Real-time Analytics with Storm over Streaming Data from meetup.com

Scalable Systems for Data Science

Shilpa Chaturvedi (12899)
Md. Imbesat Hassan Rizvi (12214)

Introduction

Stream Processing, Apache Storm, Meetup

Stream Processing & it's challenges

Continuous flow of data as streams.

Real-time decisions and analytics to be performed quickly.

Streams often delivered rapidly, Velocity aspect of Big Data

Processing all streams together can consume entire available main memory

Data rates may vary considerably.

Streams are unbounded, continuous analytics.

Often approximate answers are reasonably accurate and more efficient.

Apache Storm

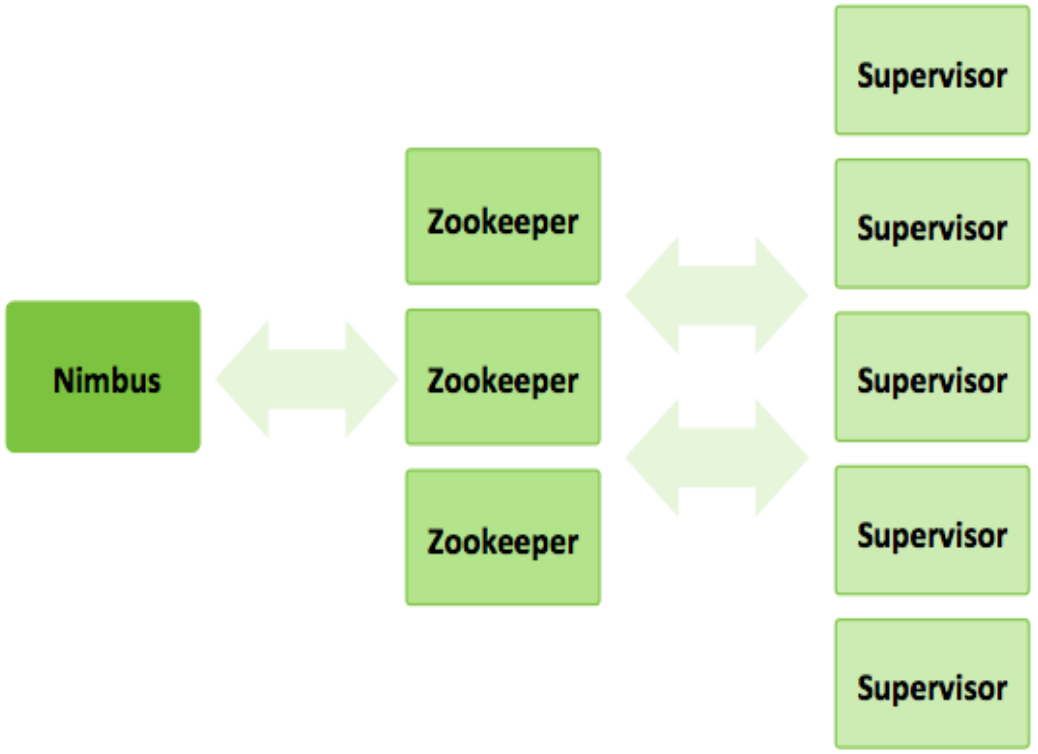
Open source, distributed real-time computation system.

Fault tolerant, easy to setup & operate and scalable.

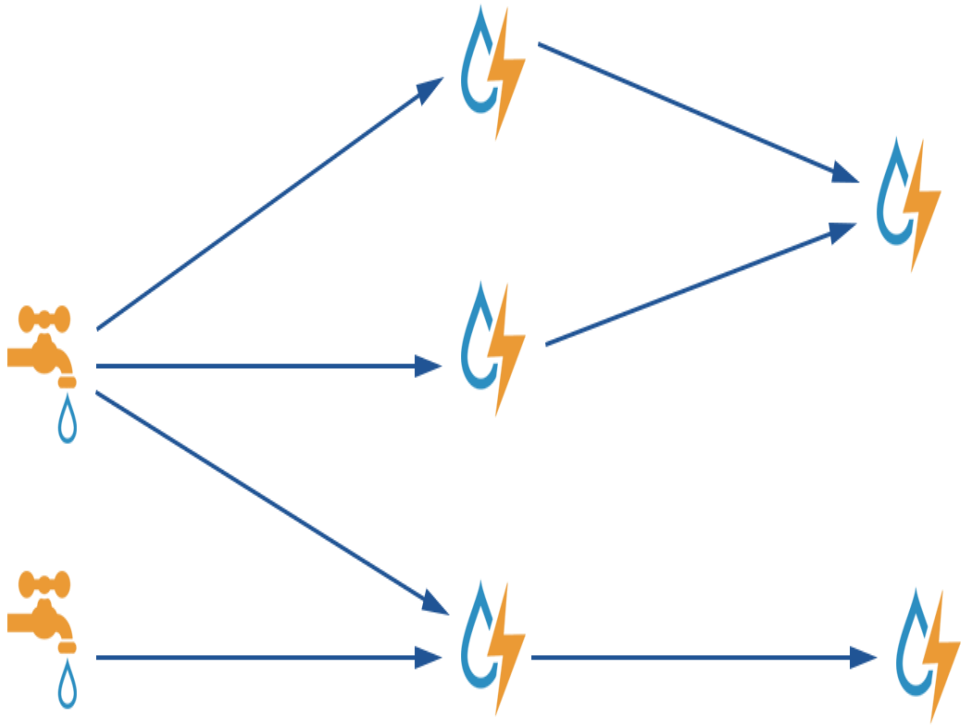
Also called Hadoop for Real-time.

Storm topology consumes streams of data and processes those streams in arbitrarily complex ways.

Apache Storm – contd…



Storm Architecture



Storm Data Flow

Meetup Platform

An online social networking platform to facilitates offline group meetings at various venues and on various categories.

Members can organize public or private events within their Meetup groups.

Anyone can browse, search for, and register to events based on their interest and locality.

It allows easy collaborations and event organization for people sharing common interests

Stream API Provided

The meetup platform provides the following streaming API ：

1. rsvps
2. open_events
3. photos
4. open_venues
5. event_comments

We are using rsvps and open_events stream in our project.

Open Event Stream

Some Major fields present

1. Event details – id , name, url , duration
2. Venue details – city, country ,
3. Payment required - In form of 0 or 1
4. Event category –For example 'tech', 'career-business', 'fitness' ,'socializing'
5. status of event
6. Group details – name , id , url of the organizing group

RSVP Stream

Some Major fields present

1. Event details – event id , name , url , time

2. Response – Yes, No

3. Member details – name , id , photo

4.Venue details – exact location of the event

5. Group details - Organizing group information like group name, id, categories tagged to group.

Distinct Count of stream elements

Estimate unique no. of elements in a data stream

Distinct Count

Distinctly different elements appeared in Stream

Our use case:

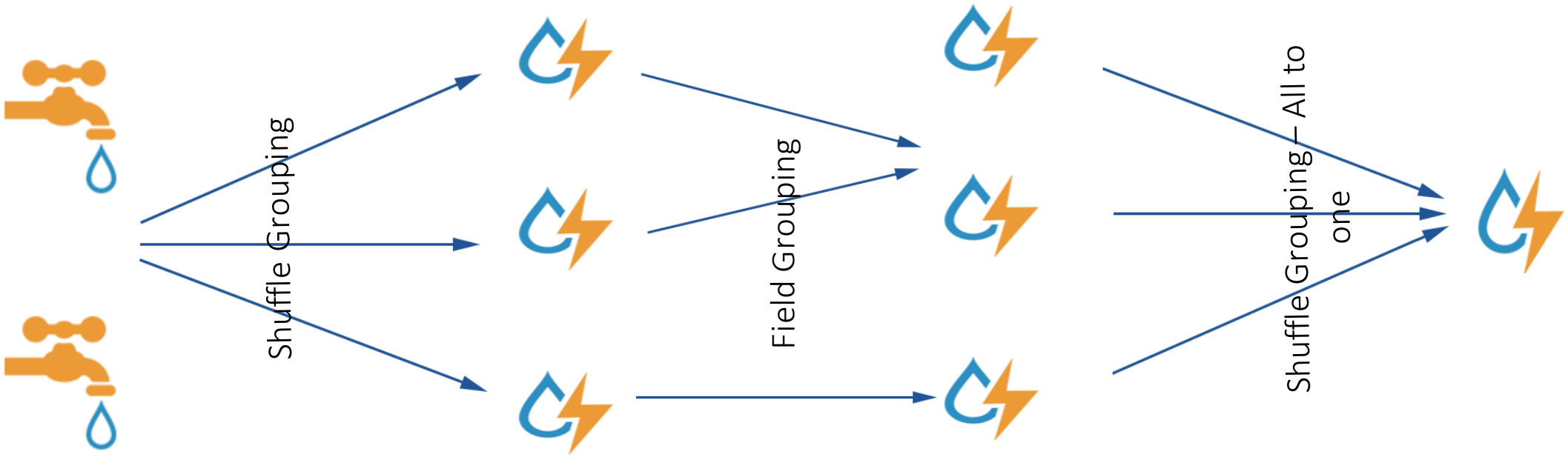
Given a location, unique number of attendees.

Attendees in same location may rsvp for different events. Avoid over-estimation.

Exact count – in Memory member id storage for look up

Probabilistic Count – Approximate estimation of unique count so far. No member id storage/lookup.

Distinct Count – Topology



Stream Gen Spout – Reads JSON stream data from HDFS files and emits tuples with fields like country, city, event name, member id etc.

Field Filter Bolts – Check against filtering criteria e.g. location and forwards fulfilling tuples

Field Counter Bolt (***HashSet or Durand-Flajolet based***) - maintains unique count of fields from incoming tuples and forwards it.

Unique Count Aggregator – Receives individual unique counts from previous level bolts and sums them up

Distinct Count – Exact Count

The topology as described before differs only at the distinct count computation step.

Create HashSet<elements> in each field count bolt.

For every incoming tuple:

If *not* HashSet.contains(element):

HashSet.add(element)

Emit(BoltIndex, HashSet.size())

Distinct Count – Durand-Flajolet Algorithm

n ⟵ num of buckets. // 1 << bucketParam

hv ⟵ HashValue(element)

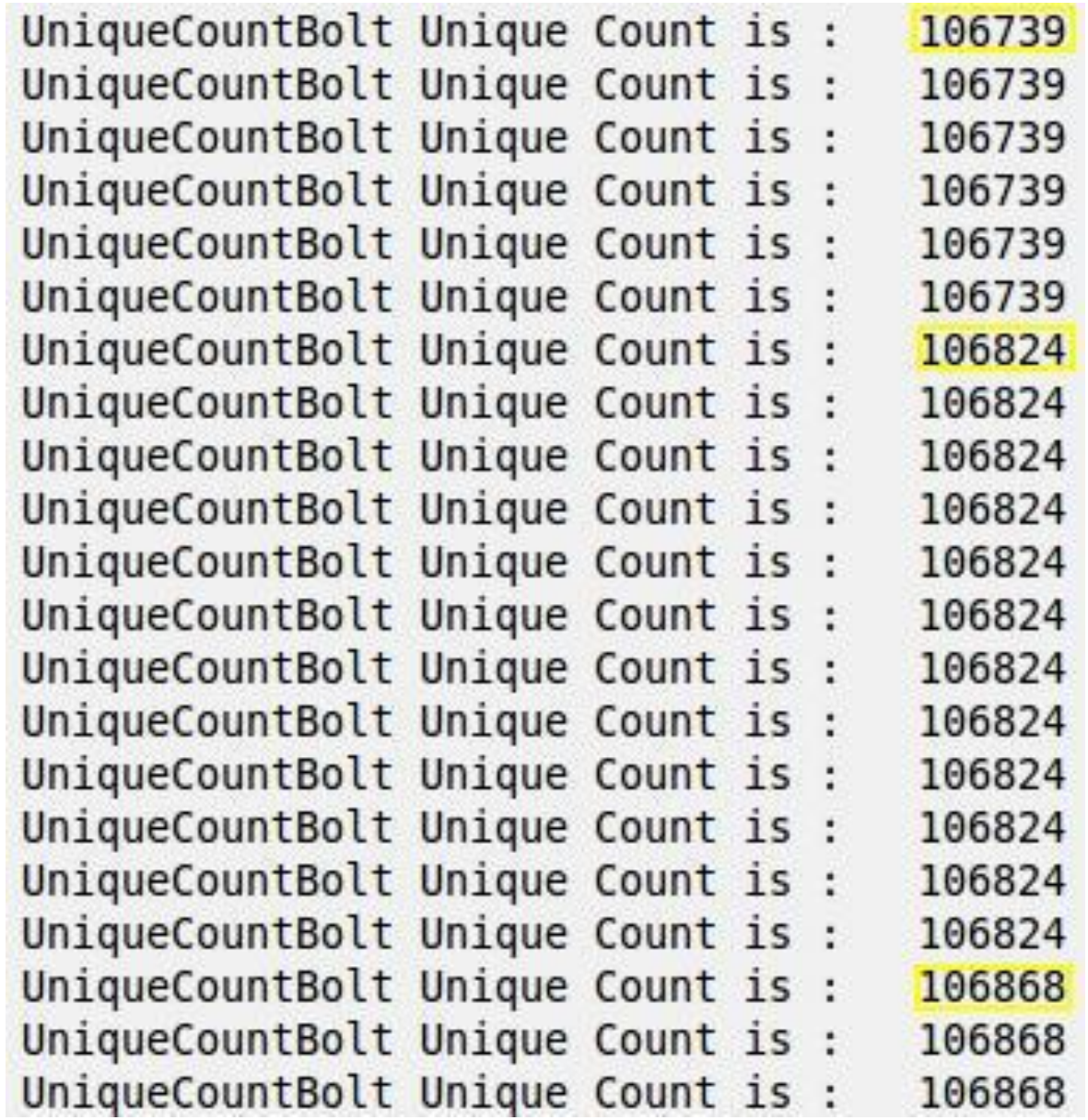
bucketId ⟵ hv & (n - 1)

modified_hv ⟵ hv >> bucketParam

maxZeros[bucketId] ⟵ max(maxZeros[bucketId], trailZeros(modified_hv))

avgMaxZeros ⟵ sum(maxZeros) / n

distinctCount ⟵ *(2^(avgMaxZeros)) x n x magicNum // magicNum = 0.79402*



Final Topology Output

This is the final output from the *"UniqueCountBolt"* aggregator bolt.

For every stream input processed, the aggregator bolt updates the distinct count and displays it (in this case writes to an HDFS file).

Upcoming events

Know upcoming events in a location on a given date

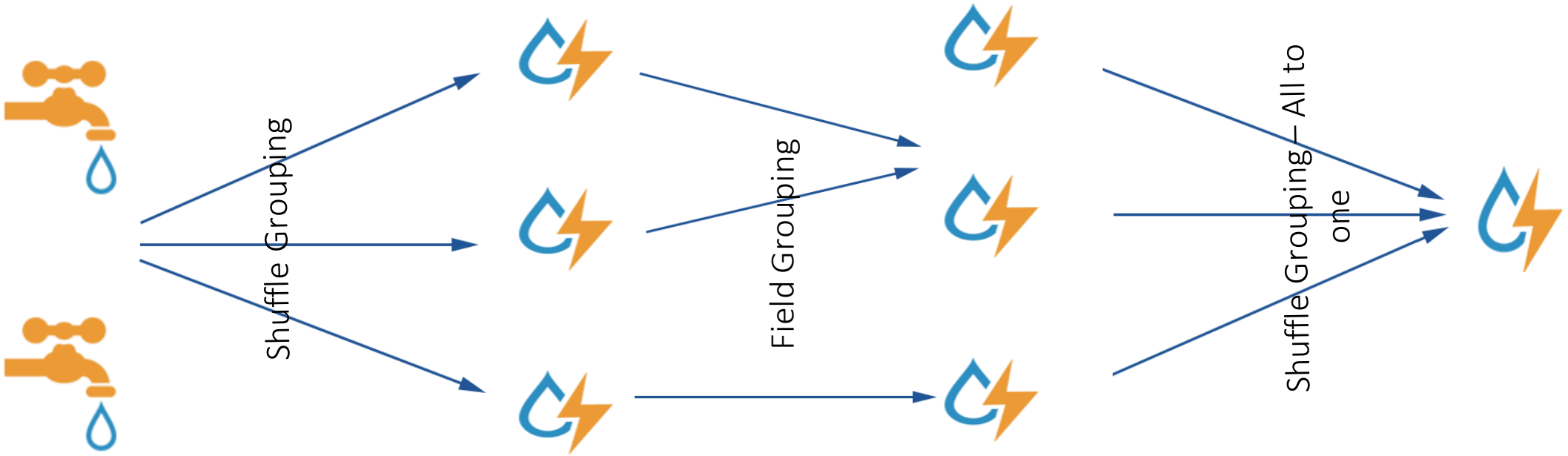
Location specific upcoming event on a given date

Motivation – We are going to visit a foreign country, hence want to know meetups happening there on a specific date or within a range of date.

Will be helpful in aligning, if possible, our schedule with the events of our interests.

The use case is much more general given a location.

Location based upcoming events - Topology



Stream Gen Spout – Reads JSON stream data from HDFS files and emits tuples with fields like country, city, event name, event time, status etc.

Field Filter Bolts, Level 1 – Checks whether incoming stream corresponds to status *"upcoming"* and accordingly forwards it.

Field Filter Bolts, Level 2 – Checks against location and whether the event name has been processed before or not and accordingly forwards it.

Upcoming Event Aggregator – Receives filtered tuples and outputs those events with event time on specified date or within a range of date

NY	Springtime Dinner in NY!	2016,04,18,22,45,00
Boston	Boston Marathon Spectator Gathering	2016,04,18,15,15,00
Reston	New 3D Printing Slicer Overview	2016,04,18,23,00,00
Valrico	Brandon Chapter Meeting	2016,04,18,22,30,00
New York	Spanish with Mi Manzana Drop-in Monday 4/18 at 4:00pm	2016,04,18,20,00,00
Taos	Update: Change in event plans for: Let's Meet Up for The Ghost of Taos	2016,04,18,00,30,00
Huntingdon Valley	Outdoor Pickkeball open play	2016,04,18,13,00,00
Atlanta	Buddhism & Psychoanalysis: Coming Together in Clinical Practice	2016,04,18,23,00,00
Arlington	Love Is In The Air	2016,04,18,23,00,00
Alexandria	Monday Night Pinochle in Springfield	2016,04,18,22,30,00
San Antonio	Monday Run at Hardberger Park	2016,04,18,23,30,00

Upcoming Events on a date – Demonstration

Location filter – Country (US)

Looking for events on date – 18.04.2016

Output:

List of city wise events with event time happening in "US" on the specified date.

Stream Sampling

Sampling and Anaytics

About Sampling

Approximate sampling is required for stream processing when the velocity of stream data is high and we cannot process all data as it arrives

Extracting reliable samples from Stream is needed such that it does not affect the overall data distribution

Some of the techniques used are hashing and creating buckets.

Generally sampling is used along with other topologies.

Randomized Sampling

We did not use hash based sampling as our intention is to check sampling accuracy across multiple fields like location, categories and payment requirement

Using hash function based approach might lead to skewed samples

In our sampling approach, every tuple is given a fair chance to be selected in the sample by generating a random number.

To obtain a fraction a/b of original data, we consider the record in sample if the random number generated is less than a.

Of course, sampling in this case will depend upon the uniformity of random number generator.

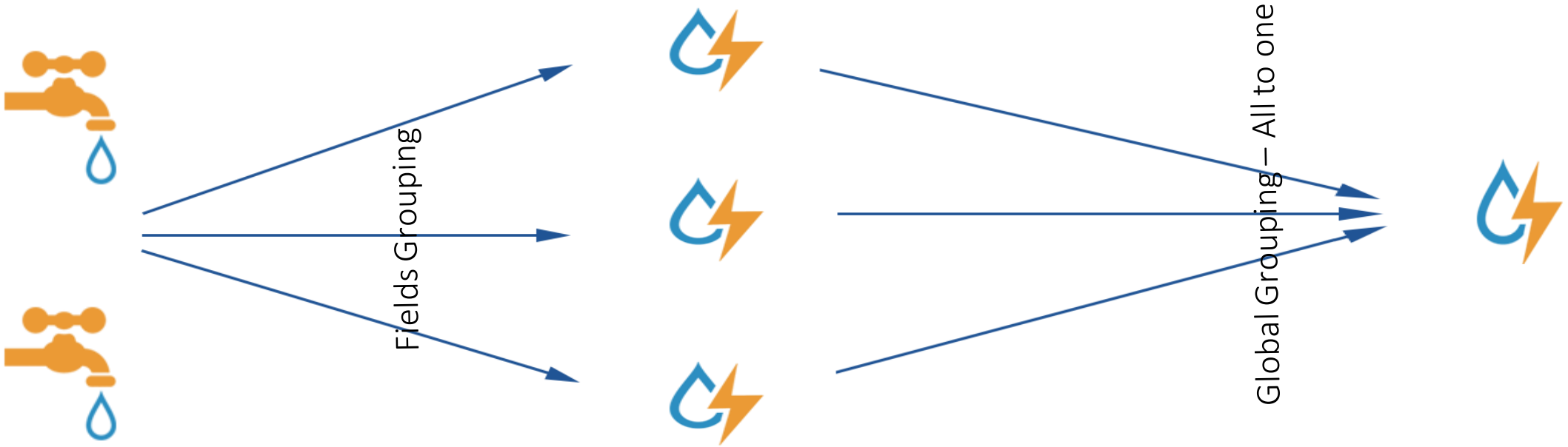
Analytics using Sampling

We have used sampling to capture the following data points from open_events stream.

- Total count of events happening in each city.
- Total tech events happening in each city
- Total number of paid events happening in each city

For the same data points, we have also calculated the exact counts to compare effectiveness of sampling.

Sampling – Topology



Open Events Spout

Counting and Sampling Bolts

Sampling Report Bolt

Results - Sampling

Accurate Data	40% Sampling	50% Sampling	60% Sampling
London	London	London	London
New York	New York	New York	New York
Los Angeles	Singapore	Los Angeles	Los Angeles
Toronto	Los Angeles	Singapore	Toronto
Singapore	Toronto	Sydney	Singapore
Chicago	Sydney	Toronto	Sydney
Sydney	Barcelona	Melbourne	Melbourne
Melbourne	Melbourne	San Francisco	Barcelona
Barcelona	Chicago	Barcelona	San Francisco
San Francisco	San Francisco	Denver	Chicago
Denver	Denver	Chicago	Denver
Washington	Washington	Seattle	Washington
Seattle	San Diego	Washington	San Diego
San Diego	Tokyo	Tokyo	Seattle
Tokyo	Seattle	Paris	Portland
Paris	Paris	Portland	Paris
Portland	Austin	San Diego	Tokyo
Austin	Portland	Austin	Austin
Hong Kong	Hong Kong	Hong Kong	Vancouver

Results – Uniform Sampling on each data field

Sampling Rate	Total Event Count	Total Tech Event count	Total Paid Event Count
Accurate	55237	4004	2241
60 percent	33238	2420	1386
50 percent	27633	1966	1104
40 percent	22008	1571	868

Actual Percentages of Data after Sampling	40% Sample	50% Sample	60% Sample
Overall Event Count	39.84	50.03	60.17
Count of Paid Events	38.73	49.26	61.85
Count of Tech Events	39.24	49.10	60.44

Bloom Filters

Probabilistic Filtering of Meetup Members

Bloom Filter - Introduction

A Bloom filter is a probabilistic data structure that can indicate quickly, in a memory-efficient manner, whether an element is present in a given set or not

A Bloom filter can tell us that the element is either "definitely is not in the set" or "may be in the set"

Driven by:

- Expected number of elements (n)
- False positive probability (fpp)
- Size of the bit vector (m)
- Number of hash functions needed (k)

Bloom Filter – Use Case

- Hypothetical scenario where we classify a set of meetup.com members as "well-behaved members" based on their historical RSVP activities
- There are some questionable users and they conduct a burst of activities once in a while, where they send RSVP responses to mislead genuine audience, or conduct a denial of service attack
- Solution: Block out members with questionable RSVP behavior
  - Check the members from the incoming bursts of RSVP stream against a prepared Bloom filter of well-behaved users
  - Only those identified as being present in the filter can be let in
  - Might still let through some of the misbehaving members, but it is guaranteed that all the genuine users' responses are honored, and a majority of the bad users are stalled

Bloom Filter – Topologies

One topology to prepare the Bloom filter on the "well-behaved users" data

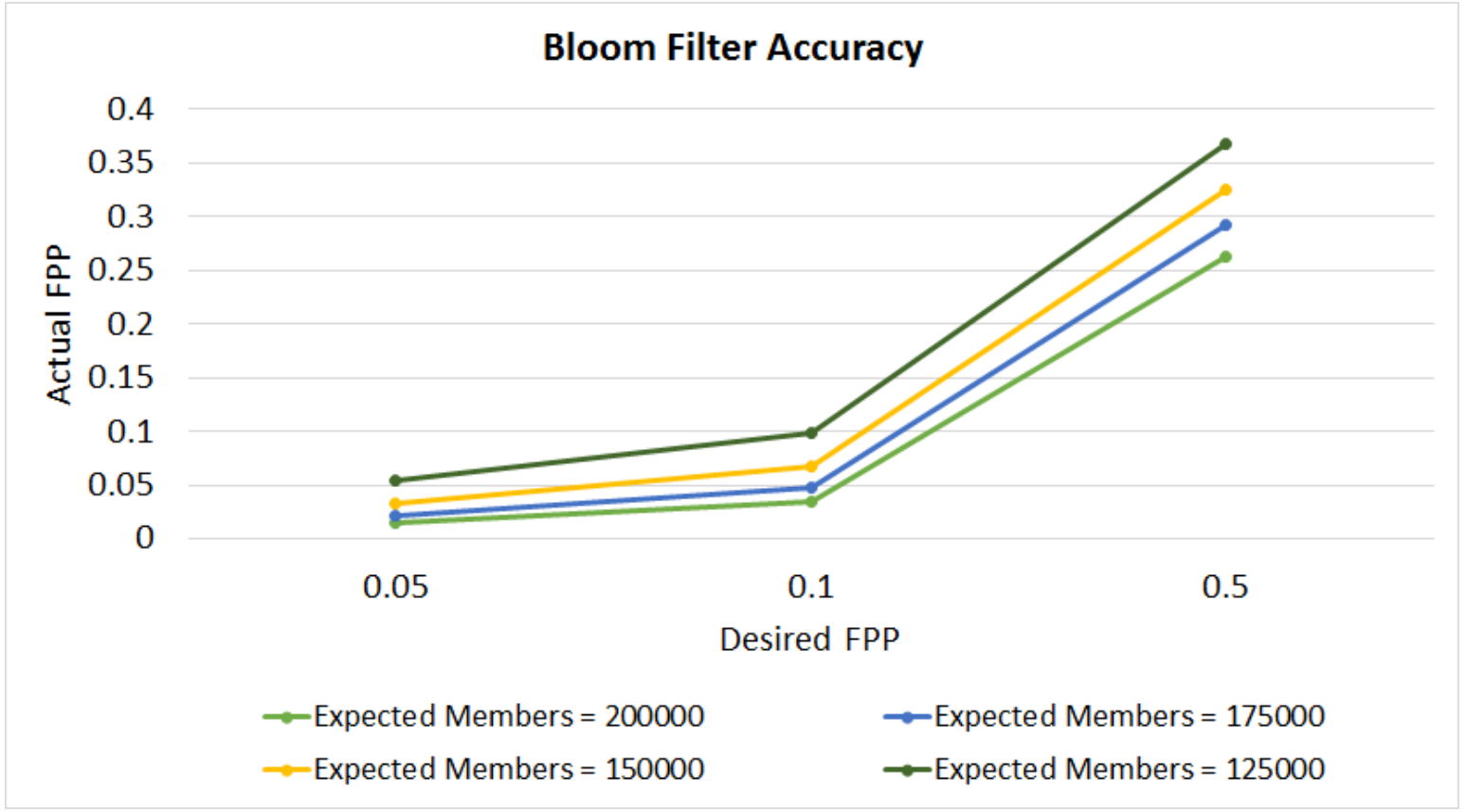
For supporting accurate count comparison, we populate both a Bloom filter and a HashMap with this data

HashMaps are proven through these experiments to be prohibitive for large data – as big as 125 times the Bloom filter in worst case examples on this data

Second topology checks for Bloom filter and HashMap membership, to bring out the actual false positive percentage (FPP)

For the example run, data considered:

- Expected count of members: 125000, 150000, 175000, 200000
- Desired FPP: 0.05, 0.1, 0.5



Desired FPP vs Actual FPP

While the Google Guava implementation of the Bloom Filter takes in desired FPP, it tries to achieve an actual FPP that is upper-bounded by the desired FPP. The plots show this, for varying number of expected members and desired FPP data.

Trending Events

Calculating Real Time Trends

Trending Events

Inspiration came form Twitter Trending Tags

Event A to be more popular than Event B (for a given time span) if Event A has received more positive RSVPs than Event B.

The logic for trending is the implementation of a sliding window and ranking the events based on the aggregate count of event windows

The window is periodically advancing to consider more recent data and leave old data behind. While doing so, it is also emitting the trending events.

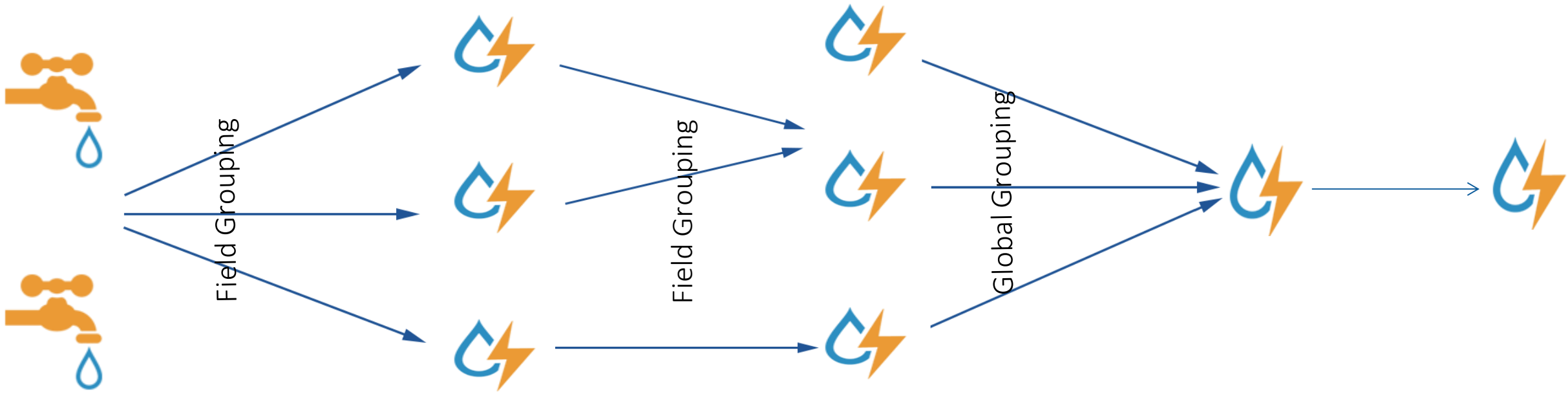
Sliding Window

A queue is maintained for every event, and data collected at some interval for each event is pushed to the queue.

A window of size say "*m*" is sliding on top of queue and the interval counts under the window are aggregated and emitted. After emitting, the window advances and removes the least recent data out of window.

We can represent sliding window count for an event at any point of time, say "*t*", as sum of interval counts from time count at time "*t"* to count at time "*t+m"*

Topology



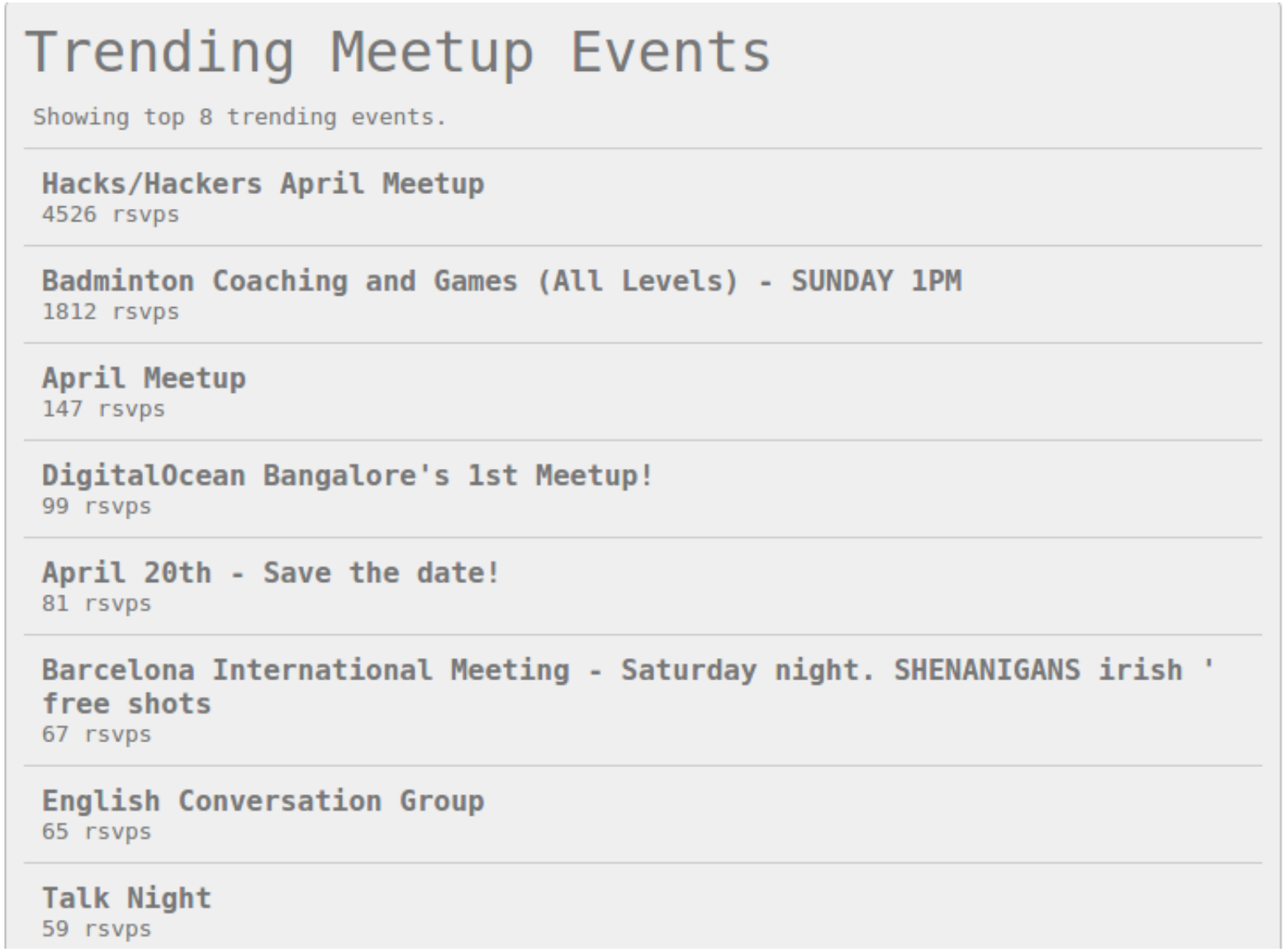
RSVP Stream Spout

Count Bolts

Aggregation bolts with sliding window

Ranking Bolt, emits TopN events

Report Bolts writes to redis/file system



Data Observations

This is how Top N trending items look when used with Redis for storing and continuous updating trending items. The basic interface is done as a web app written in python and JavaScript.

Counting 1s in a Window

Datar – Gionis – Indyk – Motwani (DGIM) Algorithm

DGIM Algorithm - Motivation

Given a window of size **N**, can we answer queries of the form **How many 1s are in the last k bits?** where interval size, $k \leq N$

Can be achieved by storing all the previous N bits of the window. But this N could be prohibitively large

If we are happy with an approximate answer, the DGIM algorithm can give this count by only storing $O(\log \log N)$ bits per stream

DGIM algorithm gives approximate answer, and is never off by more than 50%

DGIM Algorithm - Details

Divide the window into **buckets**, consisting of:

- The timestamp of its right (most recent) end.
- The number of 1's in the bucket. This number must be a power of 2.

Six rules to be followed when representing a stream by buckets:

- The right end of a bucket is always a position with a 1.
- Every position with a 1 is in some bucket.
- No position is in more than one bucket.
- There are one or two buckets of any given size, up to some maximum size.
- All sizes must be a power of 2.
- Buckets cannot decrease in size as we move to the left (back in time).

DGIM Algorithm - Operations

Update Buckets

- When a new bit comes in, drop the oldest bucket if it is completely outside the window
- If the current bit is 1:
  - Create a new bucket of size 1 for this bit; End timestamp = current timestamp
  - If there are now more than two buckets of size 1, combine the oldest two into a bucket of size 2
  - Continue above step till the there are number of buckets of the same size ≤ 2 all the way

Get Approximate Number of 1s

- Sum the sizes of all the buckets but the last one
- Add half the size of the last bucket

DGIM Algorithm – Use Case

Count the number of "Yes" RSVPs within a specified window, for a given interval size

Capturing this information regularly could help meetup.com analyze peak traffic periods in the day.

This could help them decide:

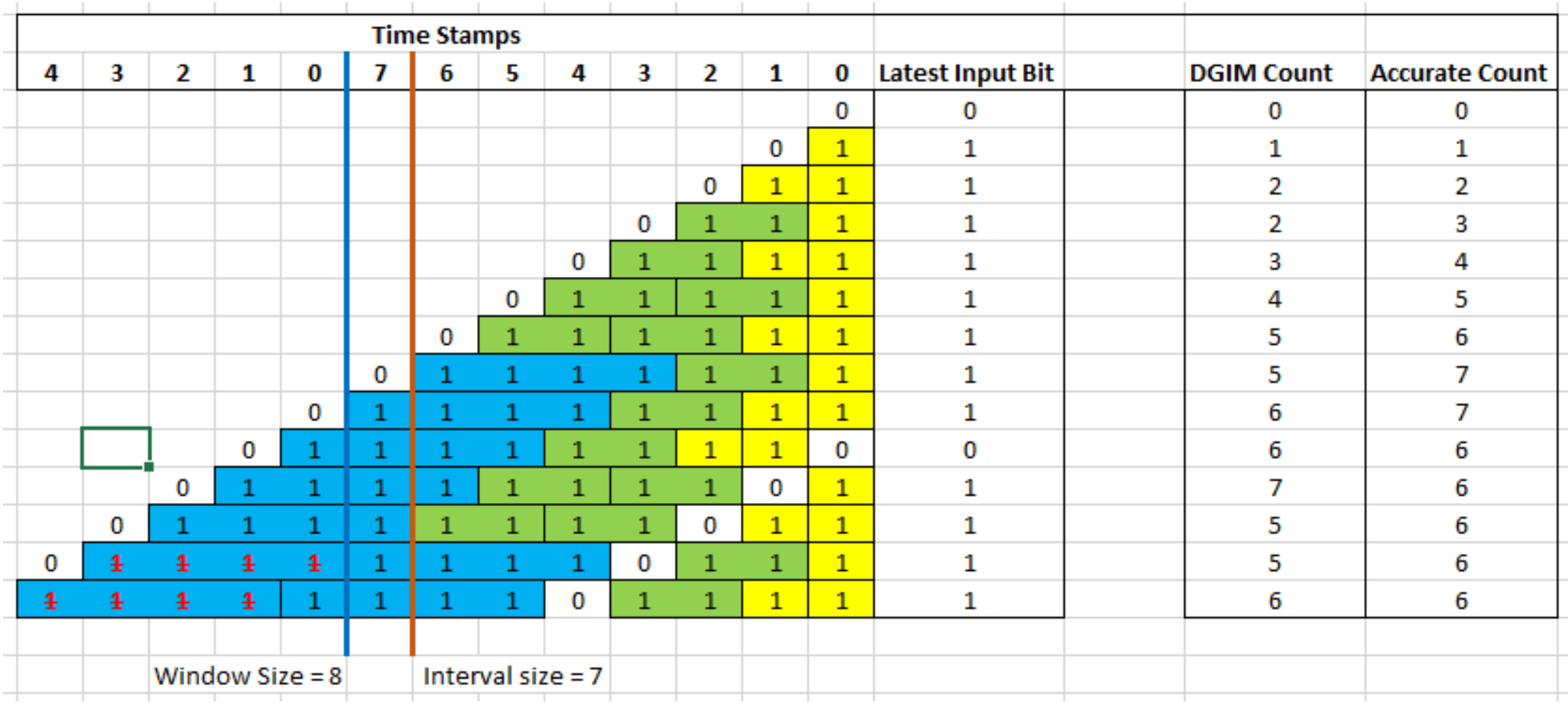
- scaling of their systems for peak load
- device marketing strategies where the usage is low

Get DGIM count, as well as corresponding accurate count and establish the error rate in DGIM approach, for the chosen window and interval sizes

Experiments conducted:

- Keep window size (N) constant and vary interval size (k)
- Keep interval size (k) constant and vary window size (N)

DGIM Algorithm – Example Run

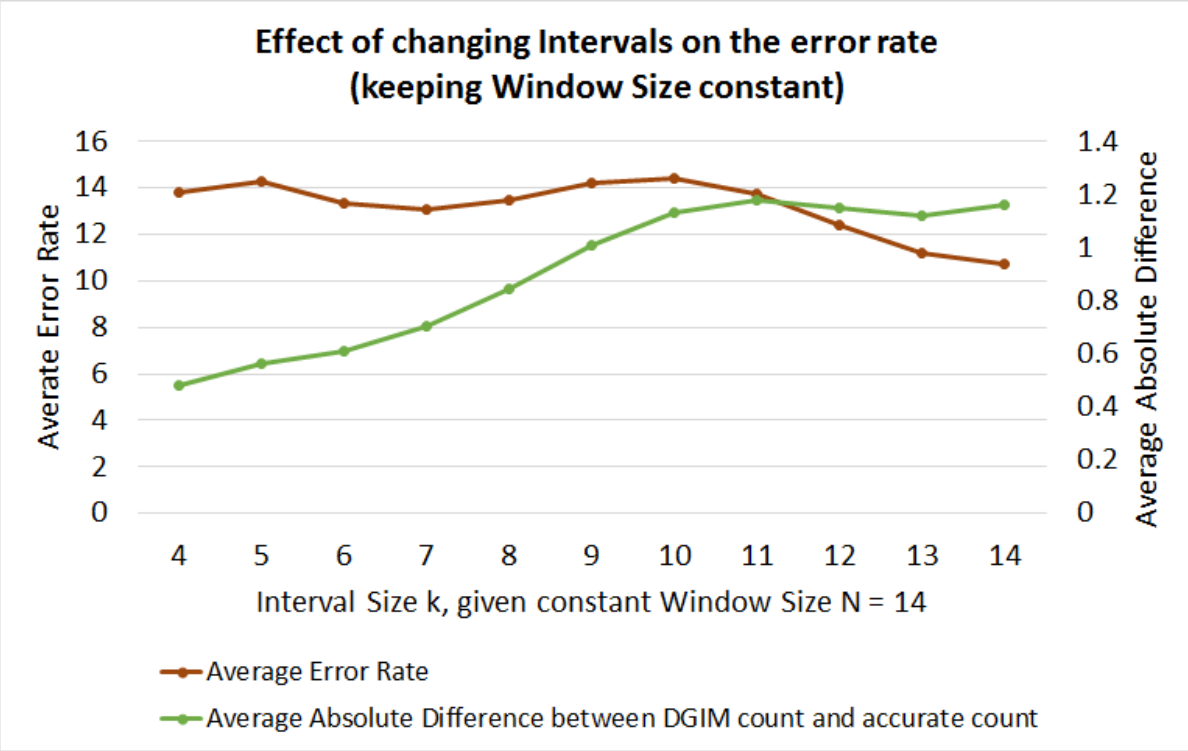


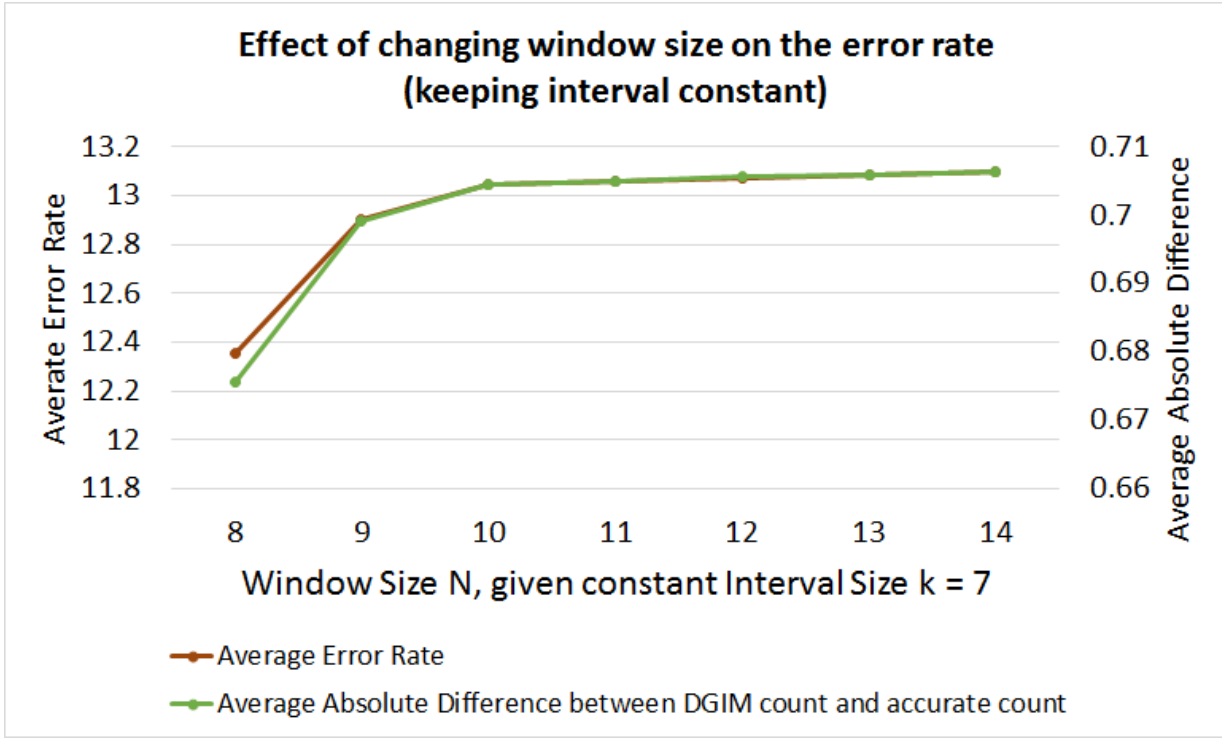
Time Stamps															
4	3	2	1	0	7	6	5	4	3	2	1	0	Latest Input Bit	DGIM Count	Accurate Count
												0	0	0	0
											0	1	1	1	1
										0	1	1	1	2	2
									0	1	1	1	1	2	3
								0	1	1	1	1	1	3	4
							0	1	1	1	1	1	1	4	5
						0	1	1	1	1	1	1	1	5	6
					0	1	1	1	1	1	1	1	1	5	7
				0	1	1	1	1	1	1	1	1	1	6	7
			0	1	1	1	1	1	1	1	1	0	0	6	6
		0	1	1	1	1	1	1	1	1	0	1	1	7	6
	0	1	1	1	1	1	1	1	1	0	1	1	1	5	6
0	1	1	1	1	1	1	1	1	0	1	1	1	1	5	6
1	1	1	1	1	1	1	1	0	1	1	1	1	1	6	6

Window Size = 8

Interval size = 7

DGIM Algorithm – Error Rate Behavior





Domain Analytics

Dive into Meetup Streams

City level Interest Groups based on events created

In this topology, we have attempted to find the inclination of each city for different categories of events, based on the number of events getting created in each city under each category.

This data can be useful in identifying not just the interest areas of a particular city, but even deriving what diverse a category of events happen in that city.

For this, we have used open_events stream to get number of events happening, on per city and per category basis. The topology gives data patterns for each city appearing in the open_events stream

City Interest Group Based on RSVPs

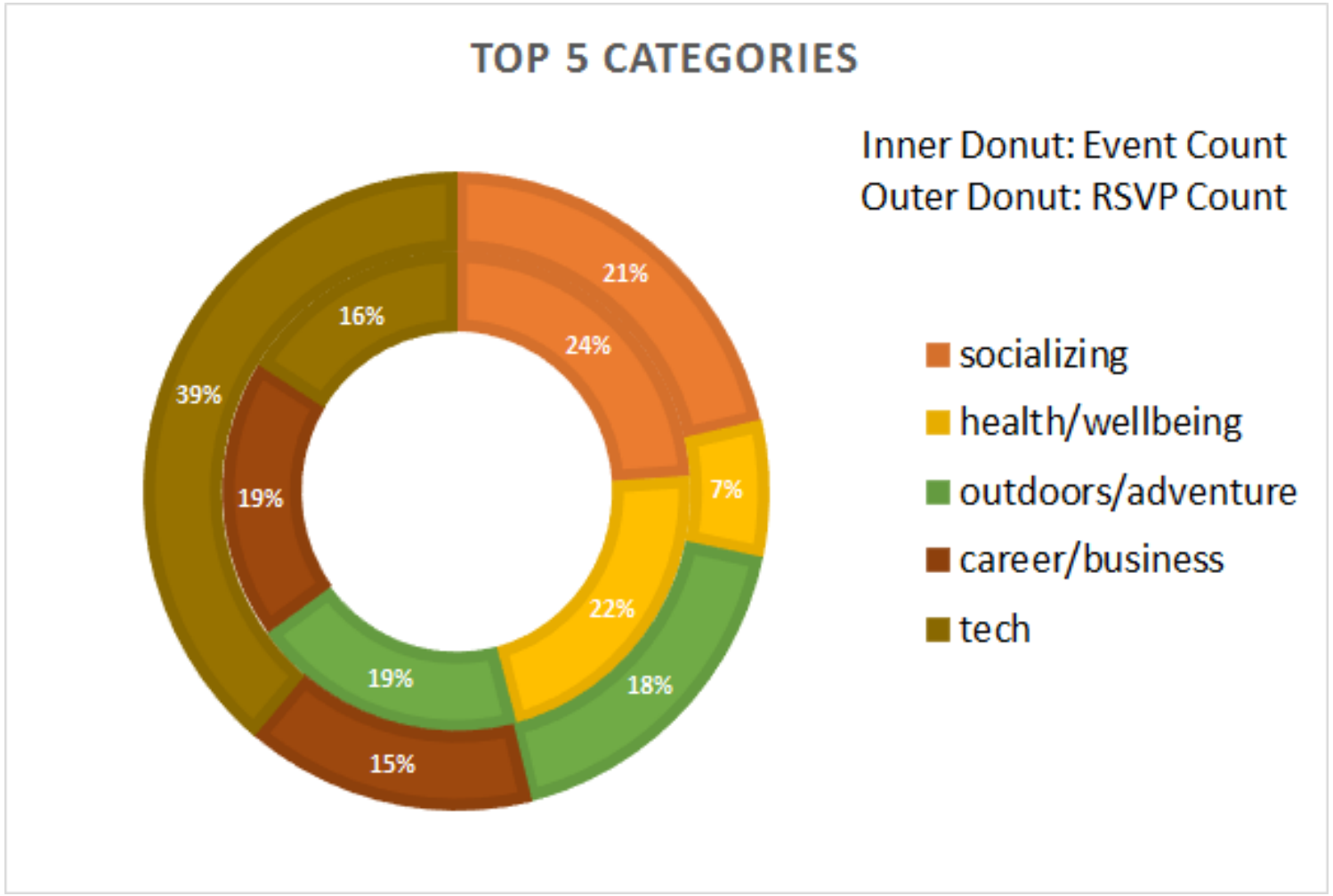
Commenting on a city's interest area on the basis of events getting created, may not be sufficient

For this, one approach could be to count the number of RSVPs for each event and map this numbers to the corresponding categories of the events

This could give a better picture of interest of people in each city

IT is not just about the events getting generated, it is about those events getting RSVP responses

For this, we have used the RSVPs stream to get the number of 'Yes' RSVPs on events and then aggregating counts of 'Yes' RSVPs across each category of events.

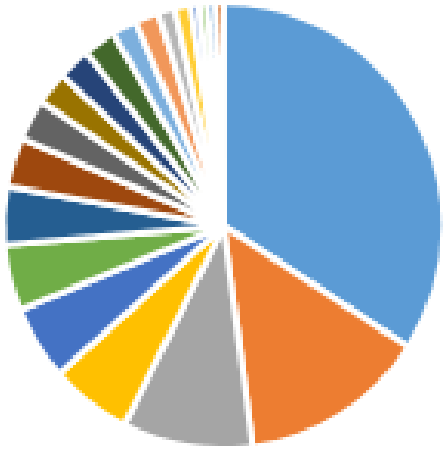


Data Observations

Results showing interest groups based on events counts and rsvps received .

Bangalore

- tech
- career/business
- outdoors/adventure
- sports/recreation
- movies/film
- games
- socializing
- photography
- education/learning
- health/wellbeing
- writing
- food/drink
- community/environment
- new age/spirituality
- music
- fine arts/culture
- religion/beliefs
- fitness
- cars/motorcycles
- language/ethnic identity



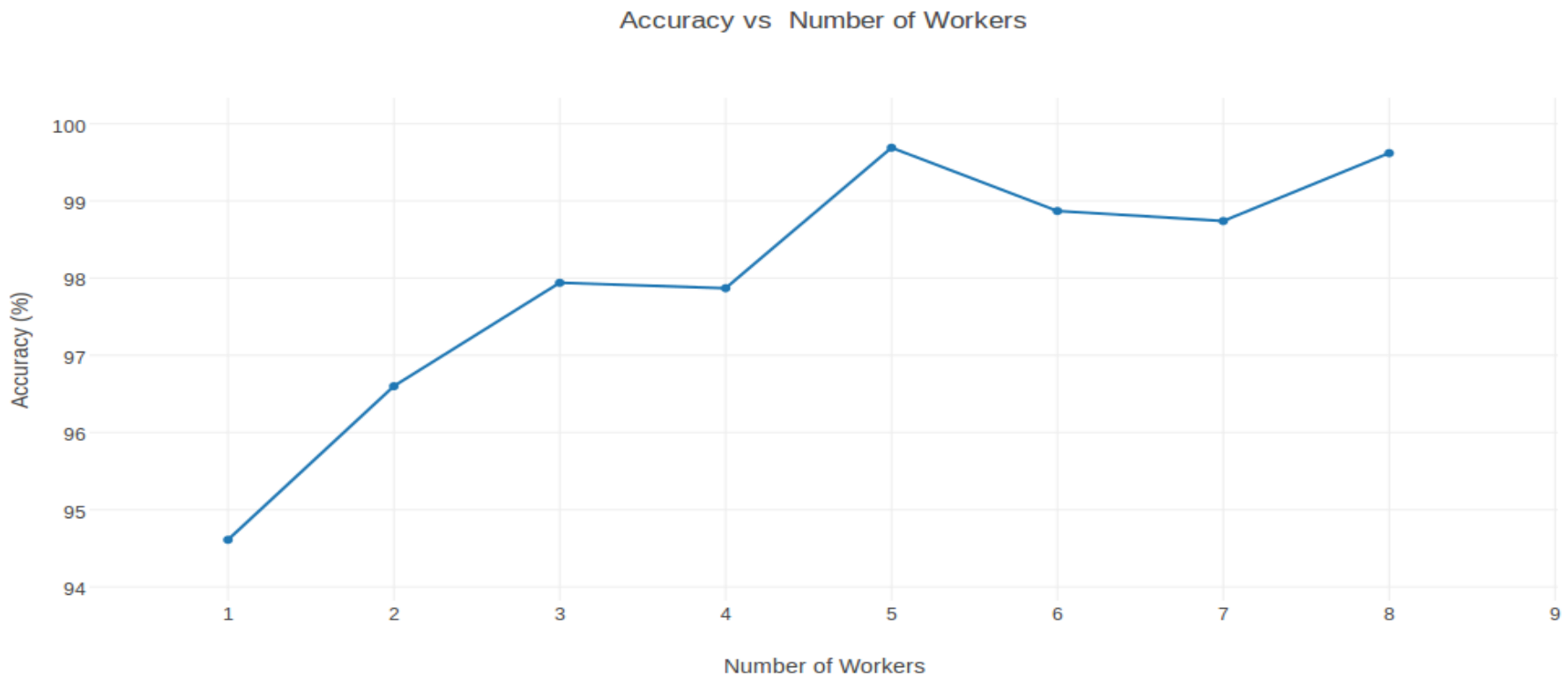
London

- socializing
- health/wellbeing
- language/ethnic identity
- food/drink
- outdoors/adventure
- hobbies/crafts
- sports/recreation
- tech
- career/business
- new age/spirituality
- fine arts/culture
- singles
- education/learning
- music
- dancing
- fitness
- photography
- support
- LGBT
- games
- movies/film
- community/environment
- religion/beliefs
- book clubs

Project Conclusions

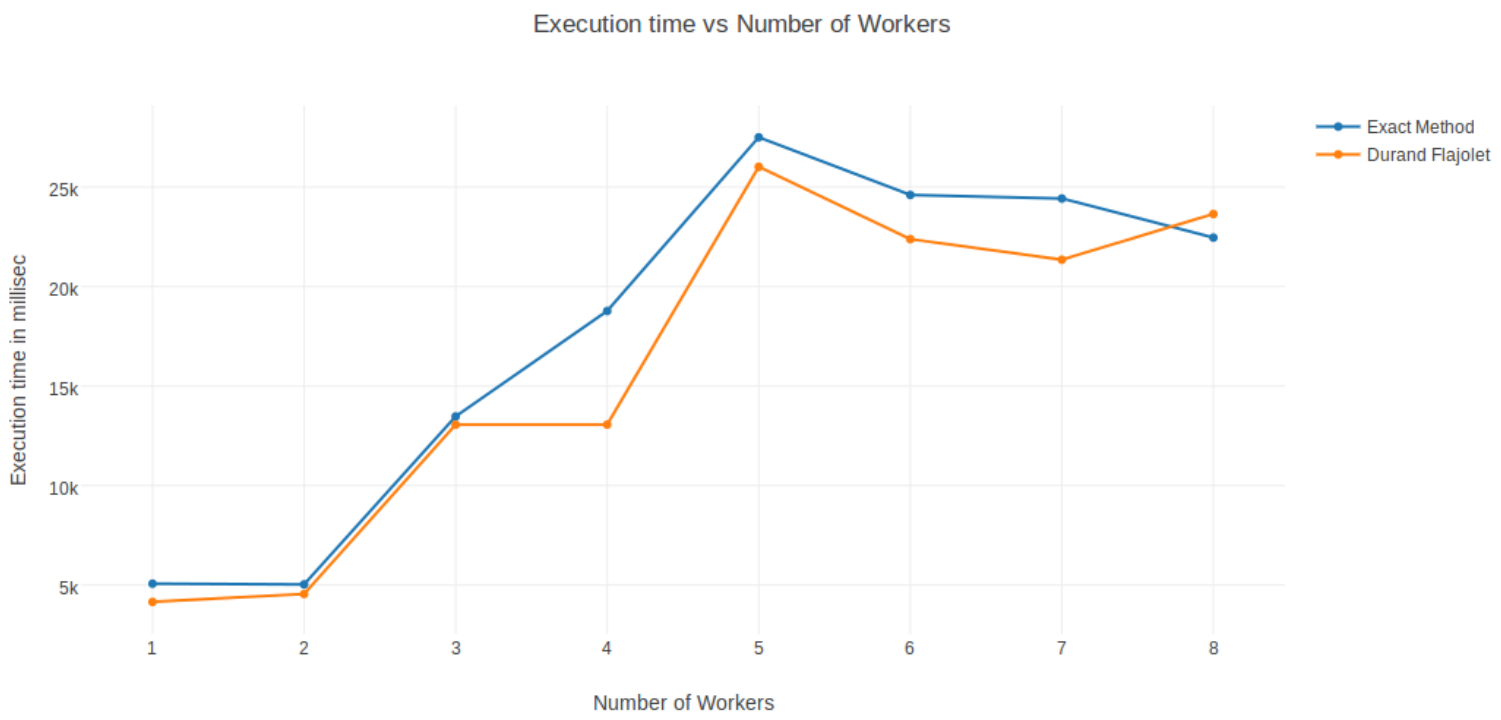
Accuracy of Durand Flajolet Approximation for unique Count

Accuracy of Durand Flajolet Approximation against Exact Count – with increase of streaming data, data rate and number of workers.



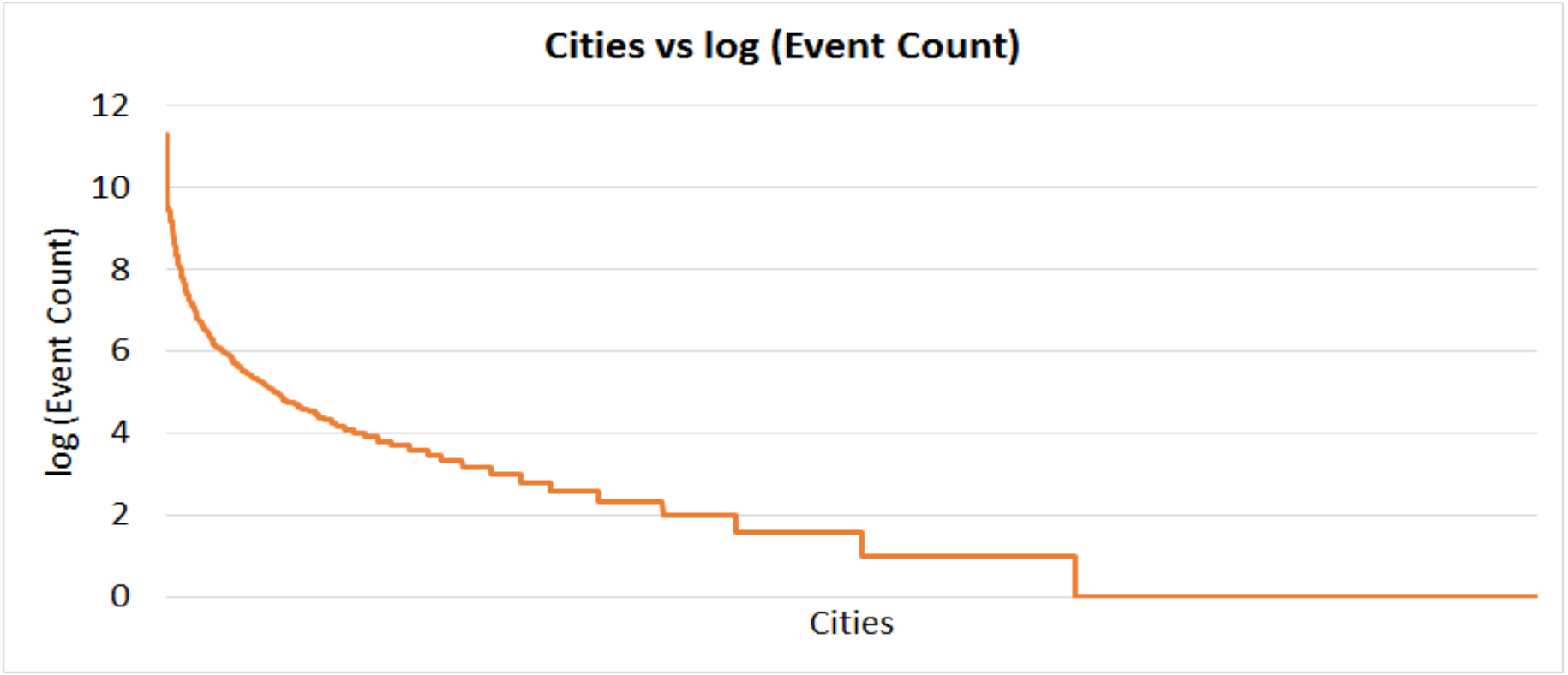
Total processing duration per tuple – Distinct Count Problem

Total processing time per tuple is plotted against the number of workers (i.e. Spouts and Bolts). It was observed that the execution time first increased up to the number of workers being 5, thereafter, it is staying to about 20-25 sec.



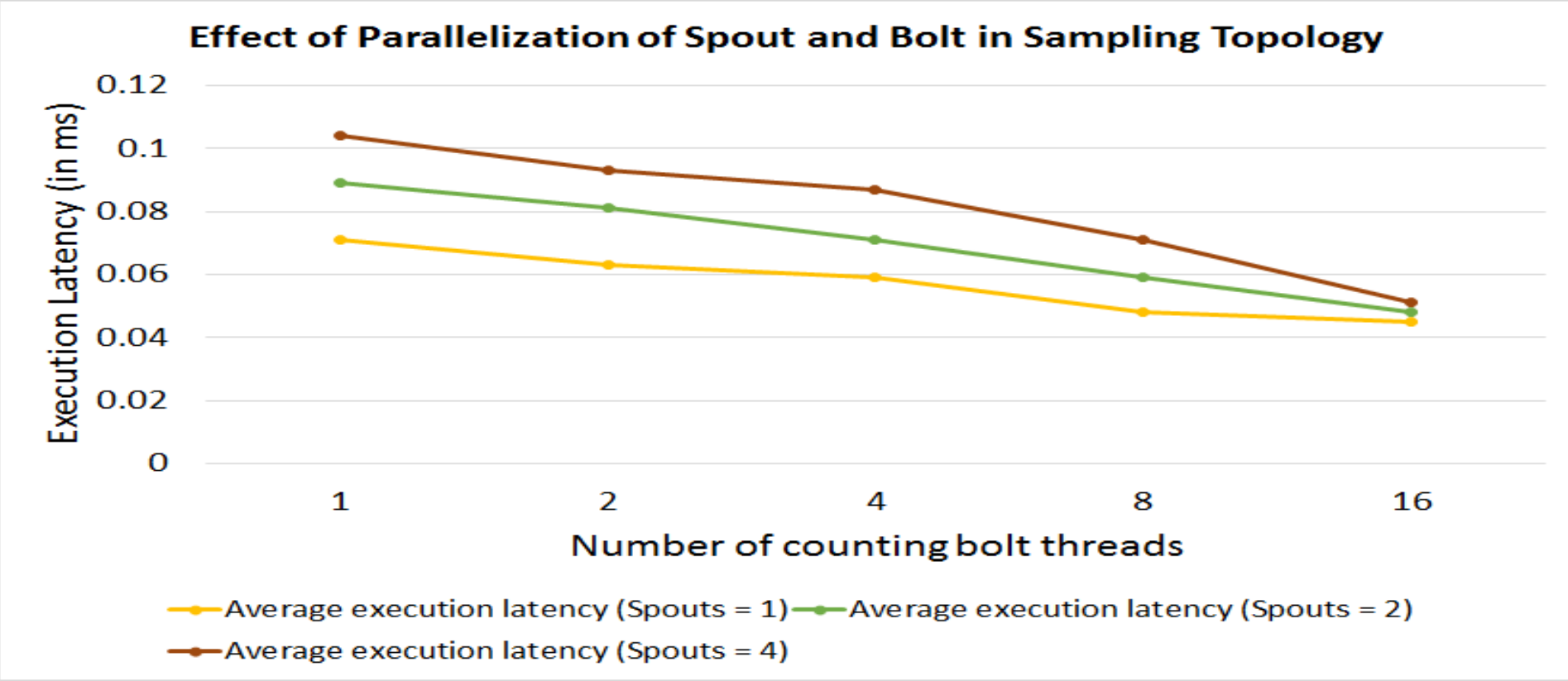
Power Law distribution for event count against cities

Cities follow a power law distribution in terms of numbers of events getting created. Few Cities contribute to major number of events. The result is similar in case of RSVPs too.



Thread level parallelism for Sampling topology

Effect of thread level parallelism in Apache Storm, as observed on the Sampling topology



Thank You