cloudera®

Scaling Flume+Kafka to 1million Events Per Second

Tristan Stevens
Senior Solutions Architect
Cloudera, UK

About me

Tristan Stevens

- Senior Solutions Architect with Cloudera Professional Services
- At Cloudera for 2½ years
- Customers in Banking, Telecoms and Defence (+others)
- Security Specialist

- Apache Flume contributor, specifically around Apache Kafka integration
- Apache Hadoop contributor

cloudera

Project Brief

- Vodafone UK one of the UK's largest telecoms provider with 18.2m mobile customers.

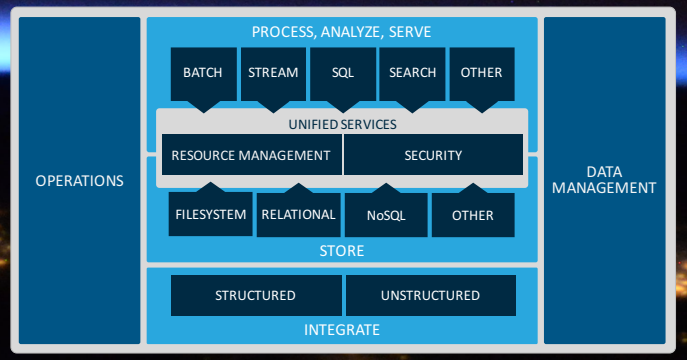
- Brief is to collect network events across fixed and mobile networks for:
  - Operations and fault monitoring
  - Threat intelligence
  - Incident response
  - Counter litigation

cloudera

Challenges

- No idea what the required ingest volumes could be.
- Need to understand log messages in order to bucket (by date and host).
- Over 100 sites in the UK from which to collect logs.
- High likelihood of non-RFC compliant messages from legacy devices.
  - RFC5424
  - RFC3164
  - Cisco ASA
  - Cisco PIX
  - …others

Regional Fan-in Architecture



Flume

Kafka Cluster

Regional DC1

Kafka Cluster

Regional DC2

Flume

Collection Site A

Flume

Collection Site B

Flume

Collection Site C

Flume

Collection Site D

Flume

Collection Site E

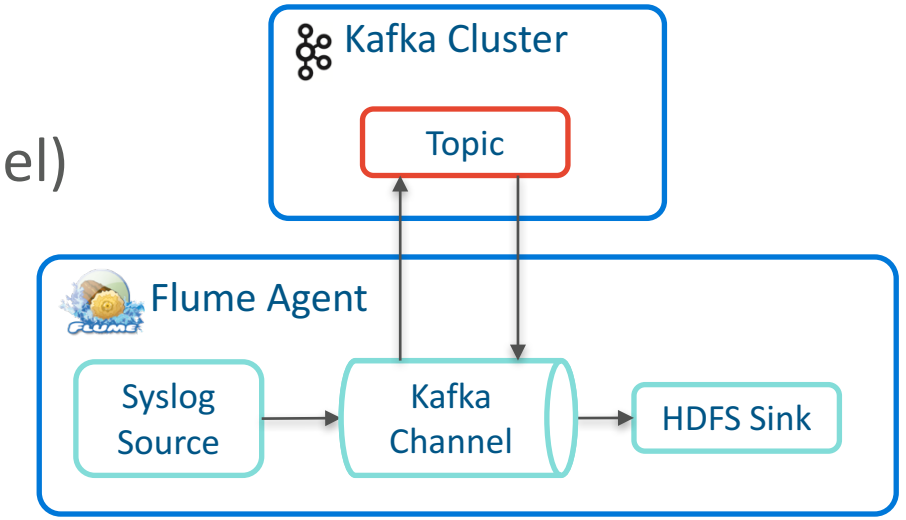
Flume

Collection Site F

About Flafka – It's Flume + Kafka

- Apache Flume has a basic Source➔Channel➔Sink topology.

- Three usages of Kafka clients:
  - Kafka Channel (use in place of File or Memory Channel)
    - Back-to-back Kafka Producer and Consumer
  - Kafka Source (a Kafka Consumer)
  - Kafka Sink (a Kafka Producer)



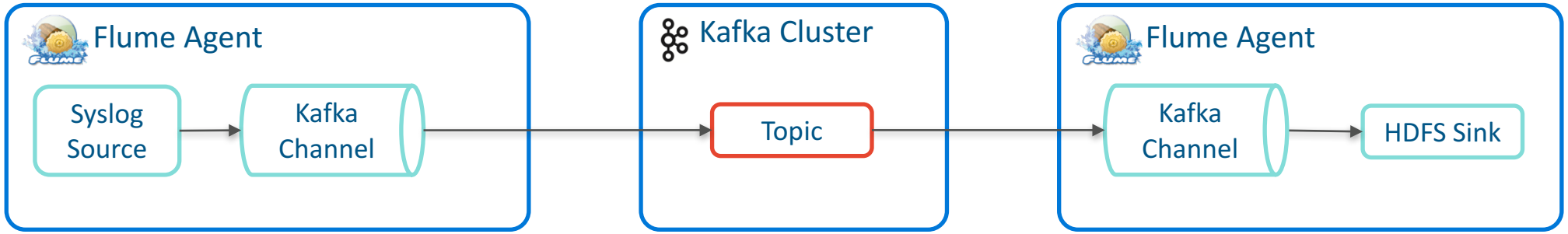
- Can also use the Kafka Channel without a Source or a Sink
  - For example: Run channels in different agents back-to-back

http://blog.cloudera.com/blog/2014/11/flafka-apache-flume-meets-apache-kafka-for-event-processing/

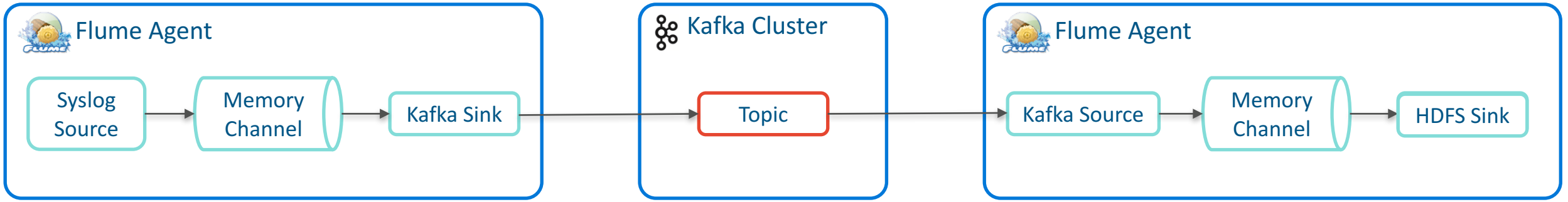
cloudera

Flafka – Channels, Sources and Sinks

- Back-to-back Kafka Channels

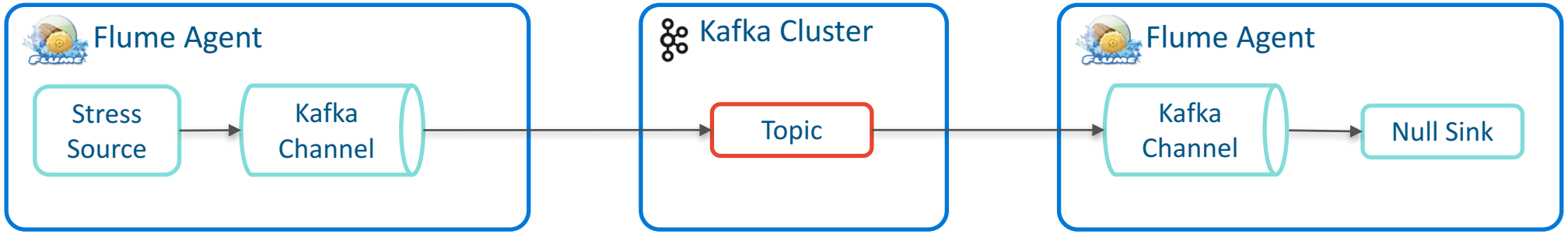


- Sink and Source topology



cloudera

Approach 1 – Kafka Channels



Benefits:

- Simple Configuration
- No memory channels
  - (Although no transactions either)
- Textbook approach

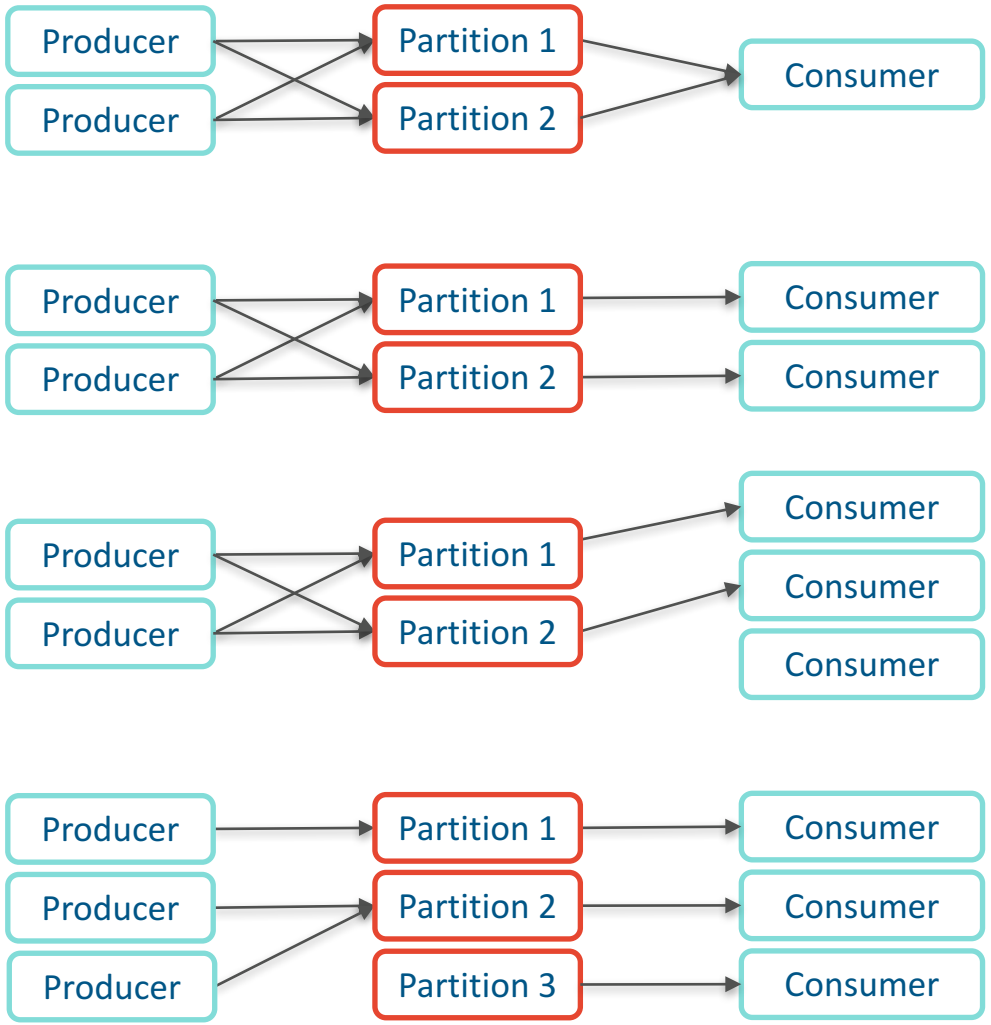
Results:

- First run 10,000 events per second.
- Increased batch sizes, got 100,000 EPS

- When replacing with HDFS Sink, the HDFS Sink became the bottleneck
- Adding HDFS sinks made no difference

cloudera

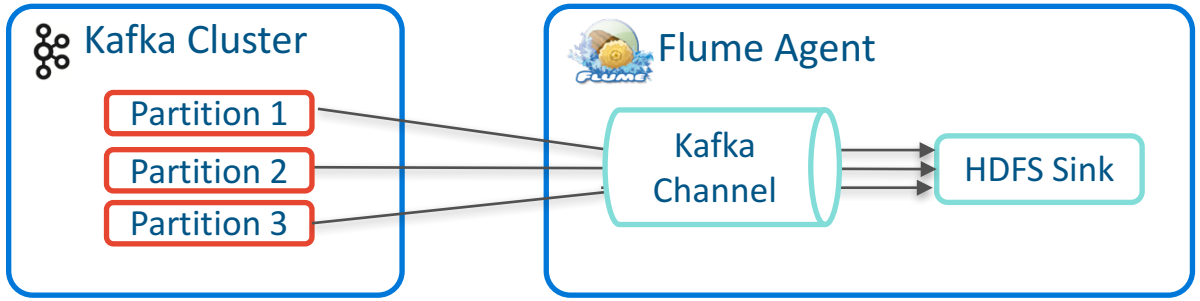
Partitioning Behaviour



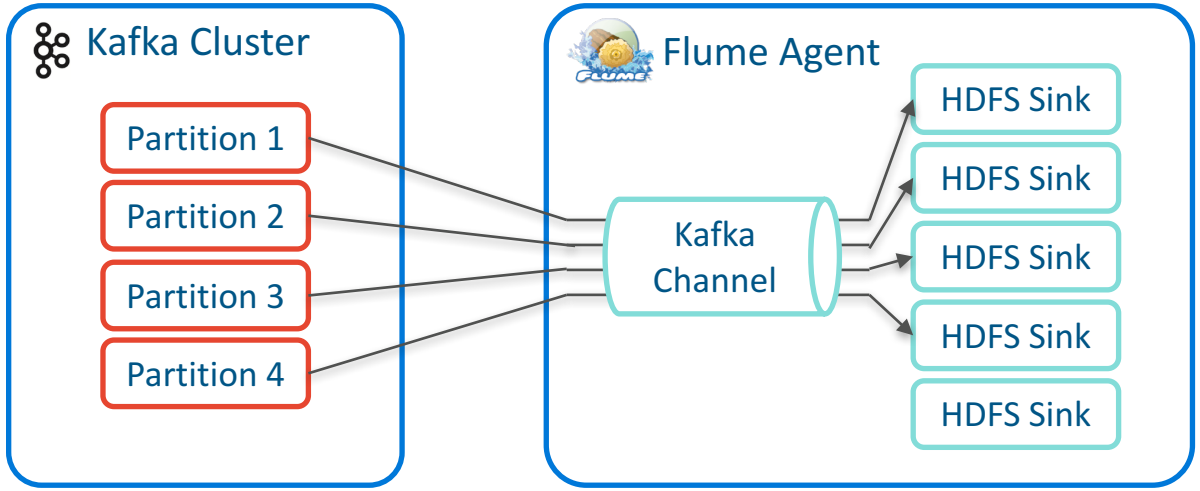
- A consumer can consume from one or more partitions
- A partition can only be consumed from by, at most, one consumer
- With the Default Partitioner (0.9+) Producers write to all Partitions
- In 0.8, partitions were randomly written to for 10 minutes at a time (KAFKA-1017) to avoid High Open File Handles

cloudera

Kafka Channel Partition Coupling



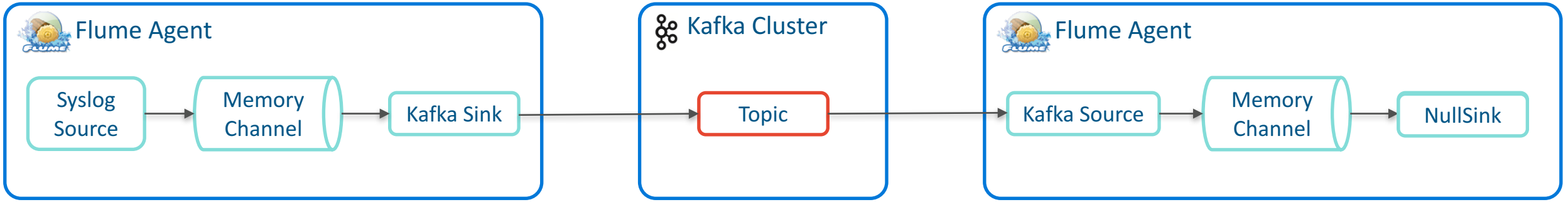
- When adding partitions, throughput didn't increase
- When adding sinks, throughput didn't increase



- Adding Sinks and partitions together doubled throughput
- Each Flume Sink becomes a Kafka Consumer
- Can leave orphaned Sinks ☹

cloudera

Approach 2 – Sinks and Sources



Features:

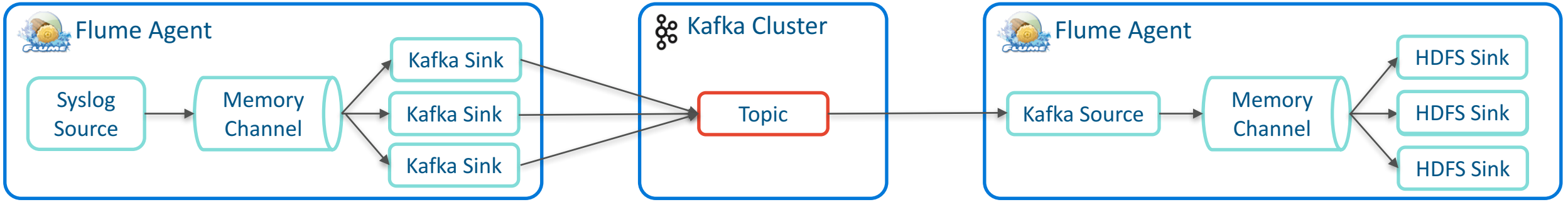
- Using Flafka Sink and Flafka Source in place of a split Kafka Channel
- Introduction of memory channels mean potential for data loss in the event of a Flume Agent Failure

Results:

- 120,000 EPS until second memory channel full – HDFS sink becomes the bottleneck. Replaced with null sink and sustained 120,000 EPS.

cloudera

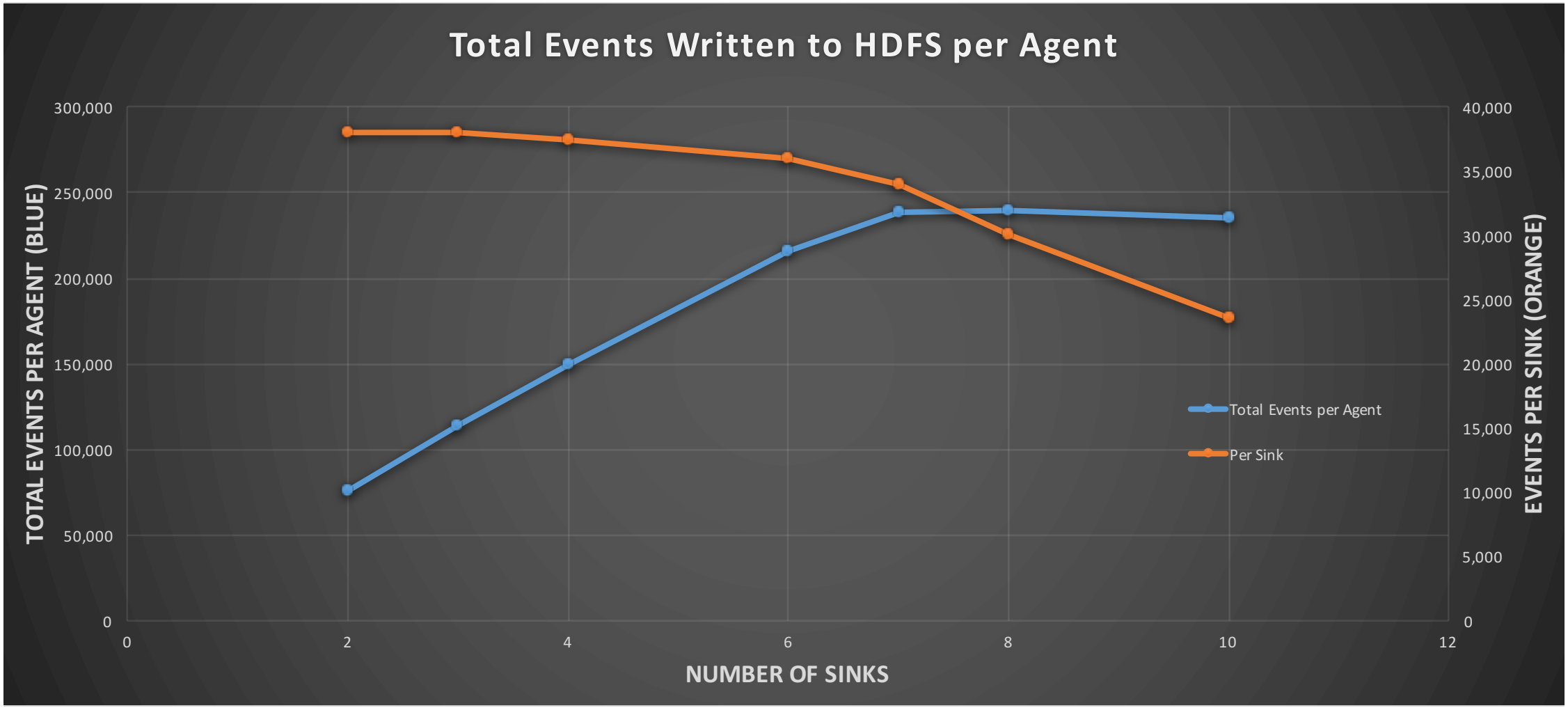
Approach 2 – Continued



- Adding additional HDFS sinks linearly increased HDFS write throughput, up to 7-8 sinks.
  - Wasn't impacted by number of partitions.

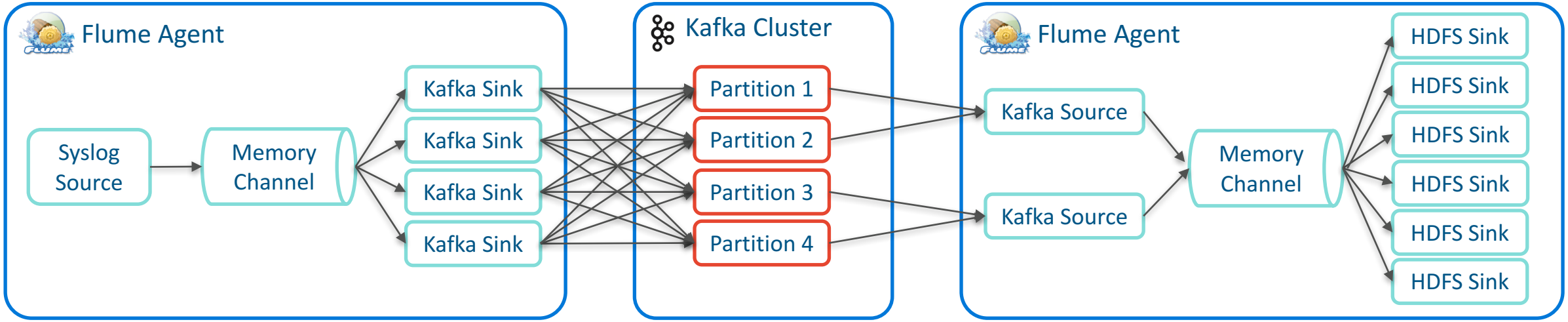
- Adding an additional Kafka Sink doubled to 240,000 events per second.
- Adding a third, capped at 256,000 events per second.

HDFS Sink Performance



N.B. Tested using 200 byte events

Four Scaling Factors



4. Use a custom Kafka partitioner, or use a static header (FLUME-2999) to scale the number of Kafka Sinks independently of the number of partitions and Sources

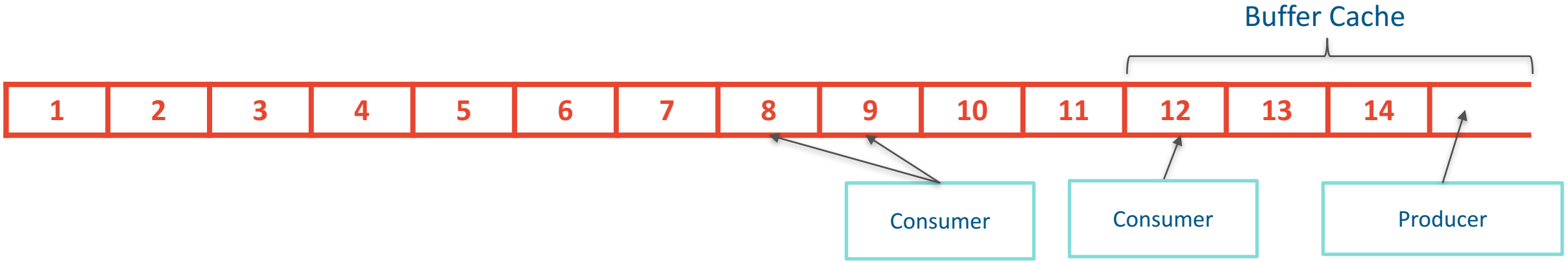
3. The number of partitions should be greater than the number of disks in the Kafka cluster divided by the replication factor. It MUST be greater than or equal to the number of Kafka Sources.

2. The number of Kafka sources may also be a bottleneck, so enough Sources need to be provisioned for the throughput required.

1. The number of HDFS sinks can be calculated from the throughput required and can be scaled independently.

cloudera

It’s all about the cache



- Kafka enjoys sequential IO and is heavily optimised to benefit from the OS buffer cache.
- Optimum performance is achieved when the consumers’ lags are low enough that read requests can be serviced from the buffer cache.
- When the disks are seeking to service reads, this interrupts the sequential write performance too.
- The hardware needs to be sized to ensure that there is sufficient memory to handle not only optimum throughput, but also catch-up scenarios.

cloudera

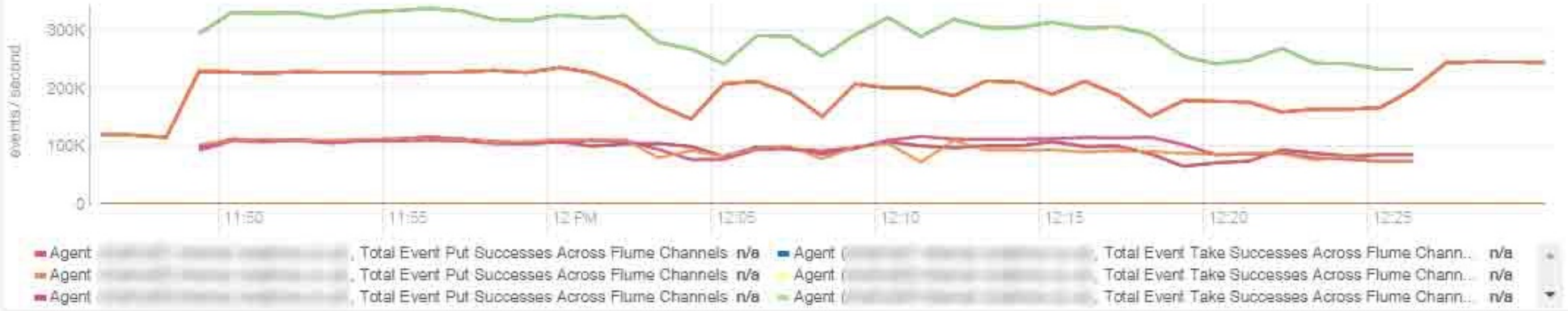
Flume Dashboard

44.4 minutes preceding December 3, 2015, 12:30

10:15 10:30 10:45 11 AM 11:15 11:30 11:45 12 PM 12:15 12:30 12:45 01 PM 01:15 01:30 01:45 02 PM 02:15

30m 1h 2h

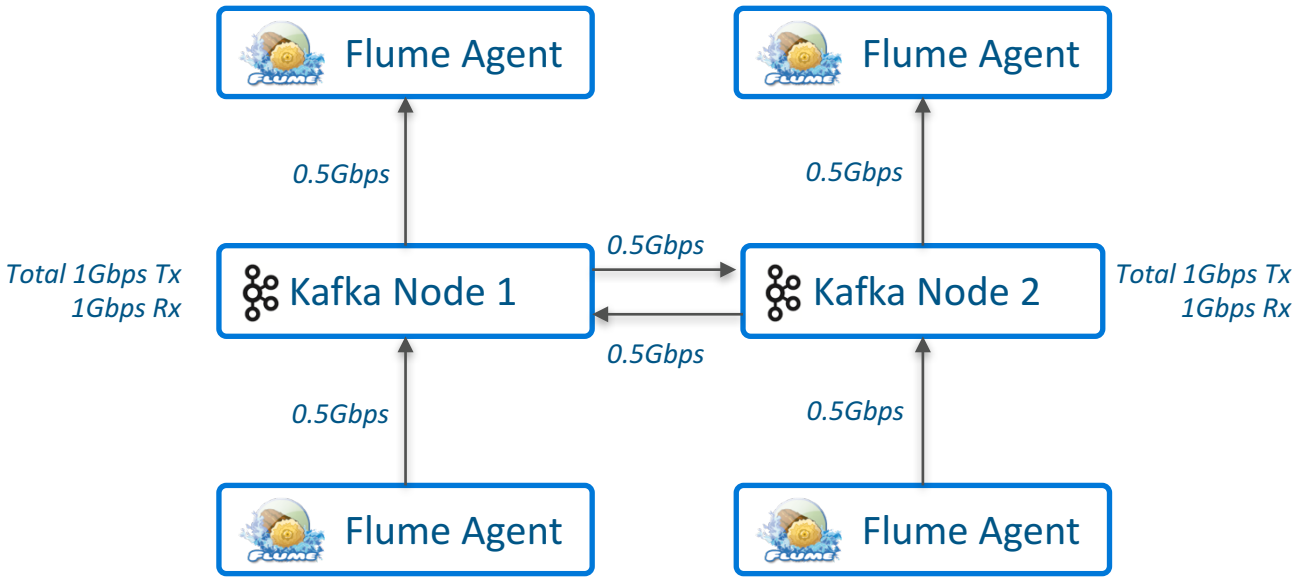
Puts and Takes



Agent [redacted], Total Event Put Successes Across Flume Channels n/a
Agent [redacted], Total Event Put Successes Across Flume Channels n/a
Agent [redacted], Total Event Put Successes Across Flume Channels n/a
Agent [redacted], Total Event Take Successes Across Flume Chann.. n/a
Agent [redacted], Total Event Take Successes Across Flume Chann.. n/a
Agent [redacted], Total Event Take Successes Across Flume Chann.. n/a

Total Event Take Successes Across Flume Channels Edge Nodes

Network



- We quickly found that the network was the bottleneck.
- For a replication factor of *r*, then *r*-1/*r* of a node's network traffic will be replication traffic (e.g. ½ at *r*=2, ⅔ at *r*=3).
- Under high network load scenarios we also saw the leaders lose touch with the followers and mark a partition replica as unavailable and creating a new one (which had to be sync'd from the leader).
- This lead to even more network load and therefore reduced read/write rates (after a period).

Recommendations:

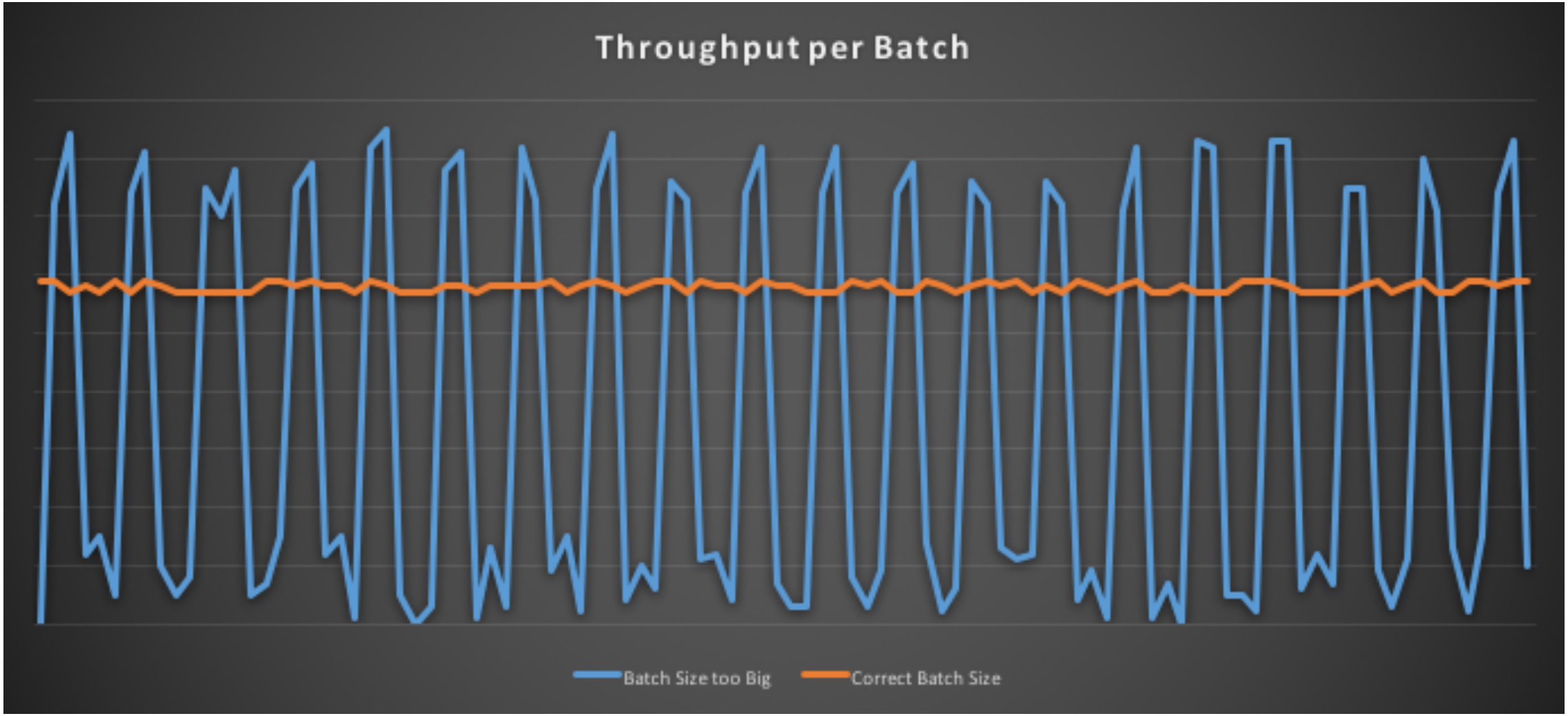
- Calculate your network requirements.
- Use non-blocking Top-of-Rack switches.
- Do not push to its limits – leave spare capacity to deal with failure scenarios.
- N.B. Use Snappy to decrease network load. On syslog data, expect 3-4x compression.

cloudera

Importance of Batching

- Flume and Kafka are both batch oriented.
- Each commit or transaction puts a load onto the system, invariably by requiring synchronisation of threads (both in terms of Flume and Kafka). Each batch has an overhead (headers etc).
- Large batches cause lumpy throughput (turbulence) and don't optimise resources (IO, CPU). Ideally we want a laminar flow.
- Need a batch size that keeps the resources busy enough, but avoids the overheads associated with too many batches.
- Batching in Kafka 0.8 was much simpler than in Kafka 0.9.
  - Synchronous producer is gone from 0.9.

Batch Sizing



cloudera

Kafka 0.9 Batch Control

- `kafka.flumeBatchSize` – The size of the batch the Sink takes from the Flume channel in a single transaction (default 1000).
- `kafka.producer.batch.size` – The size of the batch committed (asynchronously) to Kafka (default 16384 *bytes*)
- `kafka.producer.linger.ms` – The amount of time the producer will wait before sending a batch (default *0* ms)
- `kafka.producer.max.request.size` – Limits the size of a batch (and therefore the size of a message) that can be sent (default 1048576 *bytes*).

Consumer Batch Control

- `kafka.consumer.fetch.min.bytes` – Defaults to 1, but can be increased to improve batch sizes.
- `kafka.consumer.max.partition.fetch.bytes` – This limits the size of a batch (and hence the size of a message) that can be received. Impacts the amount of memory used by the consumer.
- `kafka.consumer.auto.commit.interval.ms` – Ensures that the load on the consumer offsets topic is not too high

Other Performance Optimisations

- `kafka.producer.compression.type` - Valid values are: `none`, `gzip`, `snappy`, or `lz4`
  - Compression applies to each batch rather than each message, so larger batches benefit more, although this depends on your data. Note: this will increase CPU usage on producer, brokers and consumers, and may result in unnecessary messages being retrieved by the consumer.
- `kafka.producer.acks` - Setting this to 1 improves performance, however reduces durability in the event of a failure. Note: When a broker is overloaded it is possible for a partition replica to go offline, therefore often this should be set to 2 or to -1/all.

Security

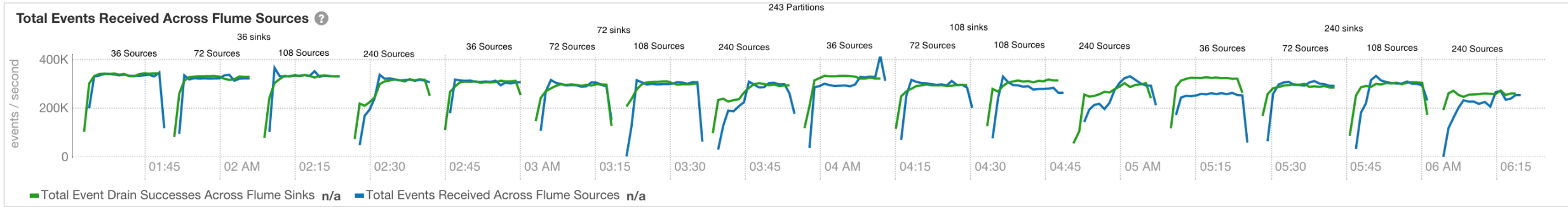
- Kafka 0.9 introduced security:
  - Authentication (Kerberos, TLS client auth, Username/password in 0.10)
  - Authorization (ACLs or Apache Sentry integration)
  - Wire-encryption (TLS)

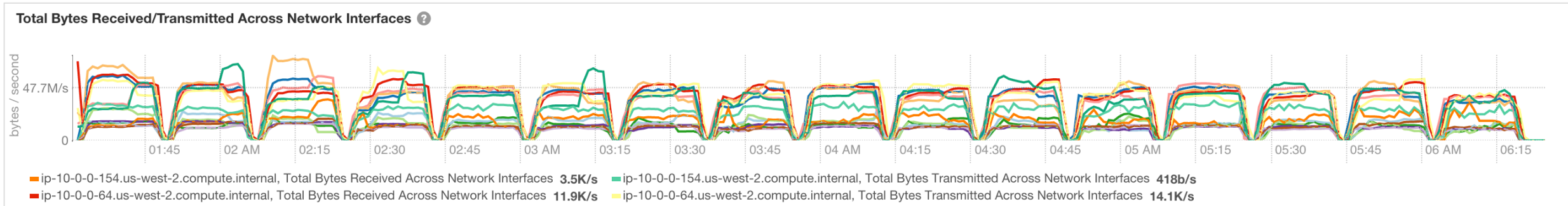
- All 0.9 security features supported in Flume 1.7 / CDH5.8
- Note: TLS introduces severe CPU overhead and degrades performance (KAFKA-2561)

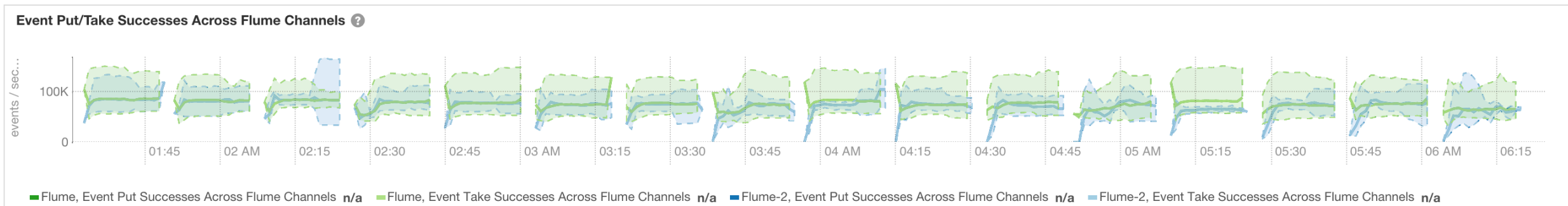
Bringing it all together

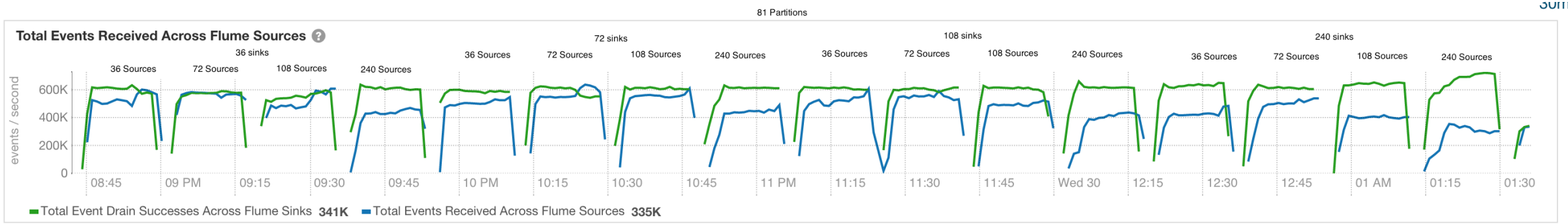
- Use the heuristics from the previous pages to give you a range and then use empirical analysis.

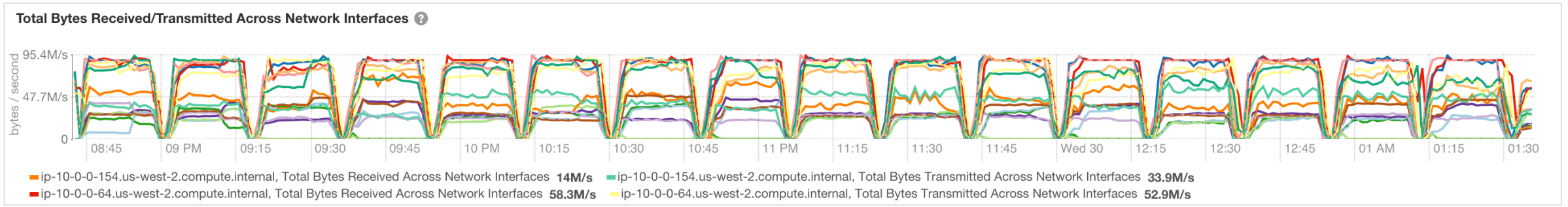
- Used the CM API to loop through different scenarios for:
  - Number of Sinks
  - Number of Sources
  - Number of Partitions
  - Batch control settings
- And permutations thereof.

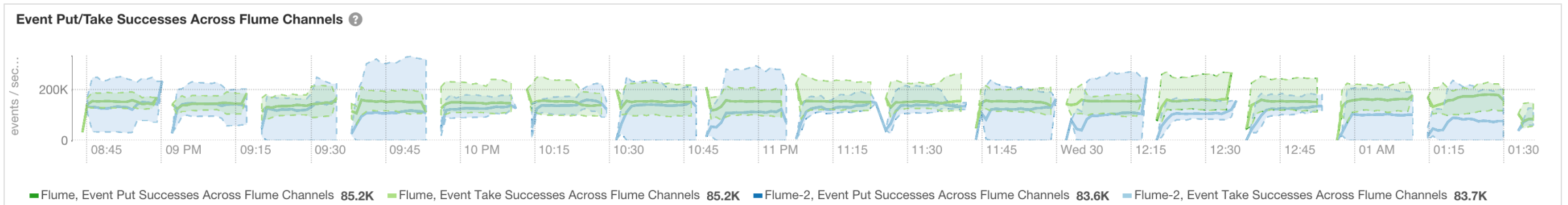






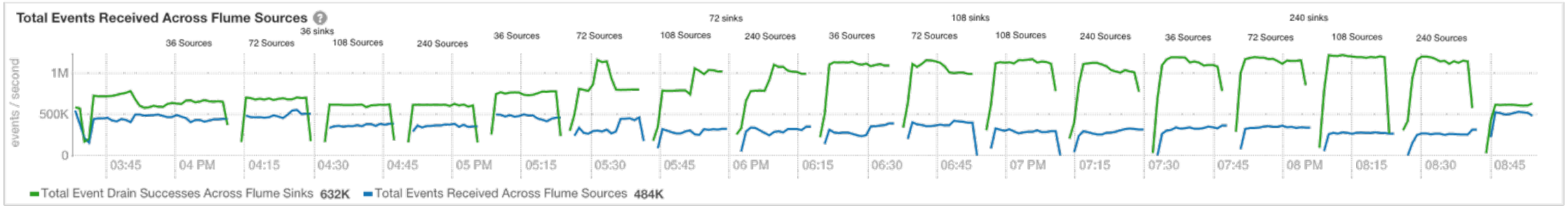


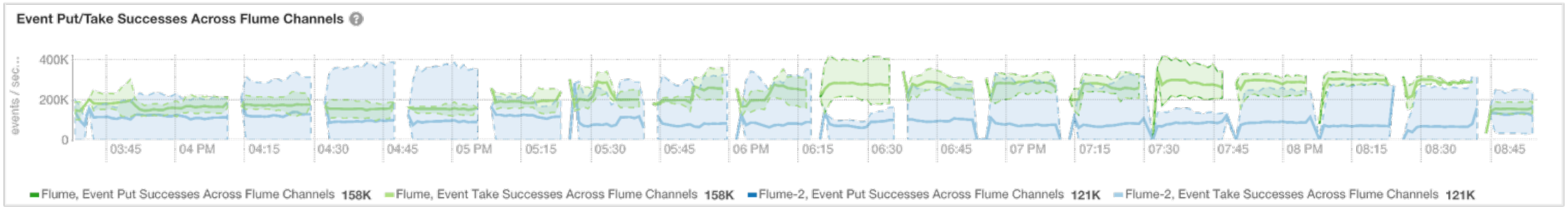


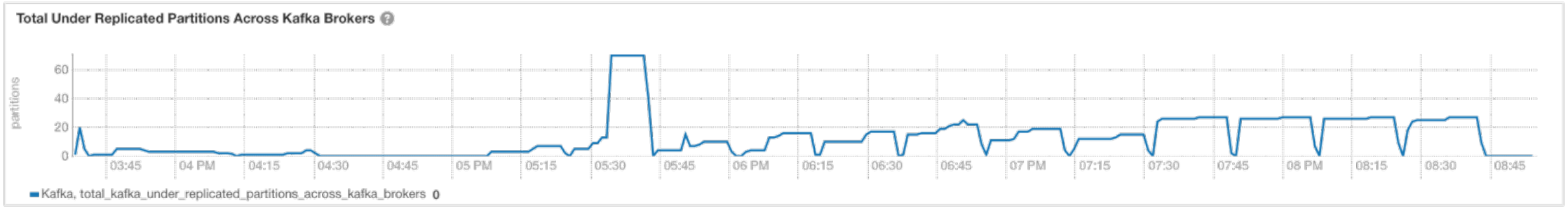


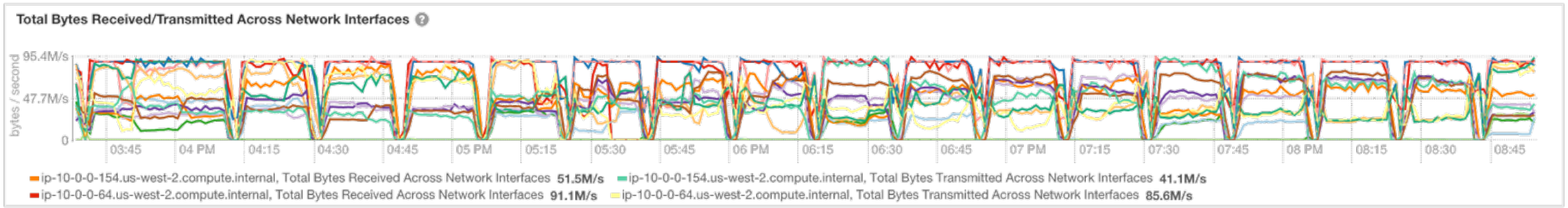
27 Partitions

30m



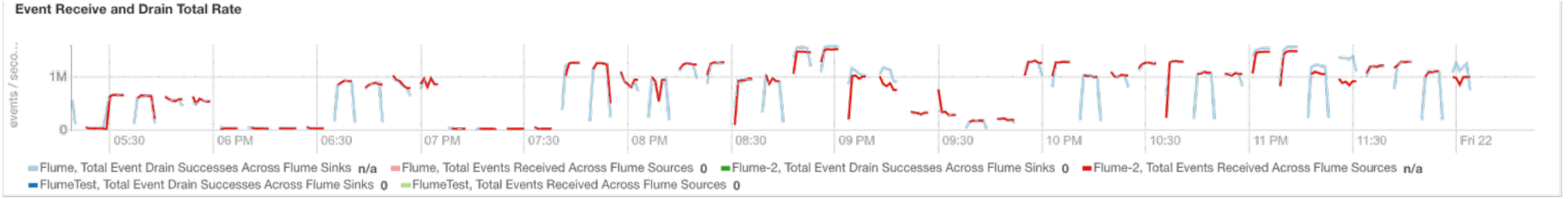






Does this matter?

- Using default values: 130,000 EPS
- Using bad values: 26,000 EPS
- Using optimal values: 1,300,000 EPS



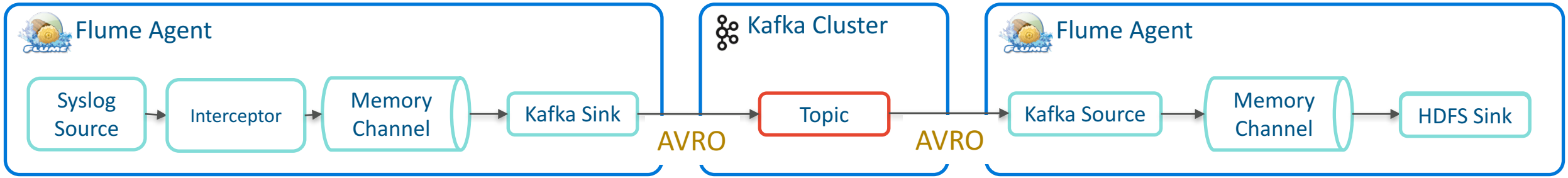
cloudera

Final Results

- Using 10Gbps networking we managed to top 1.2 million events per second.
- Includes putting Syslog Sources back in (tops at 300,000 EPS) and 40 HDFS sources (across 4 agents).
- Each Kafka Source runs at around 20-30,000 EPS to give latent capacity for failure scenarios and to ensure that each partition isn't running too hot.

What Next?

- Already loading data to HDFS for batch analysis and system-of-record storage.
- Candidate Capabilities:
  - Spark-Streaming for in-flight streaming analytics
  - Join with other data sources
  - Stream subset of data into Solr for dashboarding
  - Use Morphlines or custom interceptor for field extraction
    - Do this at the collectors and then pass as Avro through to EDH tier (FLUME-2852)



cloudera

Further Reading

- https://blog.cloudera.com/blog/2016/03/building-benchmarking-and-tuning-syslog-ingest-architecture/

- https://blog.cloudera.com/blog/2016/08/new-in-cloudera-enterprise-5-8-flafka-improvements-for-real-time-data-ingest/

cloudera

cloudera

Thank you

Tristan Stevens

Cloudera, UK


@tmgstevens


https://www.linkedin.com/in/tristanstevens/