China · Beijing

The Road to a Complete Tweet Index

Yi Zhuang

Staff Software Engineer @ Twitter

ArchSummit

全球架构师峰会 2016

[北京站]

主办方 Geekbang› 极客邦科技 InfoQ

Outline

1. Current Scale of Twitter Search
2. The History of Twitter Search Infra
3. Complete Tweet Index
4. Search Engine Applications
5. Outlook

Outline

1. Current Scale of Twitter Search
2. The History of Twitter Search Infra
3. Complete Tweet Index
4. Search Engine Applications
5. Outlook

Current Scale of Twitter Search

More than 2 billion search queries per day.

Current Scale of Twitter Search

Hundreds of million Tweets are indexed per day.

Current Scale of Twitter Search

Hundreds of billions of Tweets have been sent since company founding in 2006.

Current Scale of Twitter Search

Our Complete Tweet Index is served by thousands of instances, each with 256GB RAM and 2TB SSD.

Current Scale of Twitter Search

But …
our search infrastructure is currently supported by only a small number of engineers and SREs.

We are hiring!

Outline

1. Current Scale of Twitter Search
2. The History of Twitter Search Infra
3. Complete Tweet Index
4. Search Engine Applications
5. Outlook

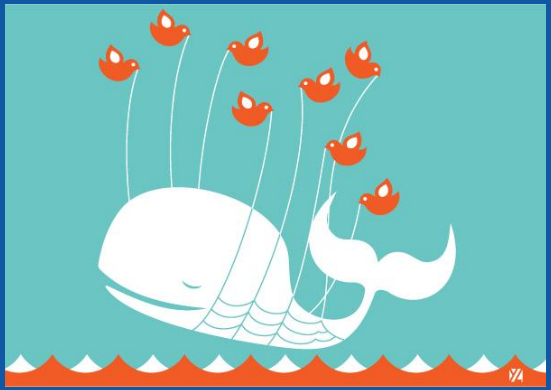
2010

Realtime Search Powered by replicated MySQL instances and MySQL text matching.

2010

Realtime Search Powered by MySQL.

- Hundreds of Tweets per second.
- A few thousand of queries per second.
- Basic text search: no fancy tokenization, no search assistance, slow geo search etc.
- Many incidents and down times.
- MySQL master/slave dying was particularly problematic.



2011

Launched Lucene-based search engine: Earlybird*.

- Lucene API, but custom data structures optimized for in-memory operations and Realtime search.
- Novel concurrent and lock free memory models: concurrently writing and searching an index segment.
- Contains about 7 days of Tweets.

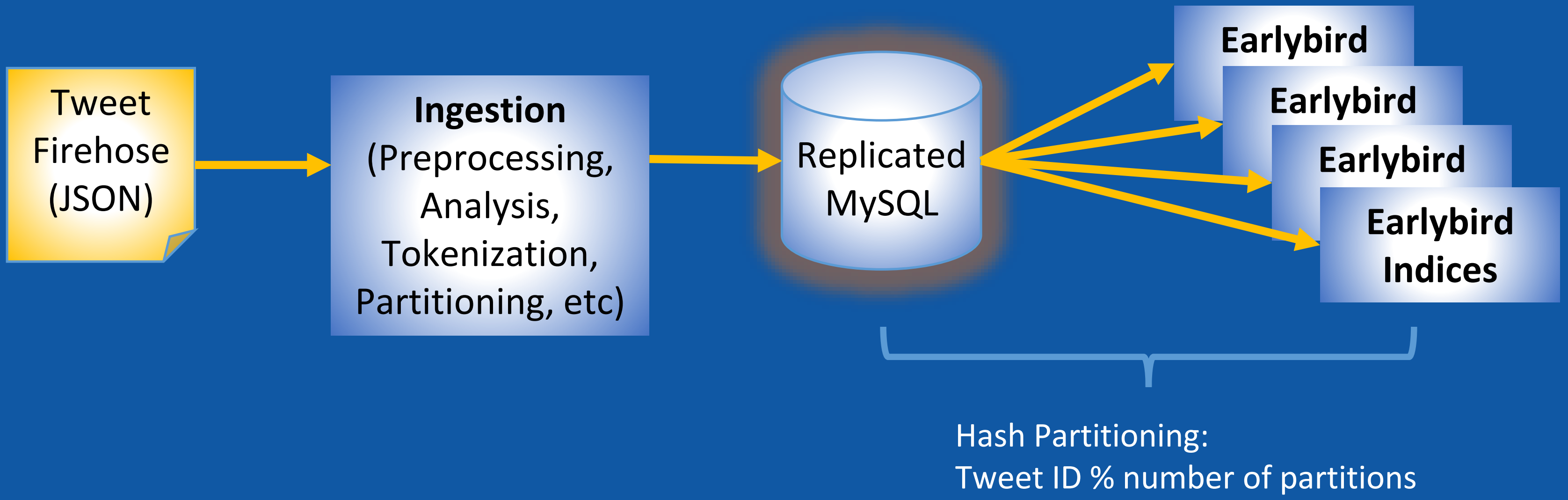
* http://www.umiacs.umd.edu/~jimmylin/publications/Busch_etal_ICDE2012.pdf

Earlybird vs Lucene / ElasticSearch

Earlybird	Lucene / ElasticSearch
Optimized for in-memory data structures	Optimized for Disks
Optimized for Realtime indexing and updates	Relatively slow Realtime indexing and updates
Optimized for Tweets	Index general documents
Facet & Term Statistics Support	N/A when we built Earlybird
Highly optimized for JVM Garbage Collection	Generates relatively more garbage
Thrift Query/Schema/Doc APIs	JSON Query/Schema/Doc APIs

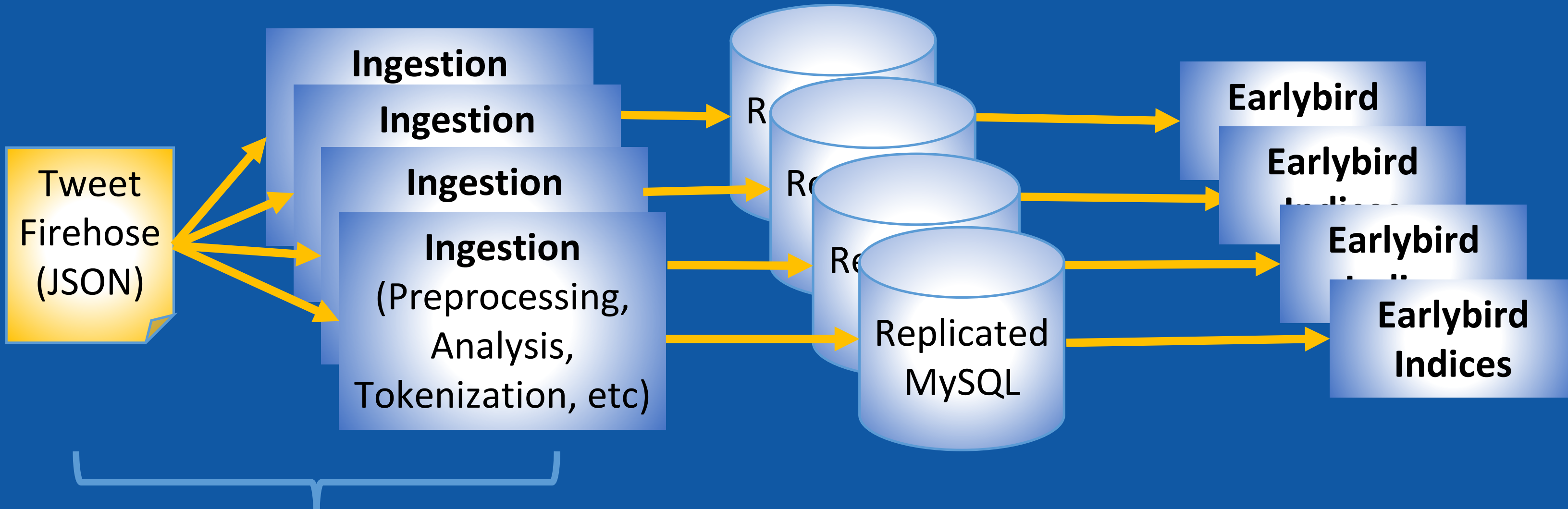
2011

Retired MySQL text matching, but still utilize MySQL to pipe data into Earlybird.



2012

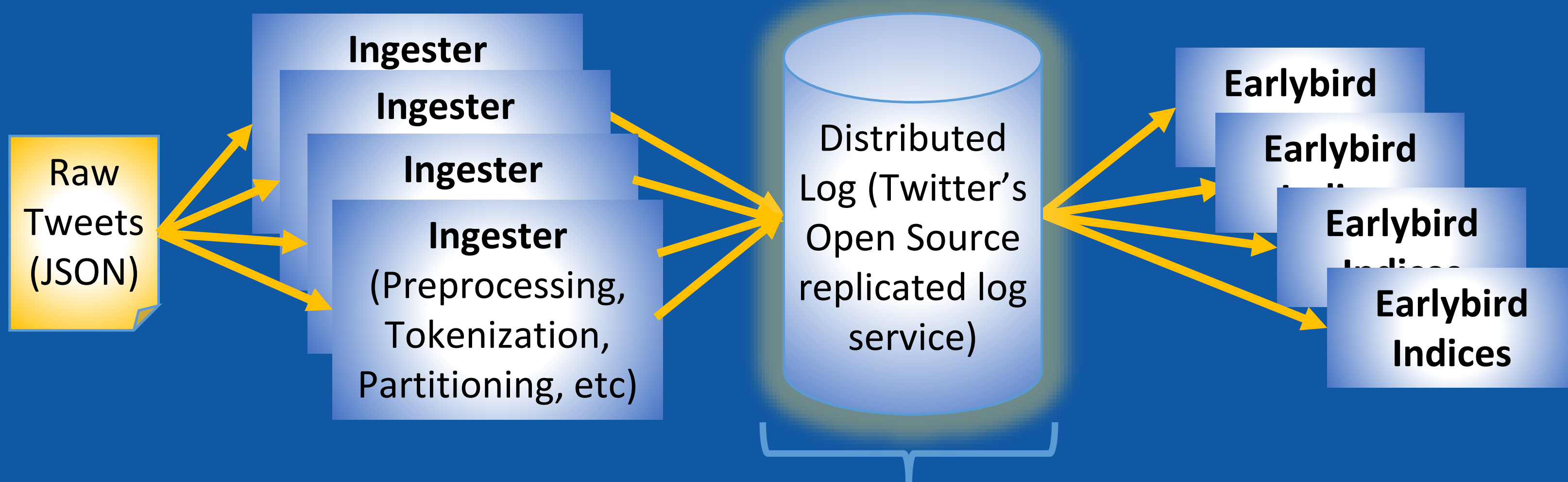
Eliminated Single Points of Failure via partitioning, decreasing the impact of MySQL master/slave failures.



Hash Partitioning:
Tweet ID % number of partitions

2013-2015

Eliminating the use of MySQL as our data bus.



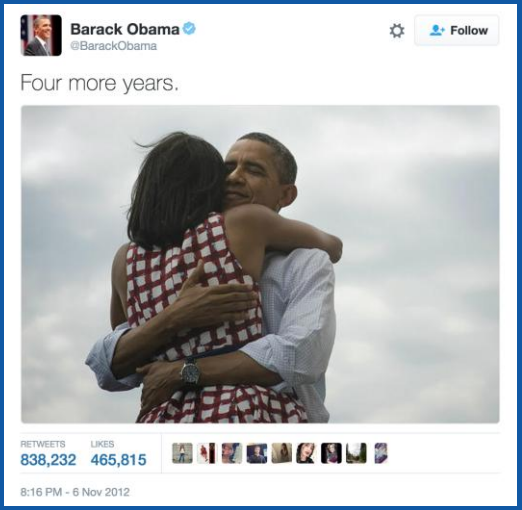
Twitter's Partitioned, Replicated,
High-performance Messaging System.
http://distributedlog.incubator.apache.org/

Outline

1. Current Scale of Twitter Search
2. The History of Twitter Search Infra
3. Complete Tweet Index
4. Search Engine Applications
5. Outlook

Complete Tweet Index Motivation

Be able to search for any Tweet ever published, not just Tweet from the latest 7 days. (approx. 300x scaling)



Existing Architecture Challenges

- Small team: limited number of engineers and SREs.
- Realtime search in-memory architecture cannot hold hundreds of billions of Tweets in RAM, we just do not have enough RAM, and even if we do, it is not cost effective.
- Scaling is non-trivial: Realtime search architecture has roughly fix size (7 days of Tweets), but the Complete Tweet Index needs to grow bigger each day.
- Ingestion parallelism is low and fixed --- parallelism is achieved via partitioning: 20 partitions means 20 parallel ingestion pipelines.

Complete Tweet Index Design Goals

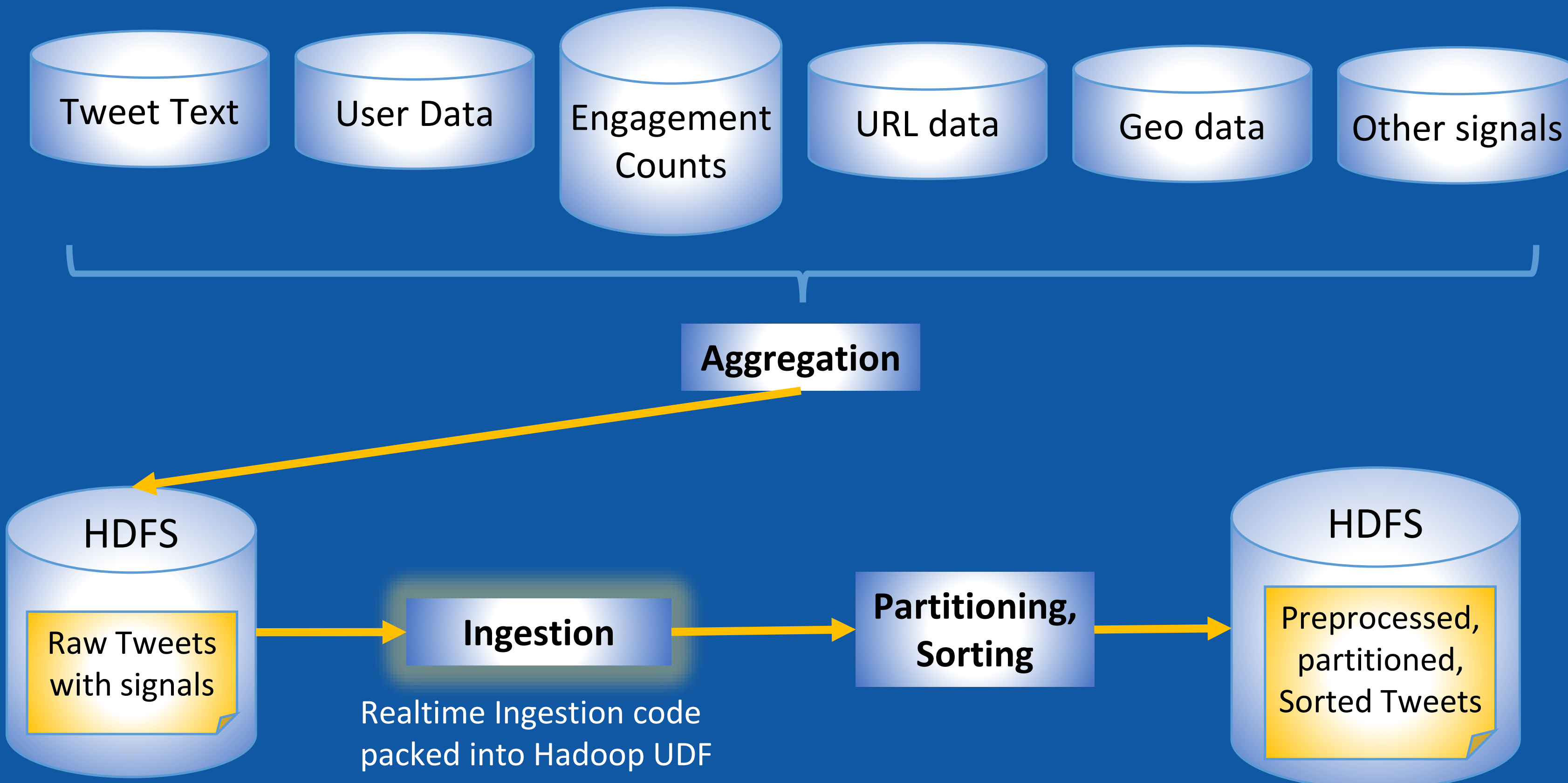
- Index every Tweet ever published.
- Modularity: Shared source code and tests between the Realtime and Complete Tweet Index where possible, which created a cleaner system in less time.
- Scalability: expands in place gracefully as more Tweets are added.
- Cost effectiveness: Using the same RAM technology for the complete index would have been prohibitively expensive.
- Highly parallel ingestion: ability to fully rebuild the index in reasonable amount of time.
- Simple interface: wanted a simple interface that hides the underlying partitions so that internal clients can treat the cluster as a single endpoint.

Complete Tweet Index Design Overview

Complete Tweet Index Major Components

- Batch ingestion: batch Tweet preprocessing pipeline on Hadoop
- Inverted index builder: stateless inverted index builder running on Mesos
- Index shards: 2-D sharded search index servers
- Roots: hierarchical scatter gather service running on Mesos

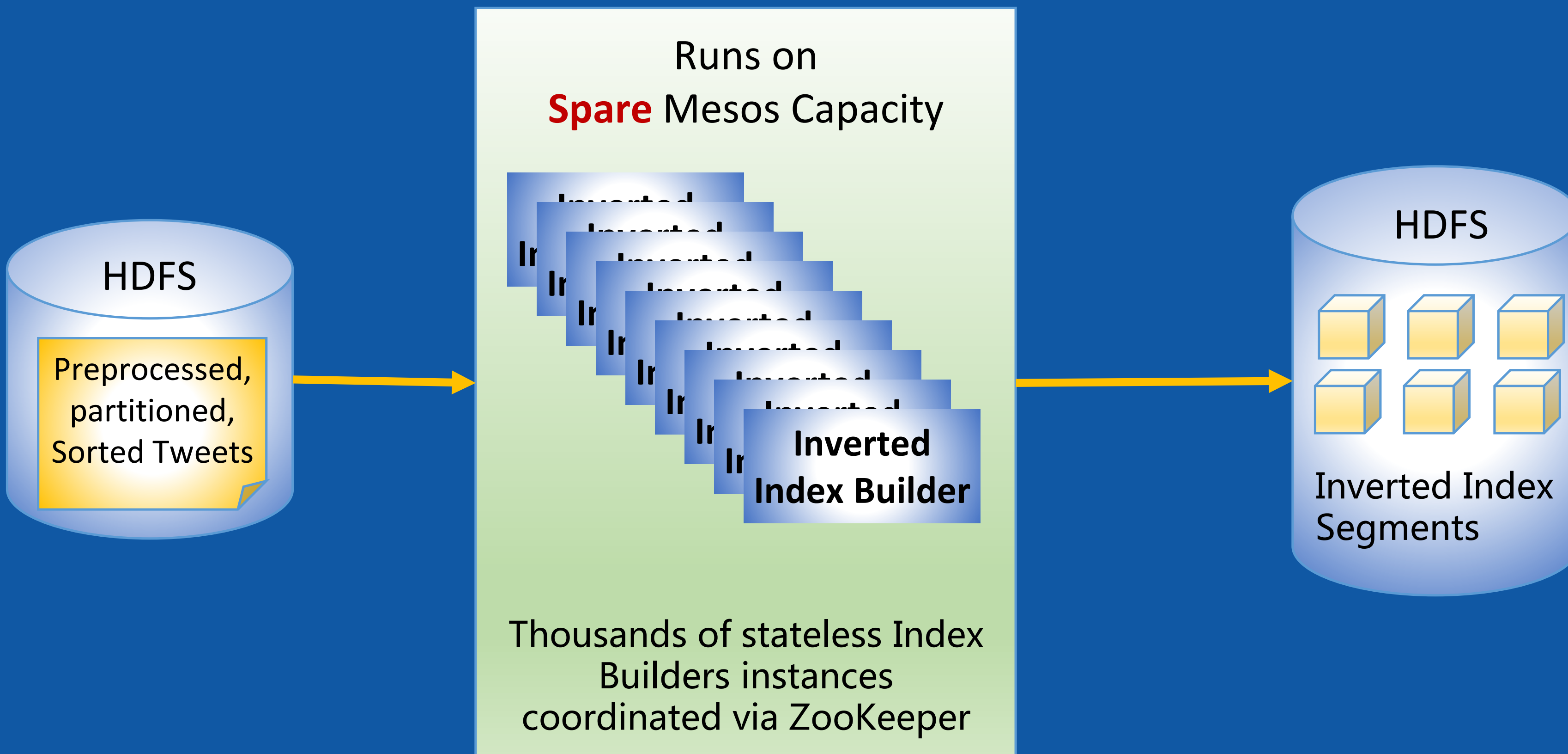
Batch Ingestion Pipeline



Batch Ingestion Pipeline

- Heavy code reuse from Realtime streaming pipeline.
  - Reusing unit tests was a big time saver.
- Heavy code sharing between daily processing and full index rebuilds.
- Parallel Tweet processing on Hadoop
  - Bad idea to hit downstream services via RPC: easy to DDoS them from thousands of nodes from Hadoop.
  - Aggregate logs on HDFS and join them on tweet ID.

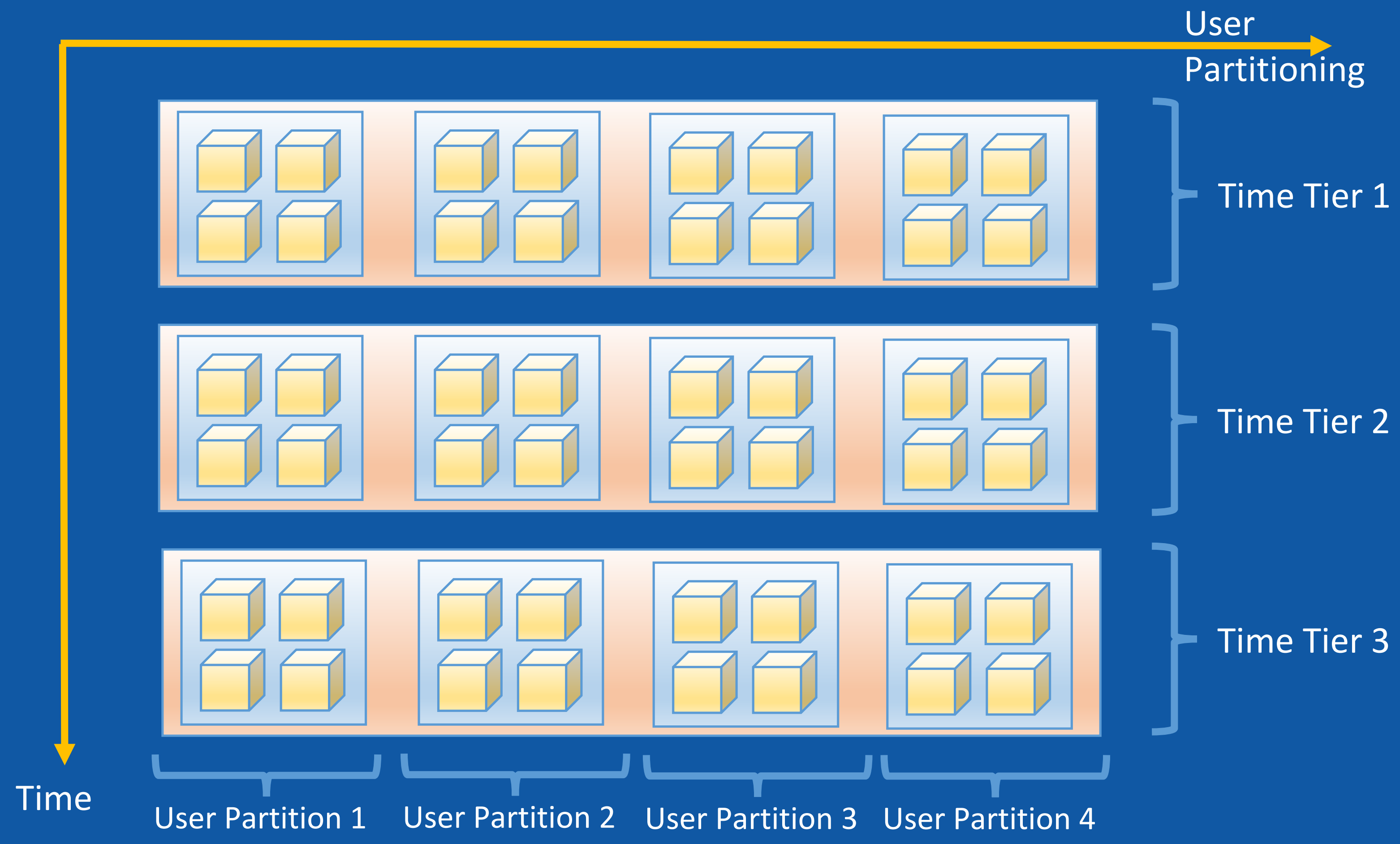
Inverted Index Builders



Inverted Index Builders

- Highly parallel
- Single threaded and stateless
- Elastic scaling: instance count can be changed with a single config update
- Runs on spare Meso capacity.
- Coordinated via ZooKeeper:
  - Avoid doing duplicate work
  - Avoid overloading HDFS namenodes and data links
  - Work progress tracking

2D Partitioned Index Shards



2D Partitioned Index Shards

- Temporal sharding: The Tweet corpus was first divided into multiple time tiers.
- Hash partitioning: Within each time tier, data was divided into partitions based on a hash function.
- Earlybird: Within each hash partition, data was further divided into chunks called Segments. Segments were grouped together based on how many could fit on each Earlybird machine.
- Replicas: Each Earlybird machine is replicated to increase serving capacity and resilience.

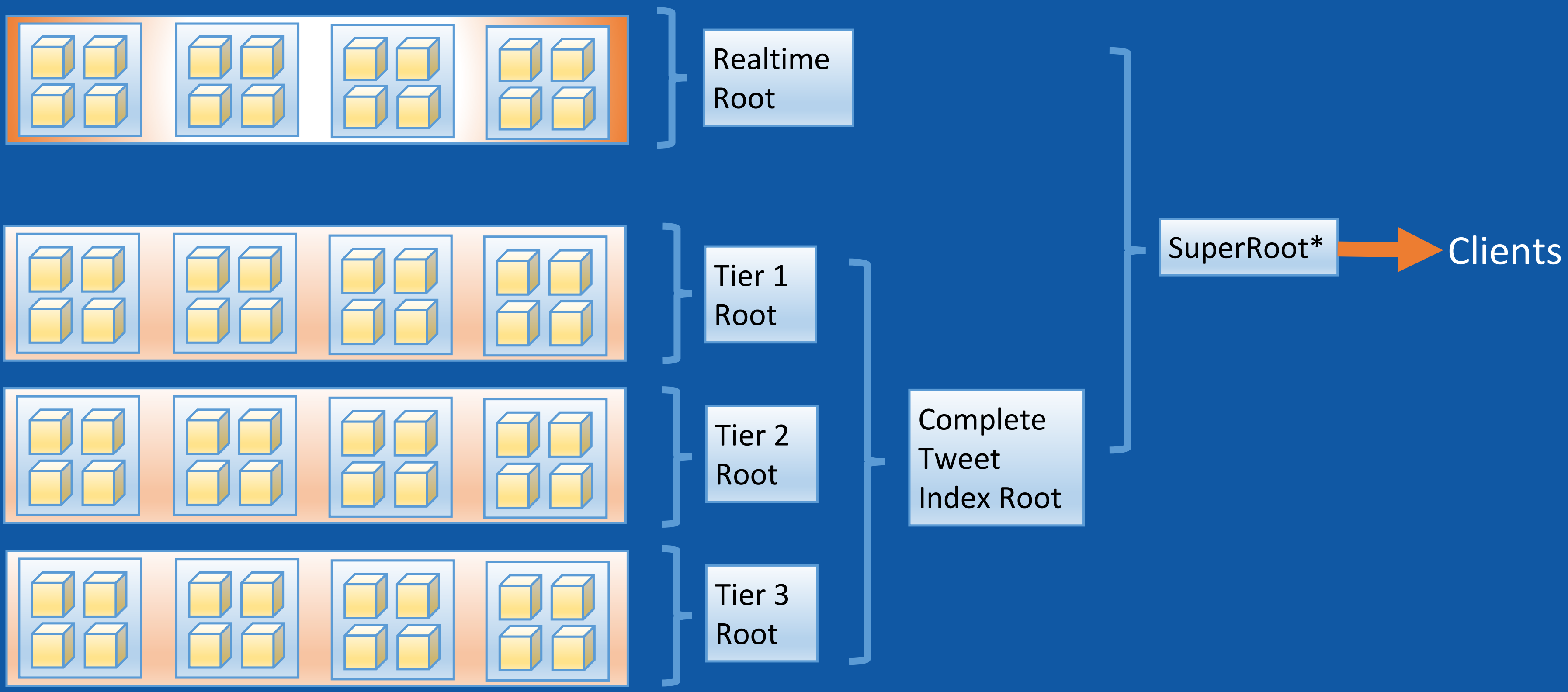
2D Partitioned Index Shards

- Easy Scaling:

  - To grow data capacity over time, we will add time tiers. Existing time tiers will remain unchanged. This allows us to expand the cluster in place.

  - To grow serving capacity (QPS) over time, we can add more replicas.

Roots: Hierarchical Scatter / Gather



* SuperRoot: https://blog.twitter.com/2016/superroot-launching-a-high-sla-production-service-at-twitter

Roots: Hierarchical Scatter / Gather

- Each tier root collects from all shards in its tier.
- Hierarchical scatter-gather.
- Simple Client API: SuperRoot collects from Complete Tweet Index root and Realtime root
  - Stitching results together
  - Pagination support
- Reduces open network connections on each root.

Complete Tweet Index
Challenges and Difficulties

The Challenges are in the Details

- Launched Complete Tweet Index in November 2014.
  - Few unexpected problems while designing and building the system.
- Afterwards, spent over a year fixing details.
  - Ran into many seemingly common and ordinary distributed system problems.
  - Resolving them was non-trivial and time consuming.
  - Tendency to underestimate the amount of effort needed to work out these "small" problems.

Query Cost and Tail Latency is High

- 7-day Realtime search: each query hits about 20 shards.
- Complete Tweet Index: each query hits about 150 shards.
  - Each query is more expensive.
  - Tail latency (p99, p999) is high.

Query Cost and Tail Latency is High

- 7-day Realtime search: each query hits about 20 shards.
- Complete Tweet Index: each query hits about 150 shards.
  - Each query is more expensive.
  - Tail latency (p99, p999) is high.
- Client side optimizations:
  - Smart request routing: time based; user hash-partition based.
  - Backup request: if a shard does not respond within a configured time limit, send request to another replica with the same data.
  - Send failure tolerance. (E.g. client can tolerate 10% shard failures/timeouts.)
  - Send timeout requirements. (E.g. client will wait up to 800ms.)

Query Cost and Tail Latency is High

- Server side optimizations:
  - Optimize JVM Garbage Collections:
    - Only one ConcurrentMarkSweep pause each day.
    - Proactive full GCs when not serving traffic.
  - Deadline estimation and early termination:
    - Measure network latency and request queueing time.
    - Estimate outstanding work time by monitoring progress.
    - Early terminate before deadline specified by client.
    - Respond with partial results and a cursor.
    - Clients can resume the search by sending back the cursor.

Large Traffic Spikes

- During events like Oscar, World Cup, Earthquakes etc, QPS (queries per second) can be an order of magnitude (more than 10x) higher.
  - Over 200% capacity provisioning in multiple datacenters.
    - Hyper-Threading: CPU cores are not always real: 25% CPU utilization does not mean the system can serve 4x more traffic.
    - Regular automated redline tests to estimate real capacity.
  - Caching
  - Per-client request rate throttling
  - Graceful success rate degradation.
  - Graceful feature degradation. E.g. turn off personalization.
    - Cheaper requests
    - Higher cache rate.

Working with Thousands of Instances and Petabytes of Data

- Accidental DDoS on downstream services:
  - Brought down HDFS namenode multiple times.
  - Overwhelmed ZooKeeper: writes into ZK are expensive.
  - Saturated HDFS download link bandwidth multiple times.
    - Long startup times: slow network infrastructure.
- Frequent hardware failures in a large cluster.
  - Provision 1 extra live replica; multiple spare machines on standby.
  - Automated hardware failure detection and alerts.
  - Automated hardware state transition and hot swaps:
    - States: managed, maintenance, allocated, blacklisted.

SSD Performance in Reality

- Actual SSD performance is lower than advertised performance.

  Things we have tried:

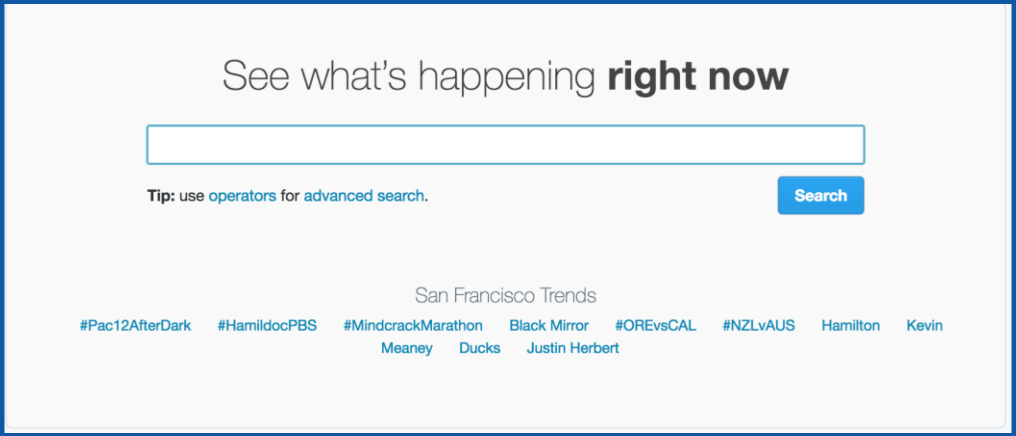
  - Heap size vs Filesystem cache balance.
  - Load frequently used random-accessed signals into RAM.
  - Avoid serving queries while indexing (while writing to the SSDs).
  - Avoid fully utilizing space on SSDs. (< 80 % full)
  - Manually tuning parameters, such as read ahead settings and I/O schedulers parameters.

Outline

1. Current Scale of Twitter Search
2. The History of Twitter Search Infra
3. Complete Tweet Index
4. Search Engine Applications
5. Outlook

Search Engine is not Just For Searching

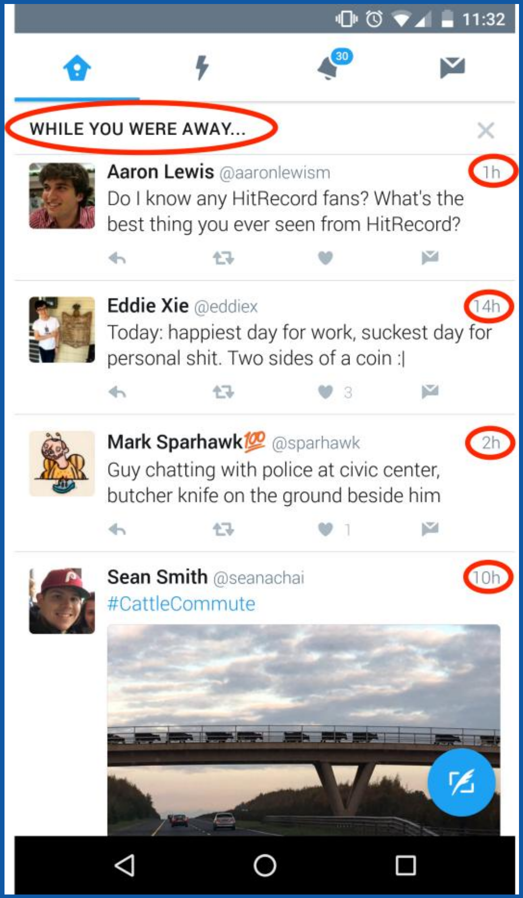
- Our team (Search Infra) builds search engines that can power many applications. Twitter Search is just one of them!

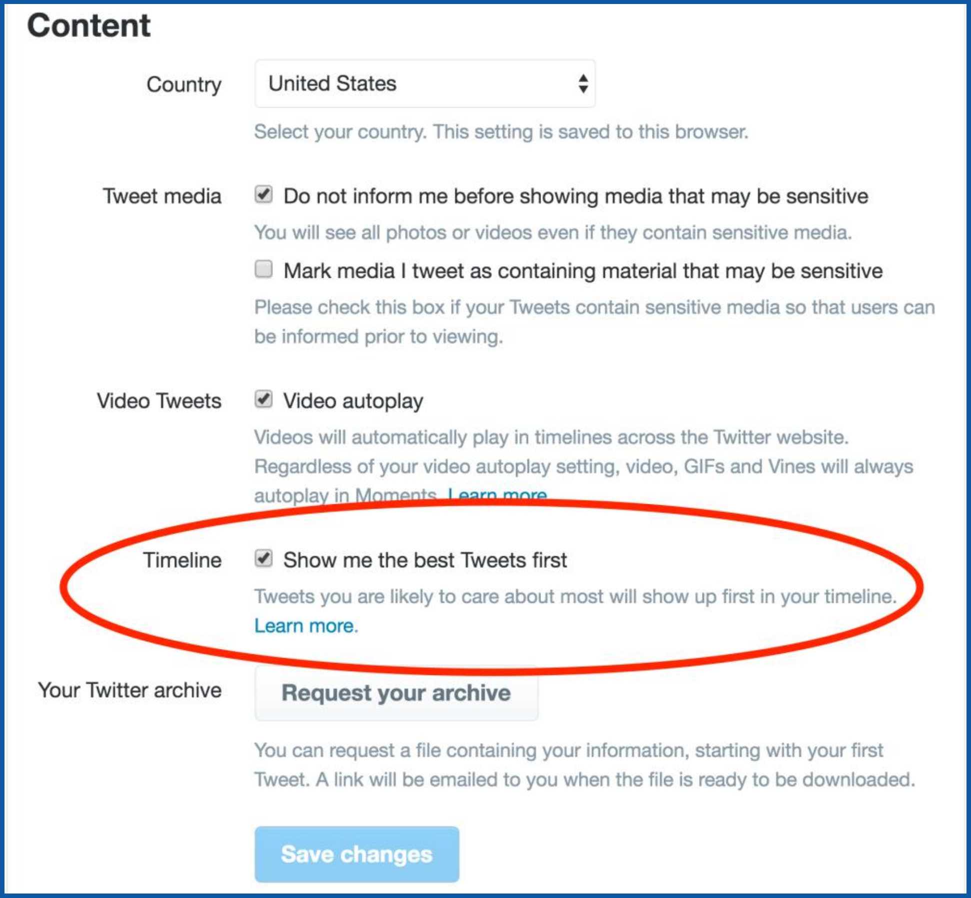


Search Engine Example Applications

- Ranked Timelines
- Gnip: Enterprise realtime and historical data API
- Ads and sales: historical term histograms. E.g. during the past 30 days, how many times was the word Twitter mentioned each hour?
- Antispam
- Many others

Ranked Timelines





Ranked Timelines Motivations

- Most users read a small fraction of Tweets in their home timeline.
  - From a random slice depending on when they open Twitter.
  - Likely not the best or most engaging Tweets for the user.
- For users who follow a lot of Twitter accounts:
  - Home timeline quality can be low.
  - Home timeline velocity is high---easy to miss high quality Tweets or Tweets from close friends.

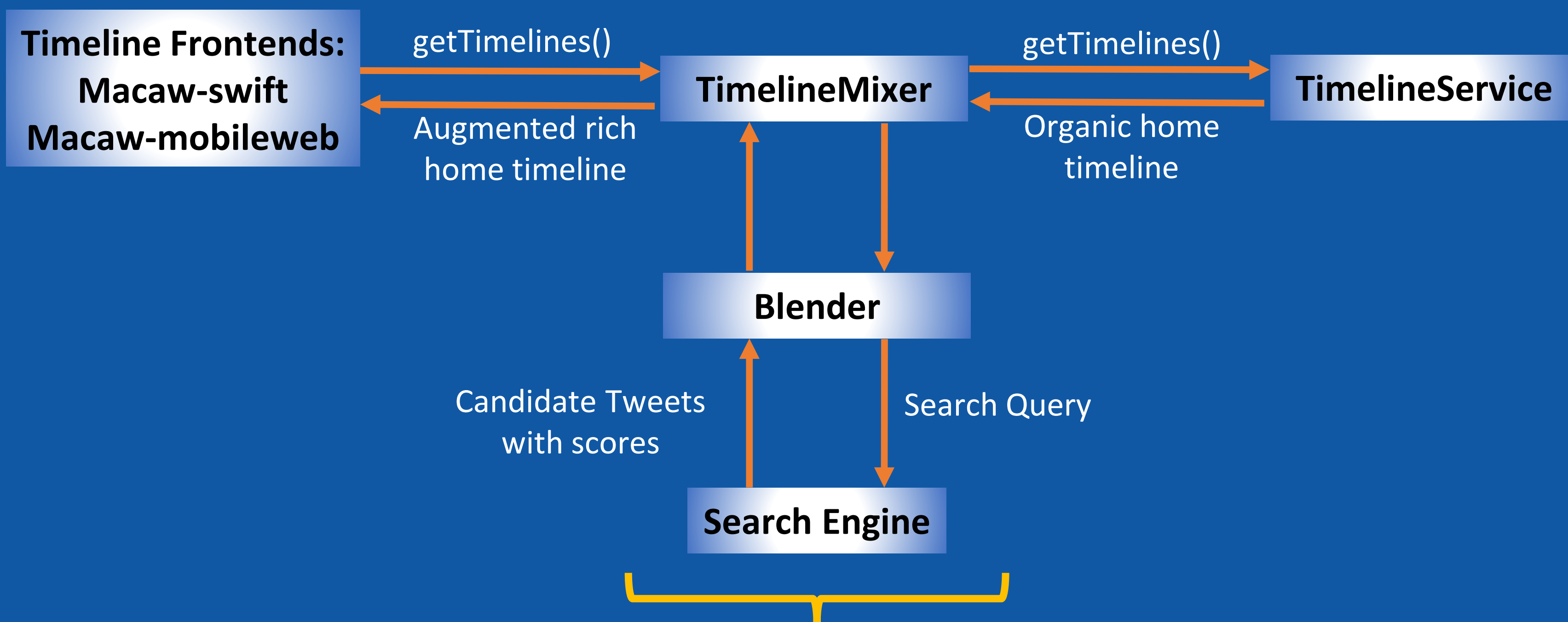
Ranked Timelines

- New timeline setting to "show the best Tweets first", as opposed to just the traditional reverse-chronological timeline.
  - Home timeline no longer just sorted by time in descending order.
  - In general, more engaging Tweets are shown at the top.
  - The rest of their Tweets will be displayed right underneath, in reverse chronological order.
  - Whenever the user pulls down to refresh, it is back to reverse chronological timeline.
- Recap module: while you were away
  - A module injected into the home timeline.
  - Shows the top Tweets the user missed since the last log in.

Ranked Timelines Using Search

- Our search engine can compute relevance scores on large amounts of Tweets efficiently.
- Home timeline construction using search queries:
  - If user X follows U1, U2, U3, ..., can construct timeline using Search.
  - Search Query: "from:U1 OR from:U2 OR from:U3 ...".
  - Search engine returns Tweets with a relevance score.
- First pass filtering based on search engine relevance scores.
- Second pass filtering / ranking based on machine learned models which predict likelihood of engagement.

Ranked Timeline / Recap High Level Architecture



When Search is not available, fall back to the organic reverse-chronological home timeline.

Outline

1. Current Scale of Twitter Search
2. The History of Twitter Search Infra
3. Complete Tweet Index
4. Search Engine Applications
5. Outlook

Outlook

- Better support for scoring and ranking inside the search engine.
  - More signals and more efficient signal access.
  - Efficiently support commonly used machine learning based scorers.
  - Allow clients to add signals and scoring functions into our search engine without friction.
- Improved SLA and resilience: from powering the search box to powering the home timelines at Twitter.
- OmniSearch*:
  - Support other use cases besides Tweet and User search.
  - Generic information retrieval and ranking system as a service.
  - Power many more products at Twitter.

* https://blog.twitter.com/2016/introducing-omnisearch

Acknowledgements

- The Complete Tweet Index is work of the Twitter Search Infrastructure team.
- Besides the Search Infrastructure team, many people contributed to this project. Please see our blog post for a full list of names of contributors: **https://blog.twitter.com/2014/building-a-complete-tweet-index**

THANKS

ArchSummit
全 球 架 构 师 峰 会 2016

[北京站]

主办方 Geekbang>. 极客邦科技 InfoQ