain <company name, stock sold>
- Need to find total shares sold per company in the last 1 minute

Show Streaming spark design for the same.



(b) Stateful Transformation

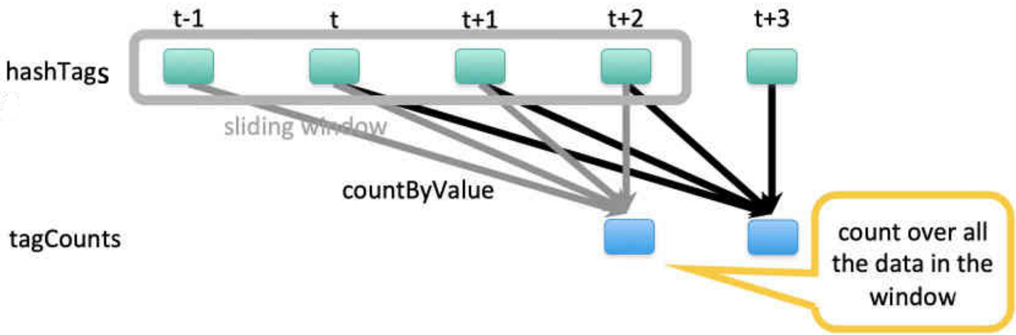
- State stored across different batches of data
- Eg: Max amount of stock sold across whole day for a company
- Data: pair RDDs
- Spark: two options
  (i) Window operator: state maintained for short periods of time (sliding window)
  (ii) Session based: state maintained for longer

(i) Window-based

Example: count hashtags over last 10 mins

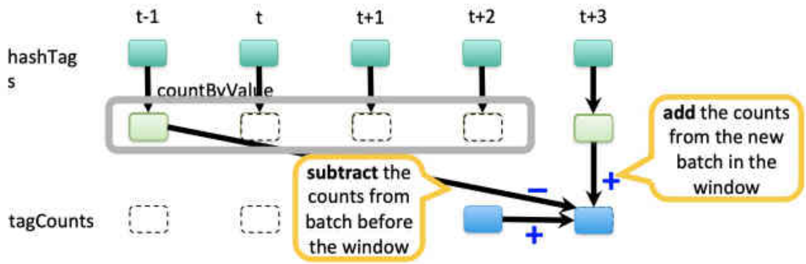
```
val tweets = ssc.twitterStream(<Twitter username>, <Twitter password>)
val hashTags = tweets.flatMap (status => getTags(status))
val tagCounts = hashTags.window(Minutes(10), Seconds(1)).countByValue()
```

(move window by)



Smart Window-Based CountByValue

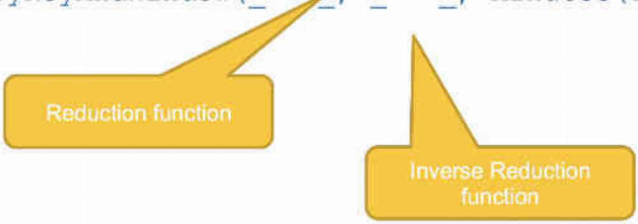
```
val tagCounts = hashtags.countByValueAndWindow(Minutes(10), Seconds(1))
```



Smart Window-based Reduce

- Reduce, inverse reduce
- Could have implemented counting as

```
hashTags.reduceByKeyAndWindow(_ + _, _ - _, Minutes(1), …)
```



(ii) Session-based

Q: Maintain per-user mood as state, update with their tweets

1. What has to be the structure of the RDD *tweets*?
   Hint – note that updateStateByKey needs a key

2. What does the function updateMood do?
   Hint – note that it should update per-user mood

```
tweets.updateStateByKey(tweet => updateMood(tweet))
```

- updateStateByKey uses the current mood and the mood in the tweet to update the user's mood

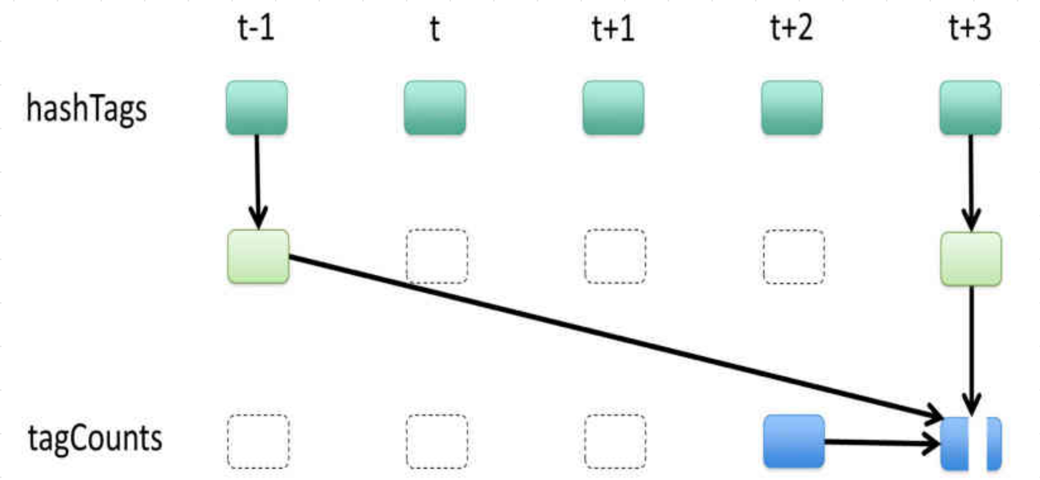
1. tweets: <user, mood>

2. Compute new mood based on current mood & new tweet

- *updateStateByKey* finds the current mood – Happy
- current mood (Happy) and tweet (Eating icecream) is passed to *updateMood*
- *updateMood* calculates new mood as VeryHappy
- *updateStateByKey* stores the new mood for Dinkar as VeryHappy

Fault Tolerant Stateful Processing

- All intermediate data are RDDs
- Can recompute if lost

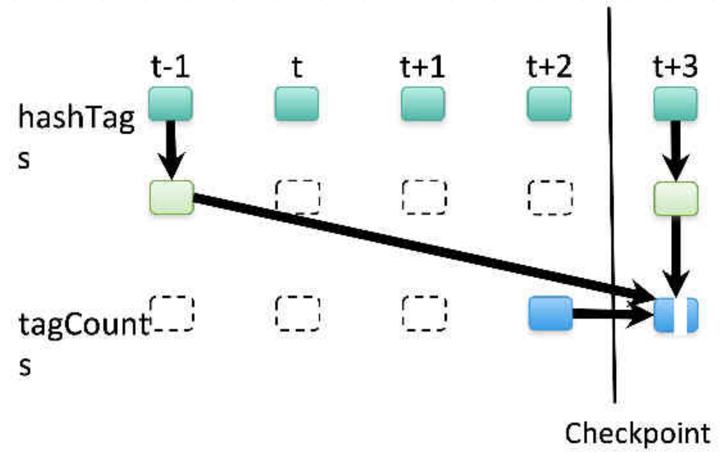


(i) Fault in Stateless
- recompute

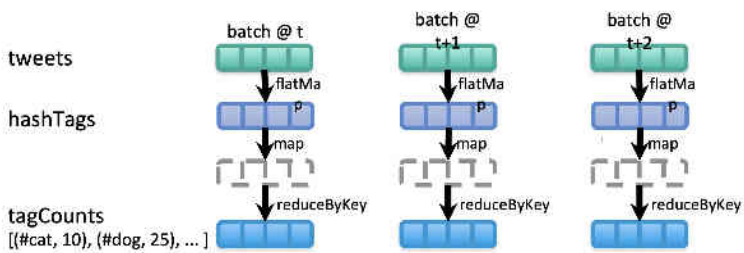
(ii) Fault in Stateful
- how much data to retain?

Checkpointing

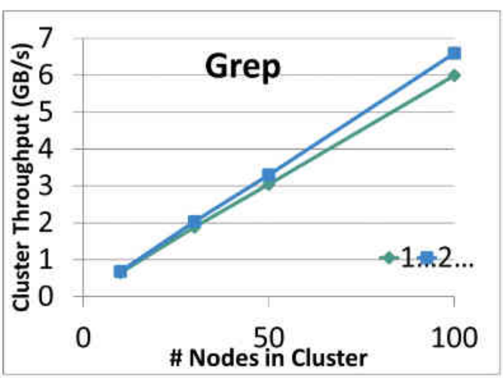
- Stores an RDD
- Forgets lineage
- Checkpoint at t+2
  - stores hashTags and tagCounts at t+2
  - forgets rest of lineage

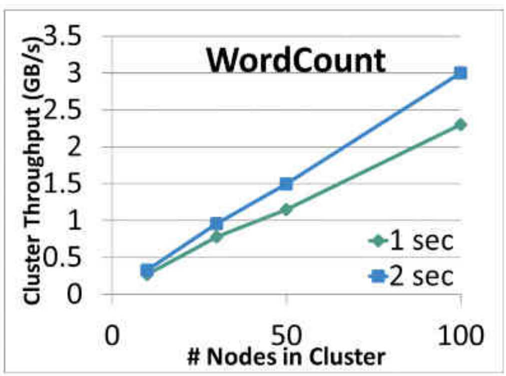



```
val tweets = ssc.twitterStream(<Twitter username>, <Twitter password>)
val hashTags = tweets.flatMap (status => getTags(status))
val tagCounts = hashTags.countByValue()
```



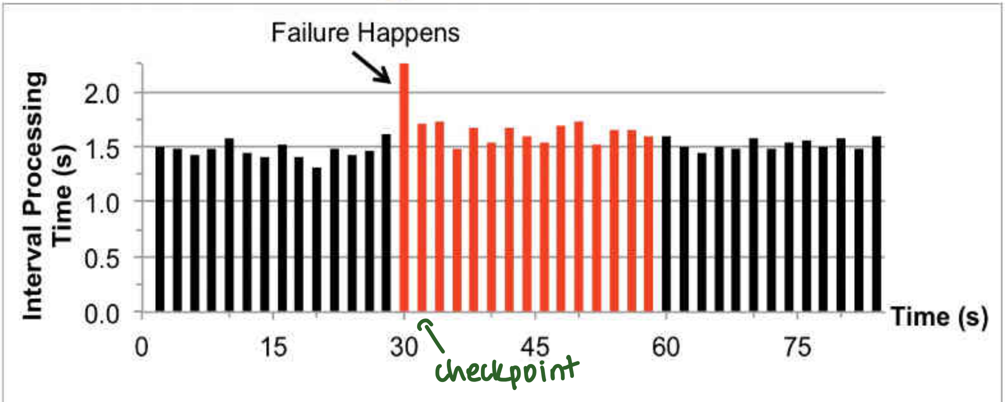
Performance





Fast Fault Recovery

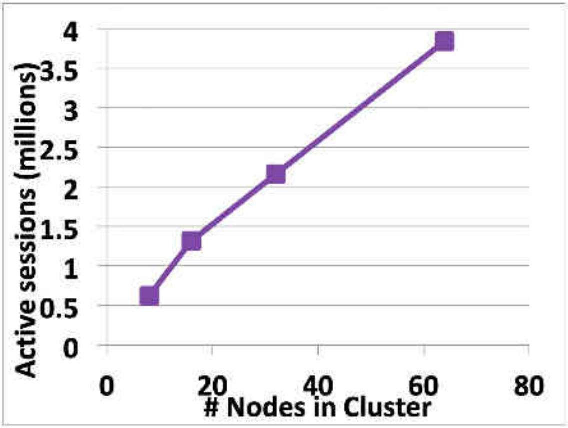
- Recovers from faults/stragglers within **1 sec**



Sliding WordCount on 10 nodes with 30s checkpoint interval

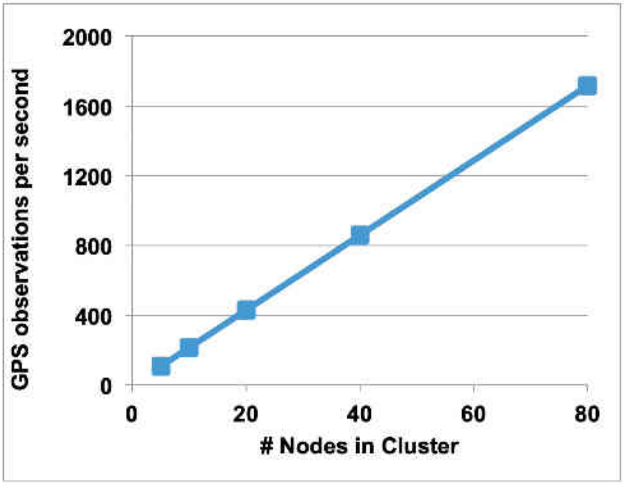
Real Applications

- Real-time monitoring of video metadata



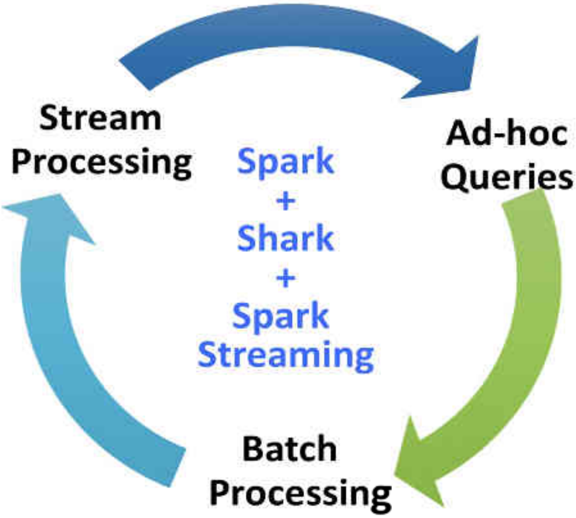
- Achieved 1-2 second latency
- Millions of video sessions processed
- Scales linearly with cluster size

- Traffic transit time estimation using online machine learning on GPS observations



- Markov chain Monte Carlo simulations on GPS observations
- Very CPU intensive, requires dozens of machines for useful computation
- Scales linearly with cluster size

Spark, Shark (like Hive), Spark Streaming



Spark vs Spark Streaming

Spark Streaming program on Twitter stream

```
val tweets = ssc.twitterStream(<Twitter username>,
<Twitter password>)
val hashTags = tweets.flatMap (status => getTags(status))
hashTags.saveAsHadoopFiles("hdfs://...")
```

Spark program on Twitter log file

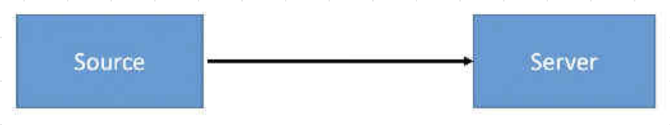
```
val tweets = sc.hadoopFile("hdfs://...")
val hashTags = tweets.flatMap (status => getTags(status))
hashTags.saveAsHadoopFile("hdfs://...")
```

Streaming Spark limitations

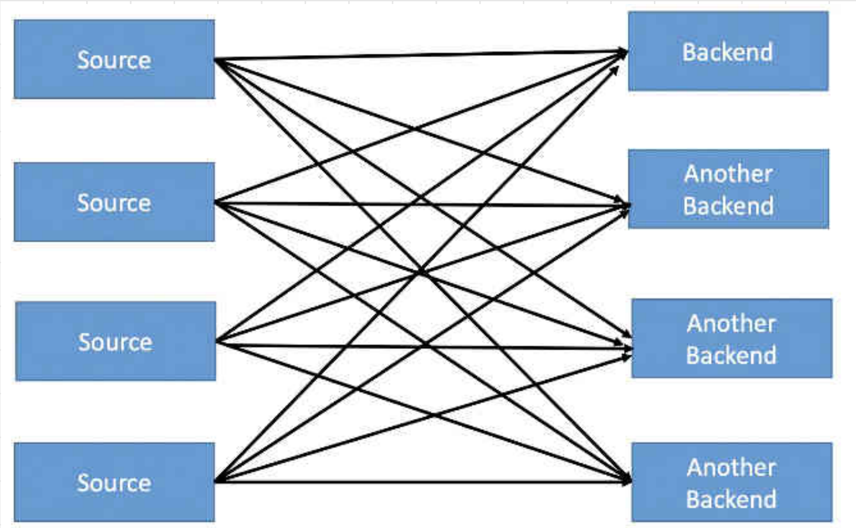
- Near real time
- Not necessarily acceptable for certain scenarios

Kafka

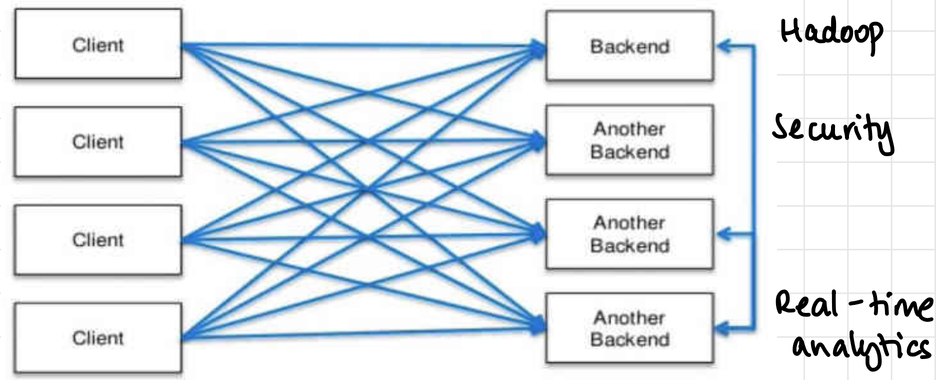
- Processing of events
- Events processed on server



- Multiple data sources
- Multiple clients over pool of connections
- Multiple backend servers on which to process same data

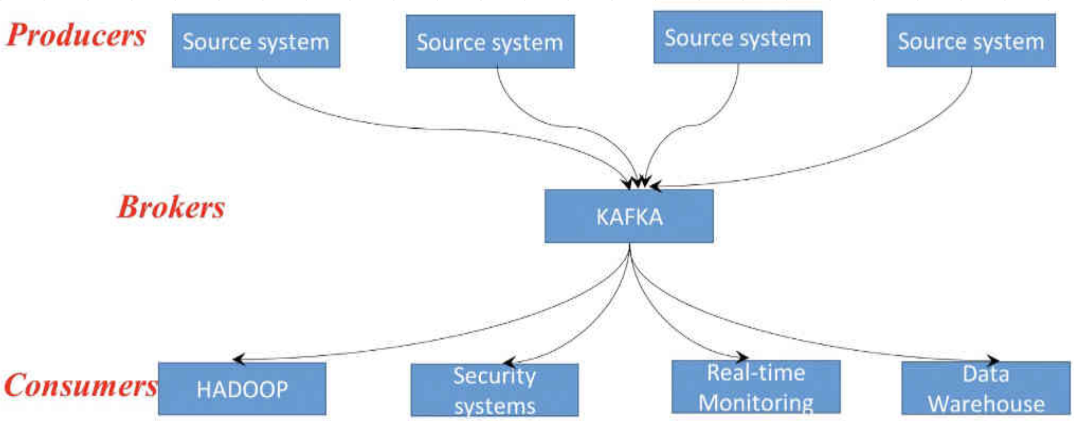


Q: Give an example of how datapipelines could be used. What are some examples of backends?



Kafka Architecture

- Publishers (producers) and Subscribers (consumers)
- Kafka is broker (decouples data pipeline)



Kafka Decouples Data Pipelines

Pub-Sub Model

Q) What does a Publisher do ..?	Q) What does a Subscriber do ..?
A. It publishes messages to the Communication Infrastructure.	A. It subscribes to a category of messages.

Role of Producer

- Defines what data it wants to send
- Publishes onto communication infrastructure
- Also called publisher

Role of Consumer

- Tells communication infrastructure what type of messages it wants to receive
- Does not specify whom to receive message from
- Messages delivered to consumer by communication infrastructure
- Also called subscriber

Role of Communication Infrastructure

- Routing
  (a) Topic-based
  (b) Content-based

(a) Topic-based routing

- Pub: send messages with topic labels
- Sub: subscribe to topics, receive all messages on that topic
- Eg: subscribe to all fire sensors in b block

(b) Content-based routing

- Sub: define matching criteria, receive all messages that match criteria
- Eg: Subscribe to ads that feature Virat Kohli
- Not supported by Kafka
- Pattern-based supported by Kafka
  - wildcard expression for a topic
  - Eg: topics with *ipl*

Pros and Cons of Communication

Pros

- No hard-wired connections between pub & sub
- Flexible: easy to add/remove pub and sub

Cons

- Design and maintenance of topics
- Performance overhead due to communication infrastructure (one extra hop)

Q: Consider a bookstore portal with various activities such as

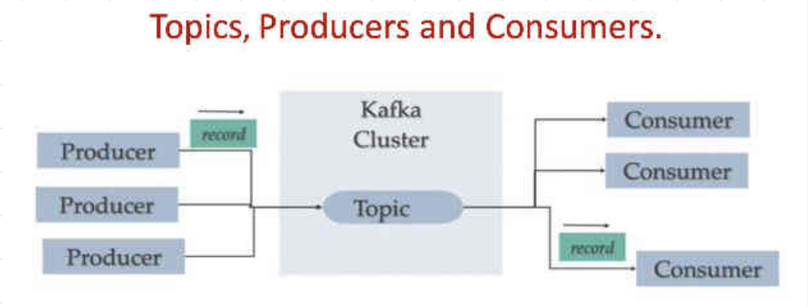
Login
List books
Get book details
Buy book
Check status of order Return book
Logout

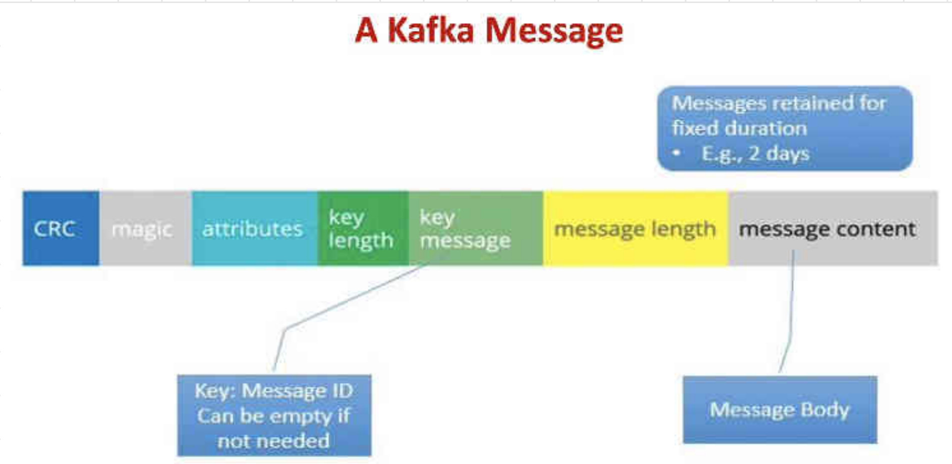
Assume we have 3 backend modules

Security
Order processing
Book information

(a) Would you use a topic-based or content-based system?
(b) What would be the topics / content?

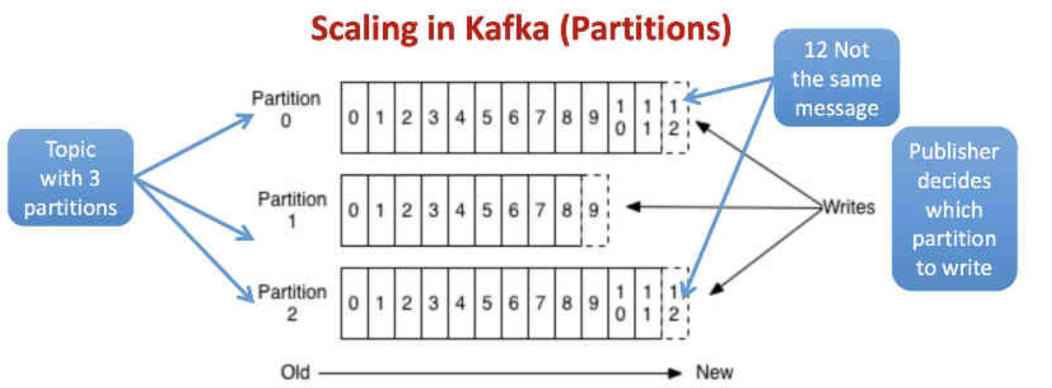
(a) Topic-based
(b) each msg can be a topic





Scaling in Kafka

- Each kafka server responsible for a certain topic (avoid bottleneck)
- What if one topic too big for single server?
  - Partitions for a topic



- Partitions allow
  - logs > disk size
  - throughput > single server
- Distributed over servers
- How to partition?
  - Round Robin
  - based on key (hash)

Example

weather — 3 partitions (buckets)

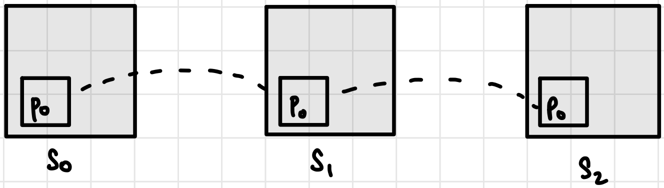
cricket — 2 partitions (buckets)

- Producer sends to partition?

Fault Tolerance in Kafka

Q: How can reliability be guaranteed in Kafka?
Hint: How does HDFS guarantee reliability?

- Redundancies across partitions



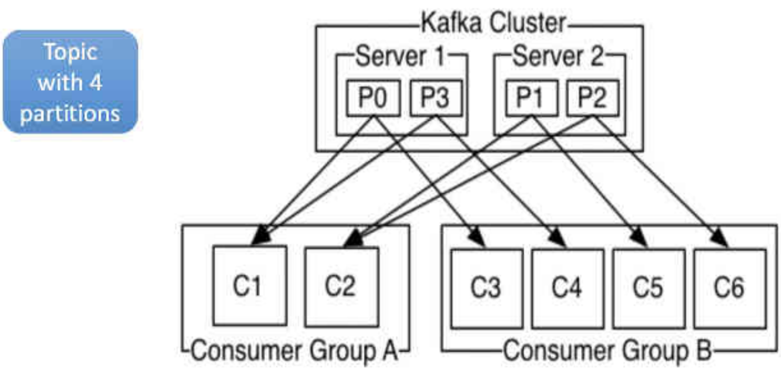
- Must be real-time (cannot wait to make copies)

Kafka

- Partitions replicated
  - leader : all reads, writes
  - followers: replicate

- Durability levels
  - Sync : after quorum writes
    - quorum = 2, replicas = 3 ⇒ if 2/3 replicas made
    - quorum = min no of replicas for the write to succeed
    - quora need to replicate
  - Async
    (i) 0 = leader only (check with leader if data received)
    (ii) -1 = no write
    - Possible loss of data (if leader fails)
    - Leader's responsibility to ensure followers are replicated (no guarantee)

Message Delivery to Consumers



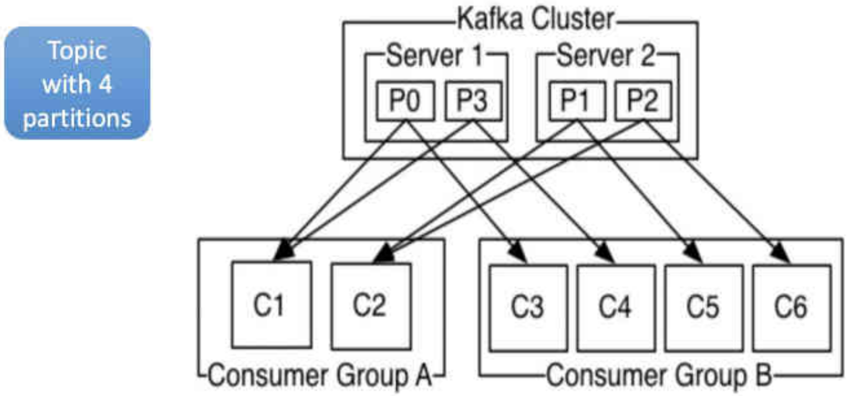
Consumer group

Typically multiple instances of an application

Partition delivers message to ONE of the group members

Load balancing

Q: In the below configuration, how is the load balanced over all the instances?



Group A

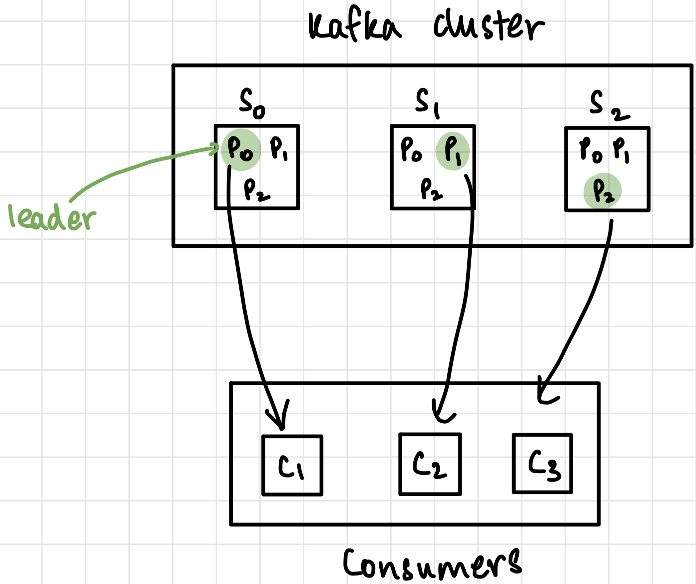
C1 assigned P0, P3
C2 assigned P1, P2

Group B

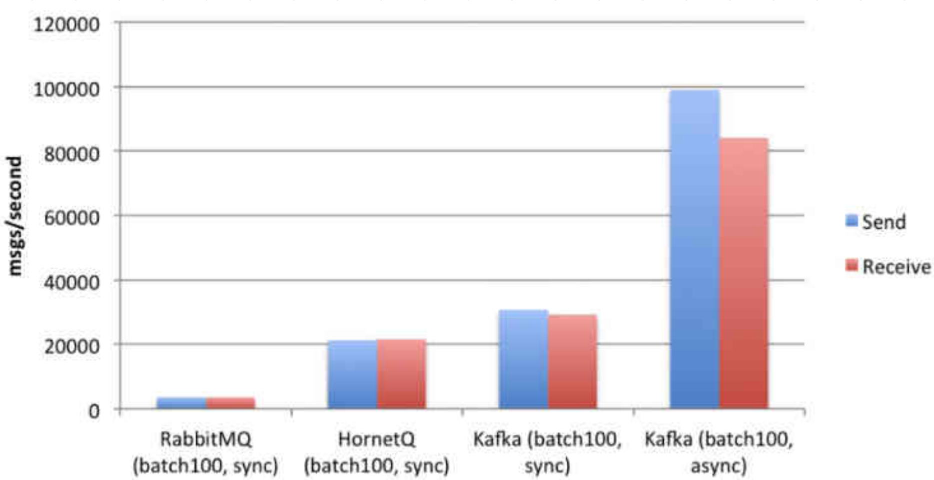
C3 assigned P0
C4 assigned P1
C5 assigned P2
C6 assigned P3

Q: Suppose we have a Kafka system
- 1 topic
- 3 servers
- 3 partitions
- 3 replicas per partition
- Consumer group with 3 instances

Draw a diagram showing
- Servers
- Partitions
- Consumer instances
- Partition assignments



Kafka Performance



I/O

- Sequential reads by consumer
- sequential writes by producers

Zero-Copy I/O

- DMA
- No copy from kernel to user

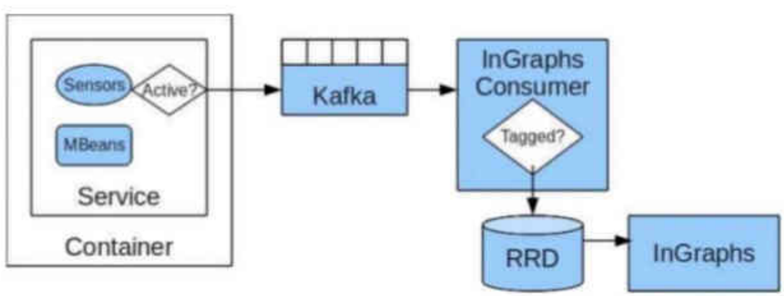
Usage of Kafka

LinkedIn: Activity data and Operational metrics.

Twitter: Uses it as part of Storm stream Processing infrastructure.

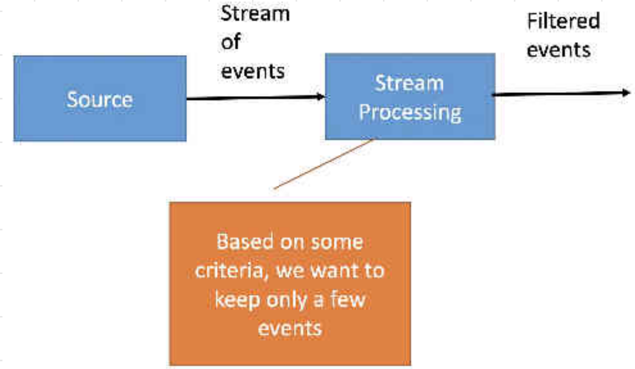
Square: Kafka as bus to move all system events to various Square data centers (logs, custom events, metrics, and so on). Outputs to Splunk, Gtaphite,Esper-like alerting systems.

Spotify, Uber, Tumbler, Goldman Sachs, PayPal, Box, Cisco, Cloud Fatr, DataDog, LucidWorks, MailChimp, Netflix, etc.



Streaming Algorithms

- Stream processing: processing of events in never-ending stream
- Need to store summaries



Approach 1

- Breakup stream into window of events
- Apache spark — relational operations on a window
- Summary of each window of size n

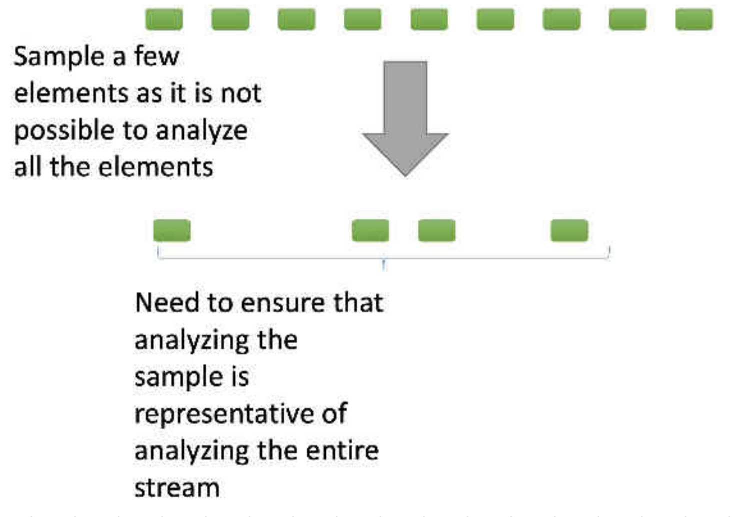
Issues

- Velocity of stream
  * diff rates for diff streams
  * instantaneous decisions
- no. of streams
  * stress in memory
- cannot store on disk
  * too slow

- need approximate solution, not exact
- often use hashing to introduce randomness

SAMPLING ALGORITHMS

- Given long stream of elements, pick representative sample
- Eg: search engine: what fraction of the typical user's queries were repeated over the past month?

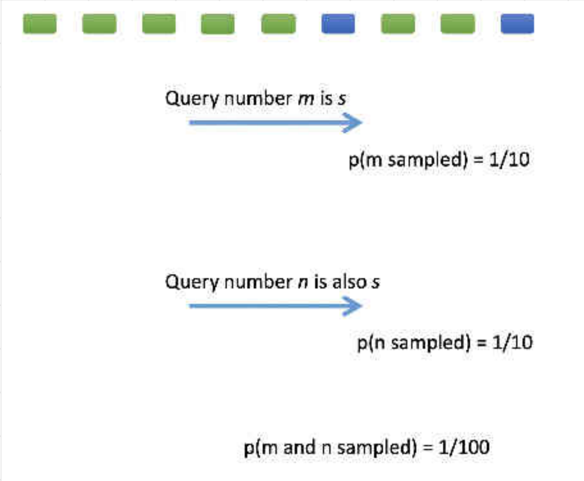


Obvious Algorithm

- For every stream tuple, generate a random number [0,9]
- If value == 0, store the tuple. Otherwise discard

Flaw in Obvious Algorithm

- Probability of duplicate query = $\frac{1}{100}$



Refined Algorithm

- Sample $1/10^{th}$ of users

- Check if user in sample
  - If they are, add query
  - If not, assign number in [0,9] to user and add if number = 0

- Hash the username to a number from 0 to 9
  - if 0, select

- No need to search the entire data structure

- GIGO: garbage in, garbage out — analysis
  - must give clean data for clean output

Generalisation

- Key components of query (here, user)
- Prev: <user, query, time>
- Hash key components in the range (0,b)
- To get sample size a/b, select query if hash(key comp) < a

Q: Suppose we want a sample dataset to debug a program that profiles transactions by user and country
How would I generate a 1/20 sample?

key component: user, country

map hash(user, country) $\rightarrow$ [0, 19]

if hash(user, country) == 0, select

FILTERING ALGORITHMS

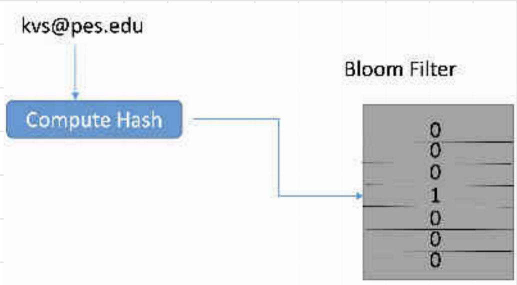
- Filter events based on data
- Eg: stream of emails, remove all spam emails
- Constraints
  - 1 GB memory
  - 1 billion well-known non-spam emails
  - 20 bytes/email address
- Cannot store on disk

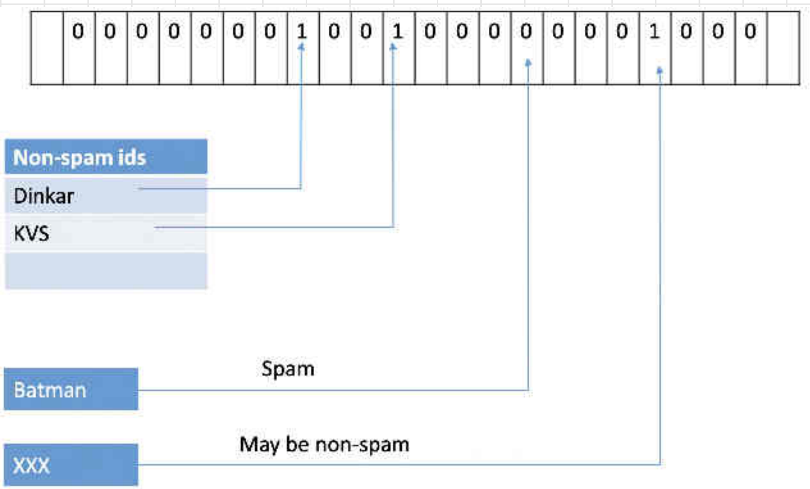
BLOOM FILTER

- 1 GB memory : 8×10^9 = 8 billion bit string

- Bloom filter initialisation
  - Hash non-spam email ids to $[0, 8\times10^9-1]$
  - set corresponding bits to 1

- Usage
  - Hash incoming email ID
  - Check bloom filter entry
  - If 0, definitely not seen before → spam
  - If 1, not sure if seen before (hash collision)





GENERAL BLOOM FILTERS

- Bloom filter consists of
  - array of n bits (size of memory)
  - collection of k hash functions $h_1, h_2, \ldots, h_k$
  - set S of keys with m elements (known non-spam)

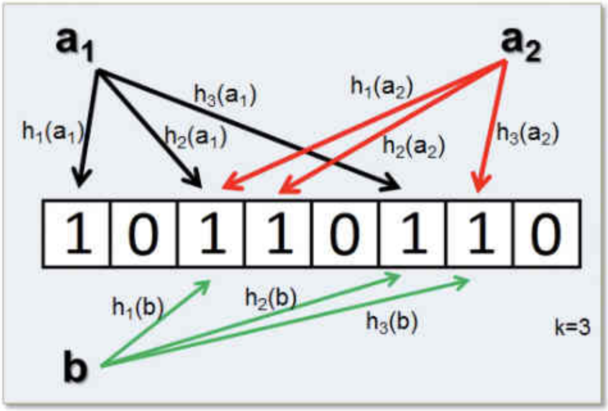
- Purpose: given a key a, determine if it is in S

- Initialisation
  - for all keys in S
  - compute all k hash functions
  - set corresponding bits to 1

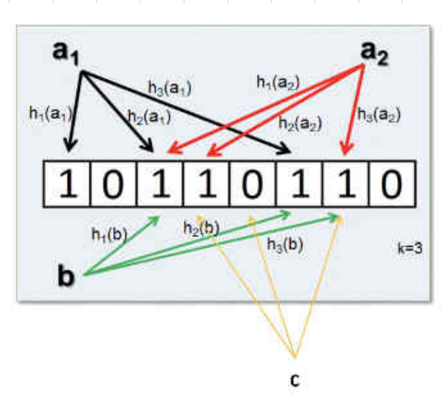
- Usage
  - hash incoming key with all k hash functions
  - if all corresponding bits are 1, known non-spam

- Chance of false positive $(1 - e^{-km/n})^k$
  derivation : RI, page 141

Eg: top – insertion, bottom – check



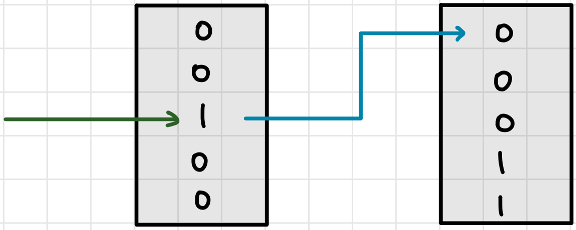
Q: Is c spam or not spam or possibly spam ?



c - spam (one 0)

Extensions

- Use secondary storage
- Cascading bloom filters
  - 2 bloom filters in series
  - If bit is 1, use second BF



COUNTING DISTINCT ELEMENTS

- No of distinct users visiting a website

Flajolet Martin Algorithm

- Pick hash function bigger than set to be hashed
- Eg: for counting IP addresses, hash > 4 billion
  for counting URLs, use 64 bits

Basic Property

- Tail length for hash function: no. of 0's at end of the hash for a given hash function
  - eg: 11111001000 tail length = 3
- Hash each element in stream
- Let R = max tail length of all bit strings
- 2^R is approximately the number of distinct elements seen

Q: Count no. of user IDs that visit a webpage using mid square hash

ID sequence : 10, 10, 7, 10, 6, 14, 14, 12, 6, 5, 7

Mid Square hash: cube user ID, make 12 bits, take middle 6 bits

10 — 1010
7 — 0111
6 — 0110

14 — 1110
12 — 1100
5 — 0101

do not use directly

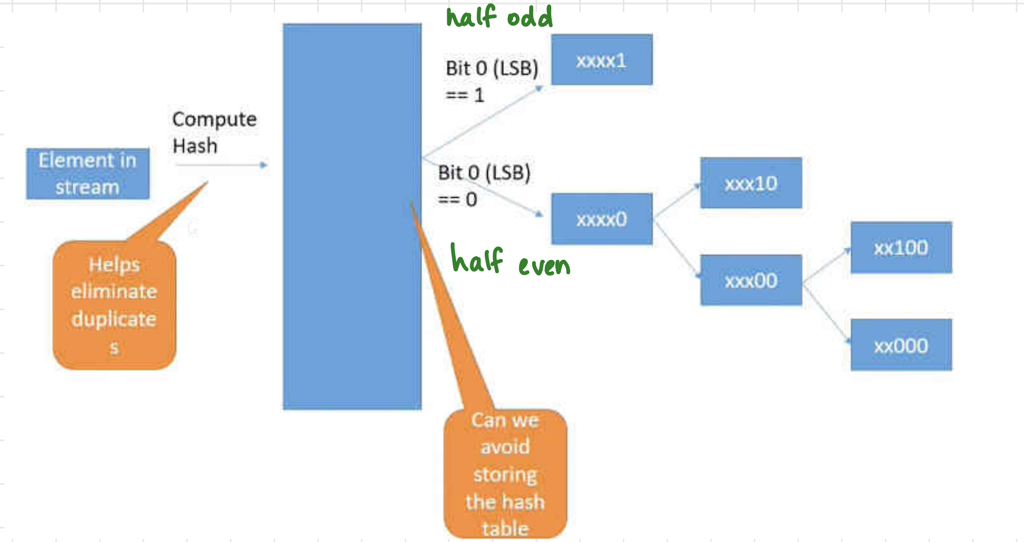
$10^3 = 1000 =$ 001	1 1110 1	000
$7^3 = 343 =$ 000	1 0101 0	111
$6^3 = 216 =$ 000	0 1101 1	000
$14^3 = 2744 =$ 101	0 1011 1	000
$12^3 = 1728 =$ 011	0 1100 0	000
$5^3 = 125 =$ 000	0 0111 1	101

middle 6

max tail length = 3

distinct users = $2^3 = 8$

Working of Algorithm



- $P(\text{hash}(a) \text{ ends in at least } r \text{ 0's}) = \frac{1}{2^r} = 2^{-r}$

- Suppose hash $= h_1 h_2 \dots h_n$

- $P(h_n = 0) = \frac{1}{2}$

  $P(h_{n-1} h_n = 00) = \frac{1}{2} \times \frac{1}{2} = 2^{-2}$

  $P(h_{n-r+1} \dots h_n = 00\dots0) = 2^{-r}$

Q: $P(\text{tail length is } r) = 2^{-r}$

If there are m elements in the stream, P(none have tail length r) = ?

$P(\text{none have TL} = r) = (1-2^{-r})^m \approx e^{-mx}$ where $x = 2^{-r}$

This estimate makes intuitive sense. The probability that a given stream element a has $h(a)$ ending in at least r 0's is 2^{-r}. Suppose there are m distinct elements in the stream. Then the probability that none of them has tail length at least r is $(1-2^{-r})^m$. This sort of expression should be familiar by now. We can rewrite it as $\left((1-2^{-r})^{2^r}\right)^{m2^{-r}}$. Assuming r is reasonably large, the inner expression is of the form $(1-\epsilon)^{1/\epsilon}$, which is approximately $1/e$. Thus, the probability of not finding a stream element with as many as r 0's at the end of its hash value is $e^{-m2^{-r}}$. We can conclude:

1. If m is much larger than 2^r, then the probability that we shall find a tail of length at least r approaches 1.

2. If m is much less than 2^r, then the probability of finding a tail length at least r approaches 0.

- $m >> 2^r$, $mx = \frac{m}{2^r} >> 0$

  $\therefore e^{-mx} \approx 0$

  $\therefore 1 - e^{-mx} \approx 1$

- $m \sim 2^r$, $mx = \frac{m}{2^r} = 1$

  $\therefore e^{-mx} = \frac{1}{e}$

  $\therefore 1 - e^{-mx}$ = some finite probability

- $m << 2^r$, $mx \approx 0$

  $\therefore e^{-mx} \approx 1$

  $\therefore 1 - e^{-mx} \approx 0$

Flajolet Martin in Practice

elements in stream

- Simple approach
  - 1 hash function, m is always power of 2
  - pick k hash functions, estimate $m = 2^R$ for each
  - take avg or mean

- Problems
  - avg pulled towards max/outliers
  - median : estimate always power of 2

- combined approach
  - divide k hashes into groups
  - compute avg of each group (unique elements)
  - median of avgs