Kafka & Redis for Big Data Solutions

- Christopher Curtin
- Head of Technical Research
- @ChrisCurtin

About Me

- 25+ years in technology
- Head of Technical Research at Silverpop, an IBM Company (14 + years at Silverpop)
- Built a SaaS platform before the term 'SaaS' was being used
- Prior to Silverpop: real-time control systems, factory automation and warehouse management
- Always looking for technologies and algorithms to help with our challenges

SILVERPOP
An IBM® Company

About Silverpop

- Provider of Internet Marketing Solutions
- 15+ years old, Acquired by IBM in May 2014
- Multi-tenant Software as a Service
- Thousands of clients with 10's of billions of events per month

Agenda

- Kafka
- Redis
- Bloom Filters

Batch is Bad

- Data Egress and Ingress is very expensive
- 1000's of customers mean thousands of hourly or daily export and import requests
- Often same exports are being requested to feed different business systems
- Prefer Event Pushes to subscribed end points

Apache Kafka

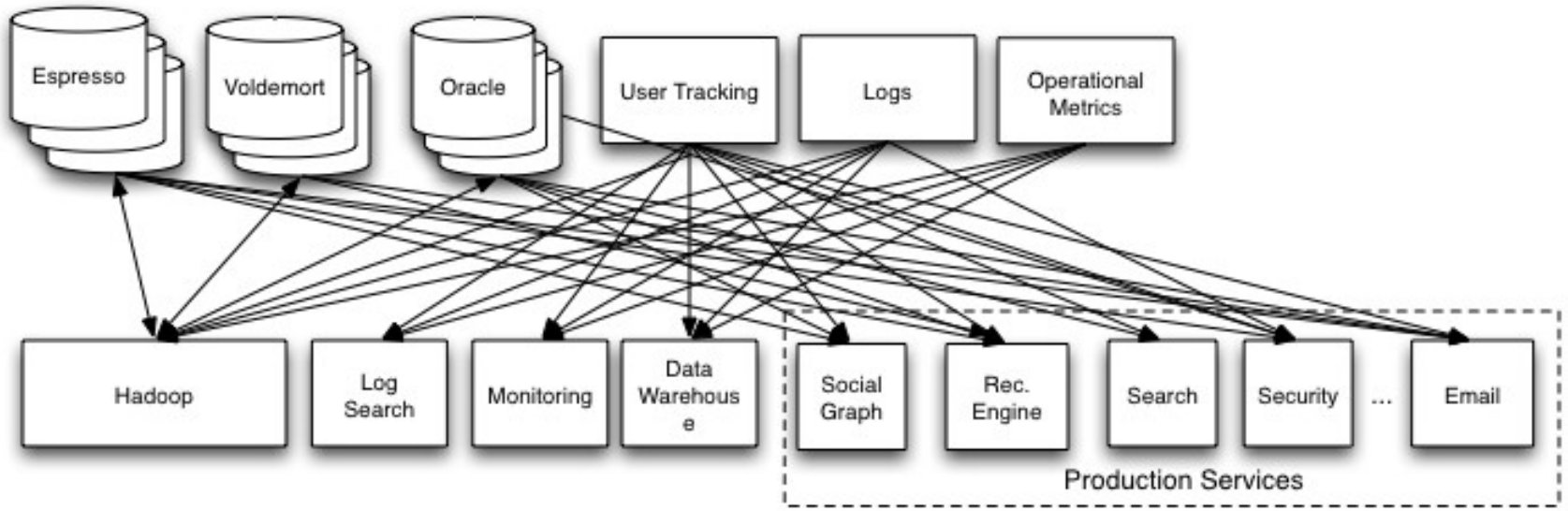
Apache Kafka is a distributed publish-subscribe messaging system. It is designed to support the following

- Persistent messaging with O(1) disk structures that provide constant time performance even with many TB of stored messages.
- High-throughput: even with very modest hardware Kafka can support hundreds of thousands of messages per second.
- Explicit support for partitioning messages over Kafka servers and distributing consumption over a cluster of consumer machines while maintaining per-partition ordering semantics.
- Support for parallel data load into Hadoop.

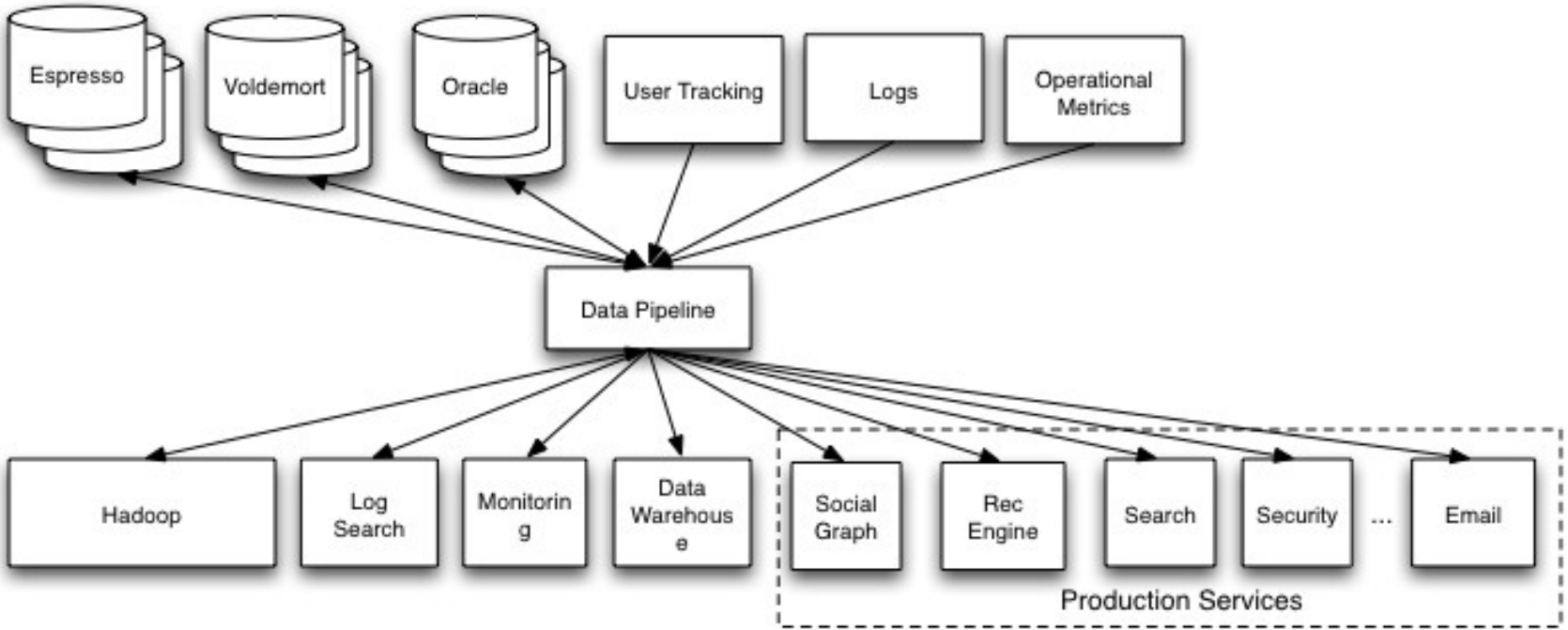
Apache Kafka – Why?

Data Integration

Point to Point Integration (thanks to LinkedIn for slide)



What we'd really like (thanks to LinkedIn for slide)



Apache Kafka

Kafka changes the messaging paradigm

Kafka doesn't keep track of who consumed which message

Kafka keeps all messages until you tell it not to

A consumer can ask for the same message over and over again

Consumption Management

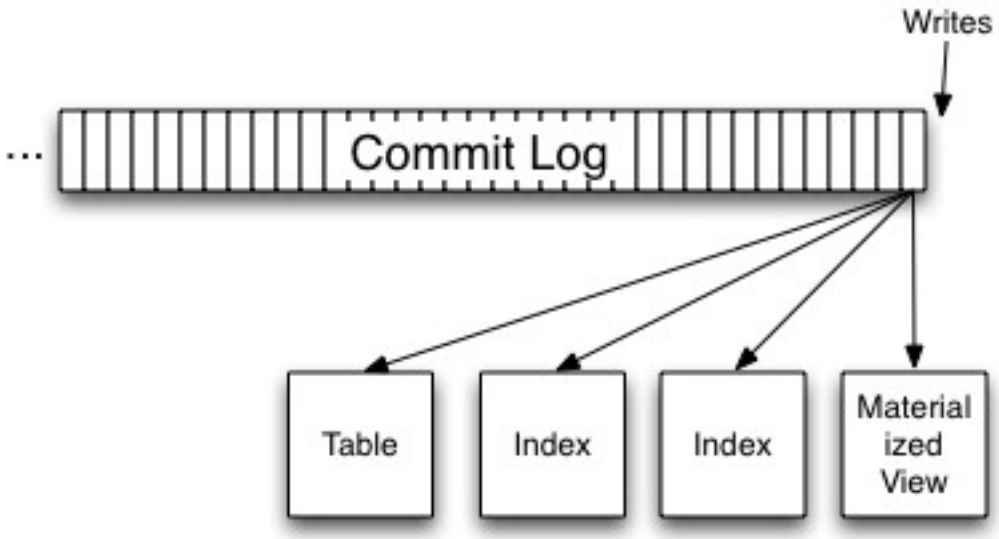
Kafka leaves the management of what was consumed to the consumer

Each message has a unique identifier (within a topic and partition)

Consumers ask for the specific message, or the next one

All messages are written to disk, so asking for an old message means finding it on disk and starting to stream from there

What is a commit log? (thanks to LinkedIn for slide)



Big Data Use Cases

Data Pipeline

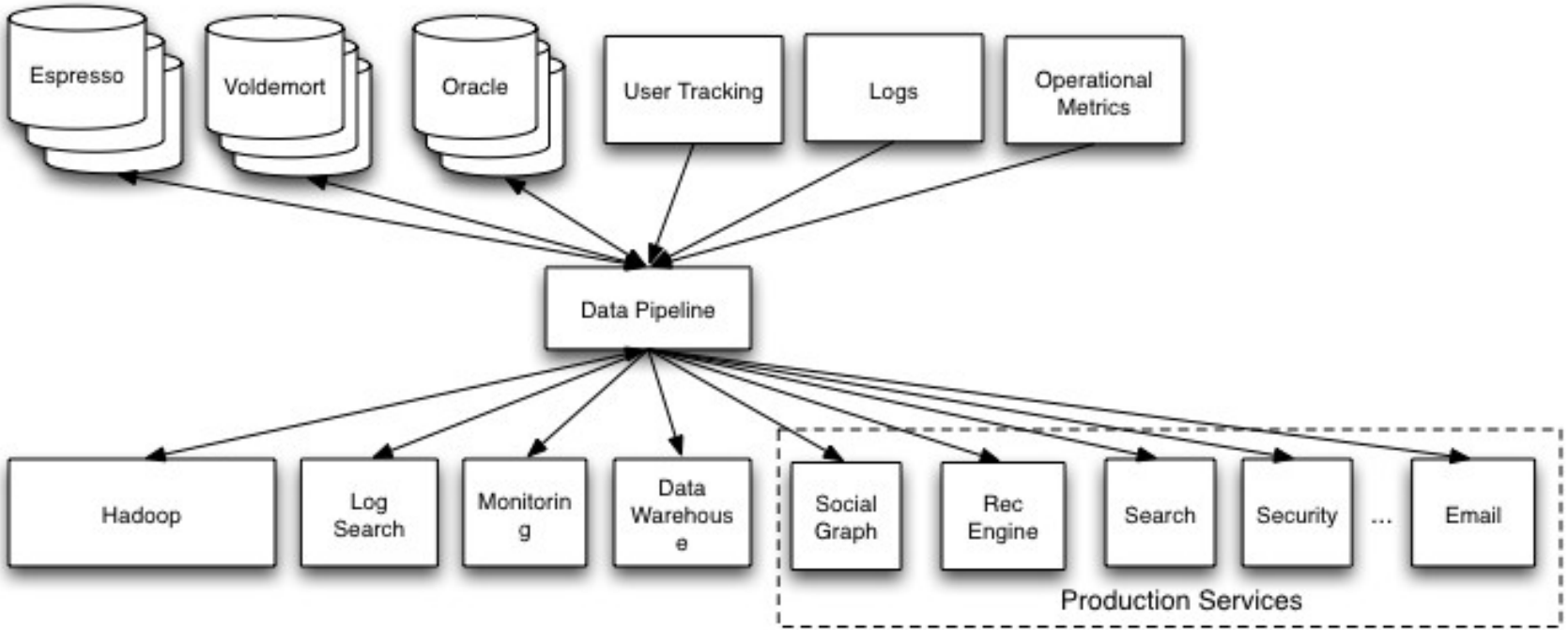
Buffer between event driven systems and batch

Batch Load

Parallel Streaming

Event Replay

Data Pipeline (thanks to LinkedIn for slide)



Buffer

Event generating systems MAY give you a small downtime buffer before they start losing events

Most will not.

How do you not lose these events when you need downtime?

Batch Load

Sometimes you only need to load data once an hour or once a day

Have a task wake up and stream the events since the last time to a file, mark where it last picked up and exit

Parallel Streaming

A lot of use cases require different systems to process the same raw data

Spark, Storm, Infosphere Streams, Spring XD …

Write to a different Kafka Topic and Partition

Batch load into HDFS hourly

Batch load into a traditional data warehouse or database daily

Push to external systems daily

Example: abandoned shopping carts

Event Replay

What if you could take 'production' data and run it through a debugger?

Or through a non-optimized, 'SPEW' mode logging to see what it does?

Or through a new version of your software that has /dev/null sinks

Note: Developers HAVE NOT made a local copy of the data

Though with a sanitized copy for QA 'load testing' is simplified.

What about Flume etc?

Very similar use cases, very different implementation

Kafka keeps the events around

Flume can do much more (almost a CEP system)

Kafka feeding Flume is a reasonable combination

Big Data Egress

Big data applications often produce lots of data

Nightly batch jobs

Spark Streaming alerts

Write to Kafka

- Same benefits when feeding big data can be applied to consumers of our output

Or …

What about Hbase or other 'Hadoop' answers For Egress?

Perfect for many cases

Real-world concerns though

- Is the YARN cluster 5-9s?
- Is it cost effective to build a fast response Hbase, HDFS etc. for the needs?
- Will the security teams allow real-time access to YARN clusters from web-tier business application?

Redis – What is it?

From redis.io:

"Redis is an open source, BSD licensed, advanced key-value cache and store. It is often referred to as a data structure server since keys can contain strings, hashes, lists, sets, sorted sets, bitmaps and hyperloglogs."

Features

- Unlike typical key-value stores, you can send commands to edit the value on the server vs. reading back to the client, updating and pushing to the server
- pub/sub
- TTL on keys
- Clustering and automatic fail-over
- Lua scripting
- client libraries for just about any language you can think of

So Why did we start looking at NoSQL?

"For the cost of an Oracle Enterprise license I can give you 64 cores and 3 TB of memory"


CHALLENGE
ACCEPTED!

Redis Basics

- In Memory-only key-value store
- Single Threaded. Yes, Single Threaded
- No Paging, no reading from disk
- CS 101 data structures and operations
- 10's of millions of keys isn't a big deal
- How much RAM defines how big the store can get

Hashes

Hashes

- collection of key-value pairs with a single name
- useful for storing data under a common name
- values can only be strings or numeric. No hash of lists

http://redis.io/commands/hget

Web-time access

Write the output of a Spark job to Redis so it can be accessed in web-time

Publish the results of recommendation engines for a specific user to Redis in a Hash by User or current page

Using a non-persistent storage model, very low cost to load millions of matches into memory

- Something goes wrong? Load it again from the big data outputs
- New information for 1 user or page, replace the value for that one only

SILVERPOP
An IBM® Company

Spark RDDs

Over time Spark can generate a lot of RDDs.

- Where are they?
- What is in them?

Some people use a naming convention, but that gets problematic quickly

Instead, create a Redis Set by 'key' you are interested in finding the RDDs later. Date, source system, step in algorithm on date etc.

Since Redis Set is only a reference to the RDD, can easily create lots of Sets pointing to same data (almost like an index)

Bloom Filters

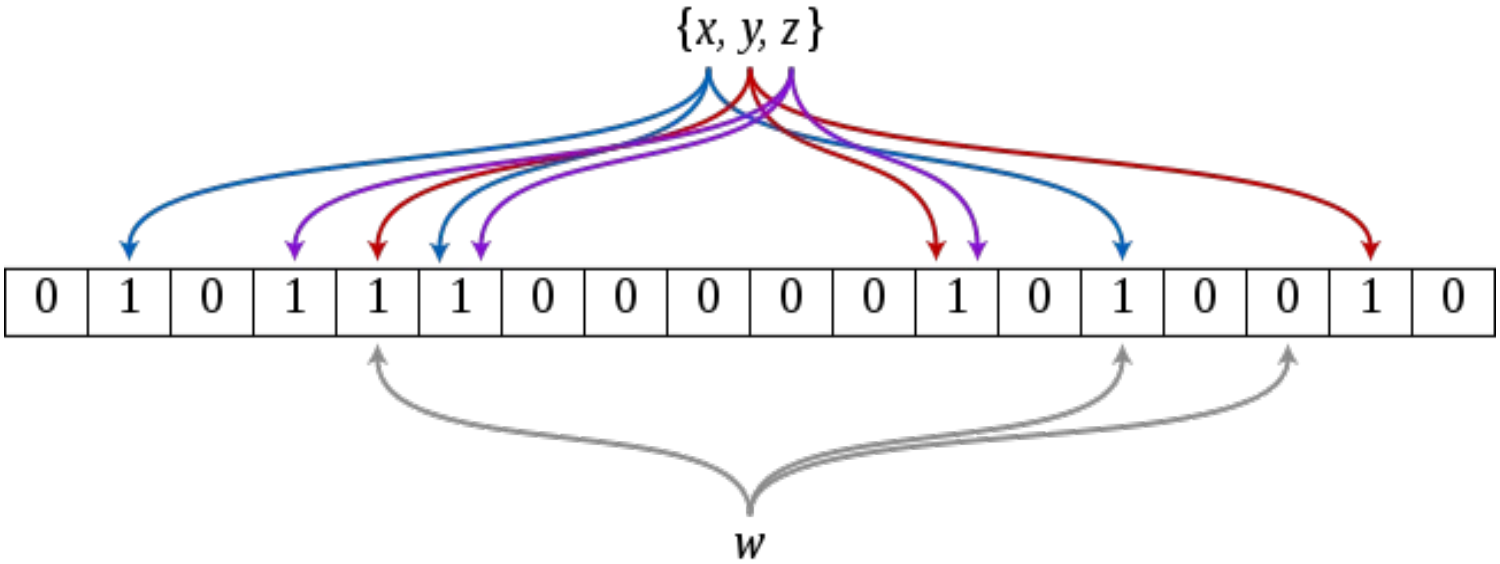
From WikiPedia (Don't tell my kid's teacher!)

"A Bloom filter is a space-efficient probabilistic data structure, conceived by Burton Howard Bloom in 1970, that is used to test whether an element is a member of a set. False positive matches are possible, but false negatives are not, thus a Bloom filter has a 100% recall rate"

Hashing

- Apply 'x' hash functions to the key to be stored/queried
- Each function returns a bit to set in the bitset
- Mathematical equations to determine how big to make the bitset, how many functions to use and your acceptable error level
- http://hur.st/bloomfilter?n=4&p=1.0E-20

Example



False Positives

- Perfect hash functions aren't worth the cost to develop
- Sometimes existing bits for a key are set by many other keys
- Make sure you understand the business impact of a false positive
- Remember, never a false negative

Why were we interested in Bloom Filters?

- Found a lot of places we went to the database to find the data didn't exist
- Found lots of places where we want to know if a user DIDN'T do something

Persistent Bloom Filters

- We needed persistent Bloom Filters for lots of user stories
- Found Orestes-BloomFilter on GitHub that used Redis as a store and enhanced it
- Added population filters
- Fixed a few bugs

Benefits

- Filters are stored in Redis
    - Only bitset/bitget calls to server
- Reads and updates of the filter from set of application servers
- Persistence has a cost, but a fraction of the RDBMS costs
- Can load a BF created offline and begin using it

Remember “For the cost of an Oracle License”

- Thousands of filters
- Dozens of Redis instances
- TTL on a Redis key makes cleanup of old filters trivial

Use Cases

White List/Black List

- Nightly rebuilds, real time updates as CEP finds something

Ad suggestions

- Does the system already know about this visitor?
- Maybe? More expensive processing. No? Defaults

Content Recommendations

- Batch identification hourly/nightly
- Real-time updates as 'hot' pages/content change the recommendation

Use Case – Cardinality Estimation

Client side joins are really difficult.

Hadoop, Spark, MongoDB – how do you know which side to 'drive' from?

We created a Population Bloom Filter that counts unique occurrences of a key using a Bloom Filter. Build a filter per 'source' file as the data is being collected (Kafka batches, Spark RDD saves etc.)

Now query the filter to see if the keys you are looking for are (possibly) in the data.

Then get the count and see which side to drive from.

Not ideal, low population may not mean it is the best driver, but could be

Use Case – A/B Testing

Use different algorithms and produce different result sets

Load into Redis in different keys (remember 10MM keys is no big deal)

Have the application tier A/B test the algorithm results

Conclusion

- Batch exports from SaaS (or any volume system) is bad
- Traditional messaging systems are being stressed by the volume of data
- Redis is a very fast, very simple and very powerful name value store “Data structure server”
- Bloom Filters have lots of applications when you want to quickly look up if one of millions of 'things' happened
- Redis-backed BloomFilters make updatable bloom filters trivial to use

SILVERPOP
An IBM® Company

References

- Redis.io
- Kafka.apache.org
- https://github.com/Baqend/Orestes-Bloomfilter
- http://www.slideshare.net/chriscurtin
- @ChrisCurtin on twitter

Questions?