Real Time Aggregation with Kafka ,Spark Streaming and ElasticSearch , scalable beyond Million RPS

Dibyendu B
Dataplatform Engineer, InstartLogic

Who We are

Making web and mobile applications fast, secure, and easy to operate.

- JS Streaming
- HTML Streaming
- Image Optimization
- Machine Learning
- Application Virtualization
- Intelligent CDN

80+

patents

- Original research
- Experts from Google, VMWare, Amazon, Twitter

$140M

invested

ANDREESSEN HOROWITZ

KPCB KLEINER PERKINS CAUFIELD BYERS

HERMES GROWTH PARTNERS

Stanford University

greylockpartners.

TELSTRA

GEODESIC CAPITAL

TENAYA CAPITAL

SUTTER HILL VENTURES

10+

awards

JMP SECURITIES HOT 100 SOFTWARE COMPANIES

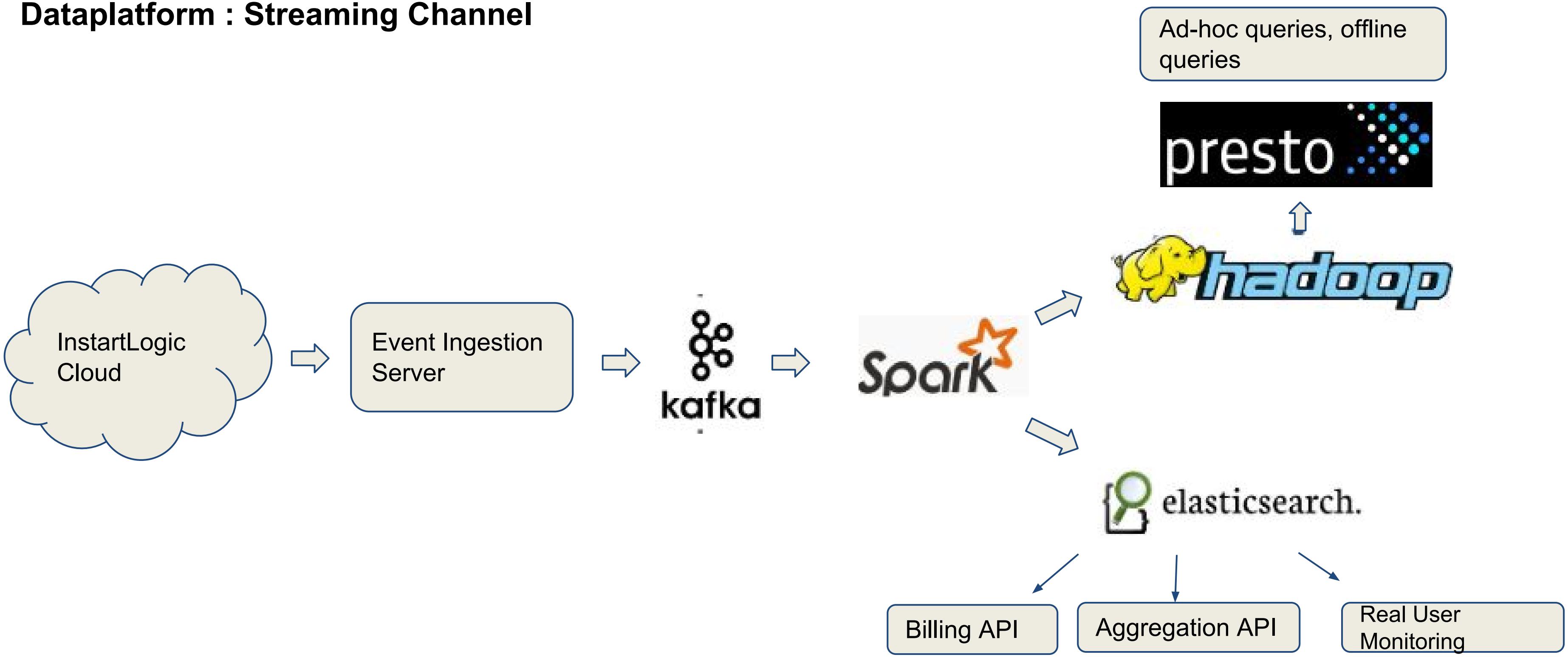
RED HERRING GLOBAL 100 WINNER

Gartner 2014 Cool Vendor

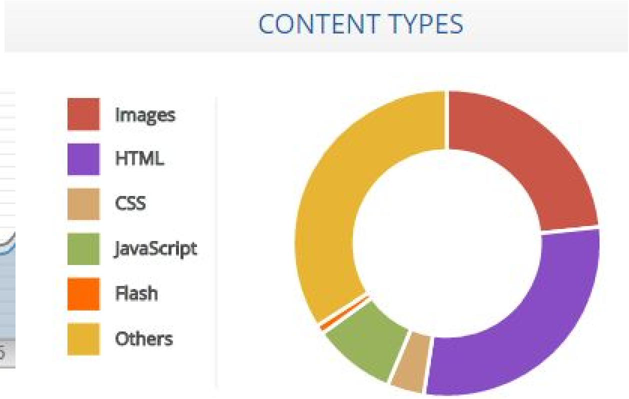
Best Place to Work

STACKWORLD CONFERENCE & EXPO — Endpoint-Aware Application Delivery Solution — TOP 25 ScaleUp Awards

Dataplatform : Streaming Channel

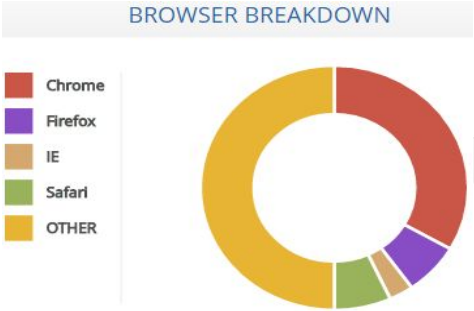


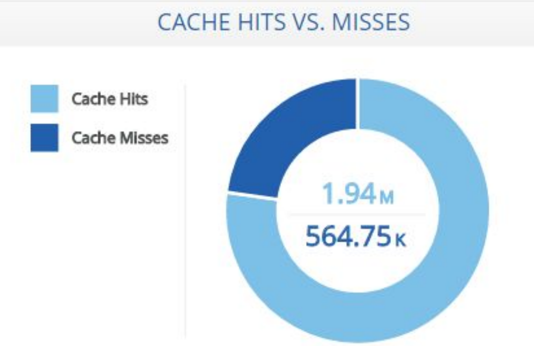
What We Aggregate



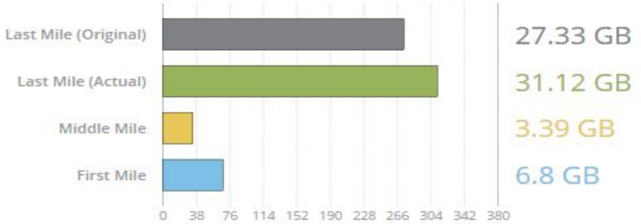
BROWSER TYPES

+ WINDOWS (41.82%)	1,644,668 Total
+ IOS (22.79%)	896,433 Total
+ LINUX (16.53%)	649,942 Total
+ OTHERS (16.51%)	649,273 Total
+ UNKNOWN (2.36%)	92,678 Total





	BANDWIDTH	PERCENTAGE	CONTENT TYPE
1	17.67 GB	42.78%	HTML
2	6.34 GB	15.34%	WebP
3	5.25 GB	12.71%	JavaScript
4	4.71 GB	11.40%	JPEG
5	2 GB	4.84%	PNG
6	1.96 GB	4.74%	Application/json



instartlogic

We Aggregate on :

Aggregate Metrics

on different Dimensions

for different Granularity


```
{
    "_index": "stats_index_v3_hourly_2016-11-14",
    "_type": "stats",
    "_id": "8511131658923208013",
    "_version": 13,
    "_score": null,
  ▼ "_source": {
      "document_id": 8511131658923208000,
      "fromTimestamp": 1479139200,
      "toTimestamp": 1479142800,
      "grouping": "accesslog_hour/host/country/customer/",
    ▼ "dimensions": {
        "host": "www.1800petmeds.com",
        "country": "United States",
        "customer": "1800petmeds"
      },
    ▼ "metrics": {
        "hits": 717091,
        "original_bandwidth_bytes": 20025223984,
        "middle_mile_bandwidth_bytes": 6780100067,
        "last_mile_bandwidth_bytes": 10528499146,
        "first_mile_bandwidth_bytes": 7173548320,
        "time_to_last_byte_ms": 35960341,
        "request_time_ms": 42242244,
        "time_to_first_byte_ms": 18599604
      }
    },
```

Dimensions

```yaml
- [ content_type ]
- [ content_type, cache_status, property ]
- [ content_type, property ]
- [ country ]
- [ country, browser_name ]
- [ country, browser_name, device_type, property ]
- [ country, browser_name, property ]
- [ country, cache_status ]
- [ country, cache_status, property ]
- [ country, content_type, property ]
- [ country, device_os, browser_name ]
- [ country, device_os, browser_name, device_type, property ]
- [ country, device_os, browser_name, property ]
- [ country, device_type, property ]
- [ country, property ]
- [ country, scheme, property ]
- [ device_os, browser_name, device_type, property ]
- [ device_os, browser_name, property ]
- [ device_type, property ]
- [ gna_cache_status ]
- [ host ]
- [ host, browser_name ]
- [ host, cache_status ]
- [ host, content_type ]
- [ host, content_type, cache_status ]
- [ host, country ]
- [ host, country, browser_name ]
- [ host, country, cache_status ]
- [ host, country, content_type ]
- [ host, country, device_os, browser_name ]
- [ host, country, device_os, browser_name, device_type ]
- [ host, country, device_type ]
- [ host, country, scheme ]
- [ host, device_os, browser_name ]
- [ host, device_os, browser_name, device_type ]
```

We have configurable way to define what all Dimension are allowed for given Granularity

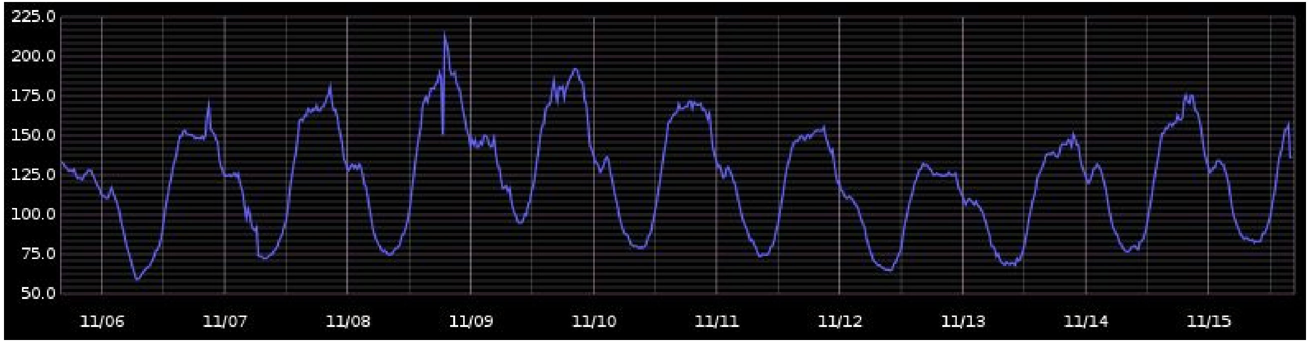
This example for DAY Granularity

Similar Set Exists for

HOUR and MINUTE

Let see the challenges of doing Streaming Aggregation on large set of Dimensions across for different Granularities

Some Numbers on volume and traffic



Streaming Ingestion ~ 200K RPS

50 MB / Seconds ~ 4.3 TB / Day

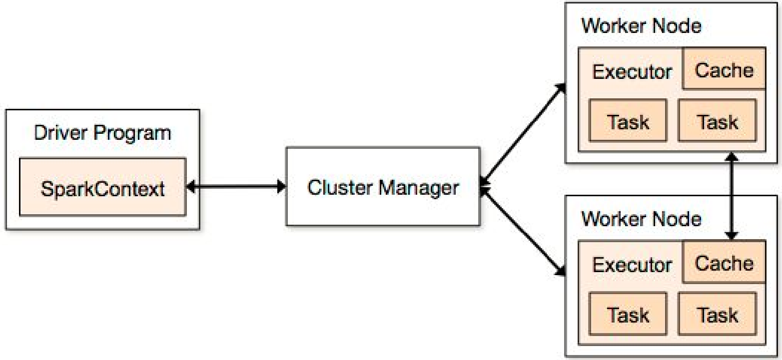
Streaming Aggregation on 5 min Window.

- 60 million Access Log Entries within 5 min Batch
- ~100 Dimensions across 3 different Granularities.
- Every log entry creates ~ 100 x 3 = 300 records

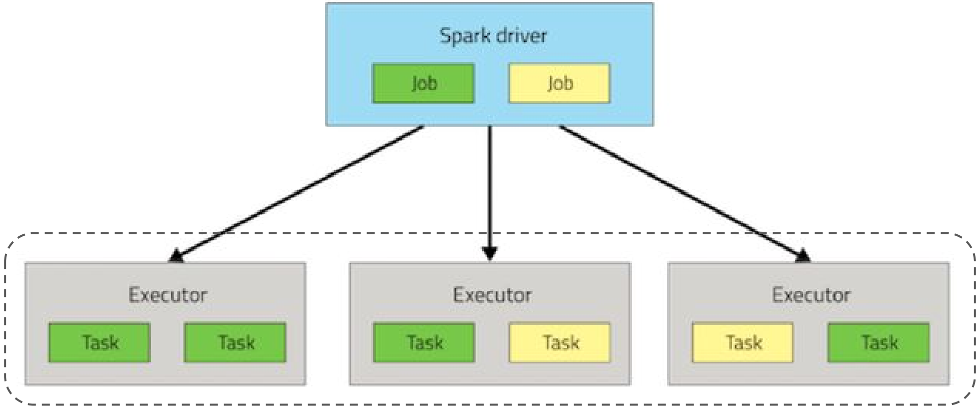
Key to handle such huge aggregations within 5 min window is to aggregate at stages..

Multi Stage Aggregation using Spark and Elasticsearch…

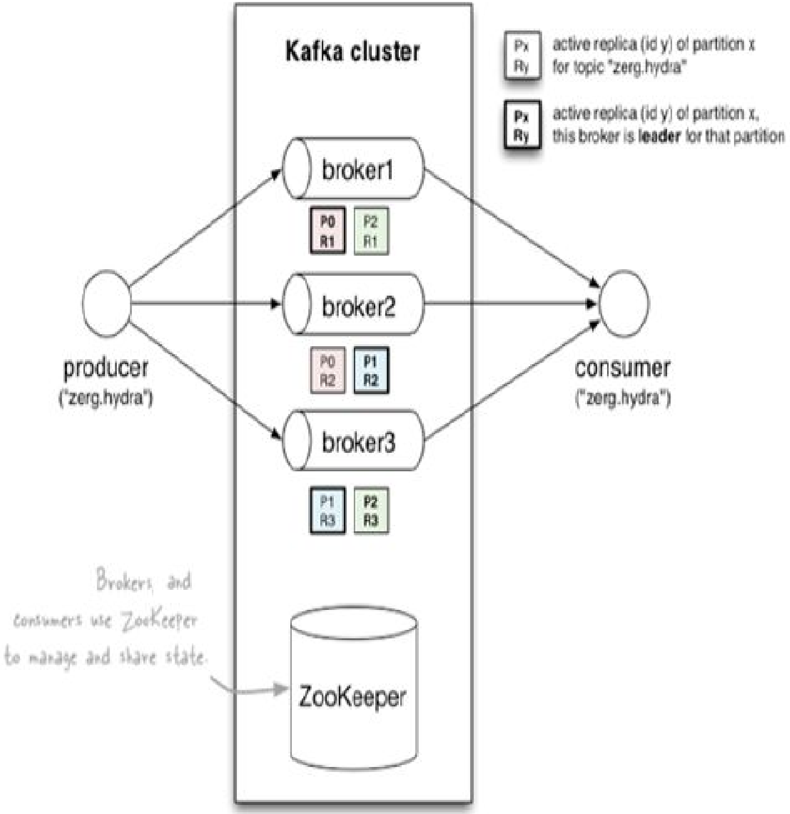
Spark Fundamentals



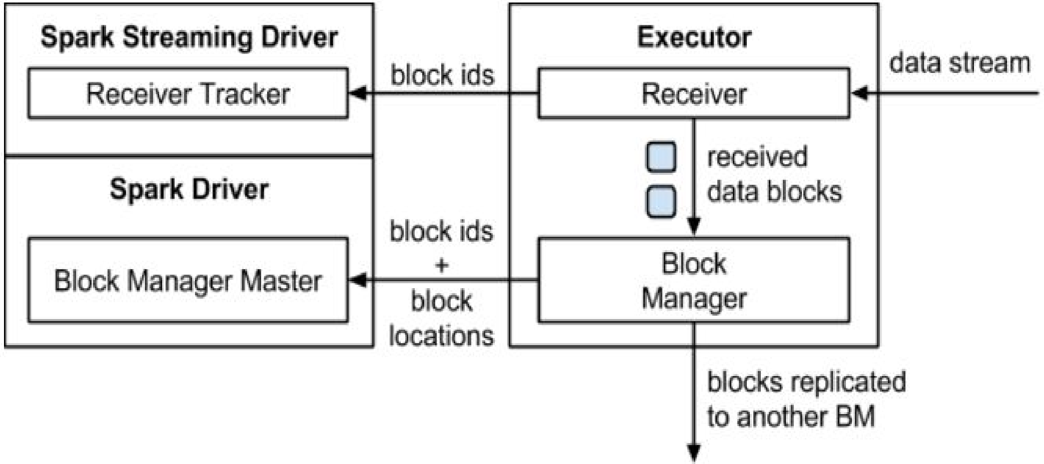
- *Executor*
- *Worker*
- *Driver*
- *Cluster Manager*

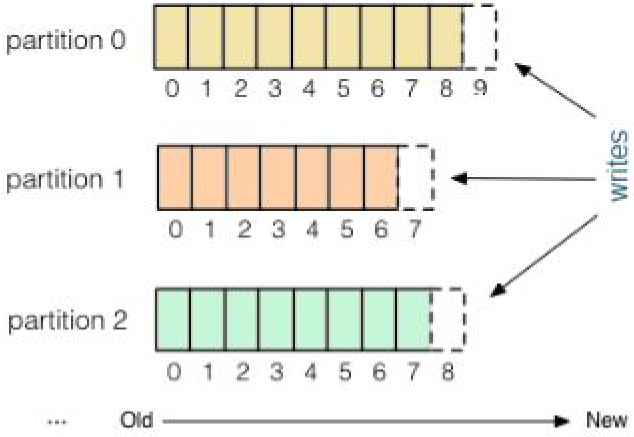


Kafka Fundamentals

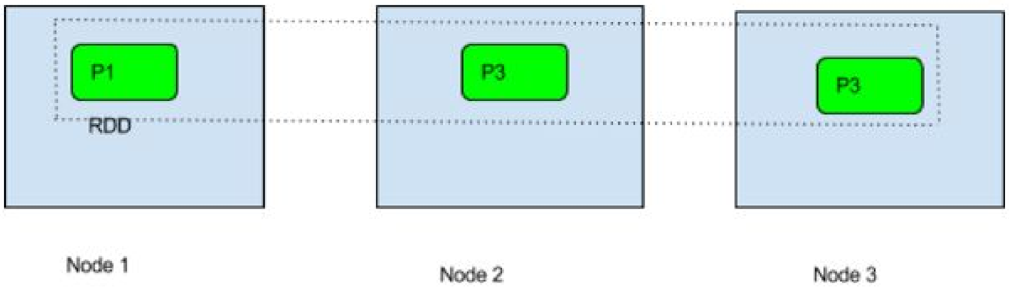


Kafka and Spark

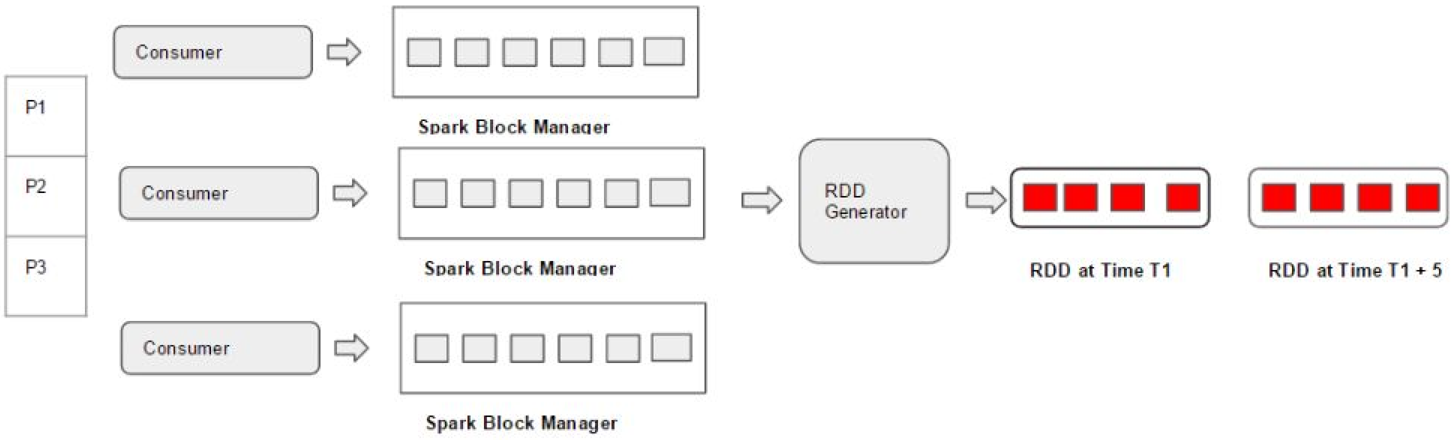




Spark RDD ..Distributed Data in Spark



How are RDDs generated ? ..Let’s understand how we consume from Kafka



Kafka Consumer for Spark

Apache Spark has in-built Kafka Consumer but we used a custom high performance consumer

I have open sourced Kafka Consumer for Spark Called Receiver Stream

(https://github.com/dibbhatt/kafka-spark-consumer)

It is also part of Spark-Packages : https://spark-packages.org/package/dibbhatt/kafka-spark-consumer

Receiver Stream has better control on Processing Parallelism.

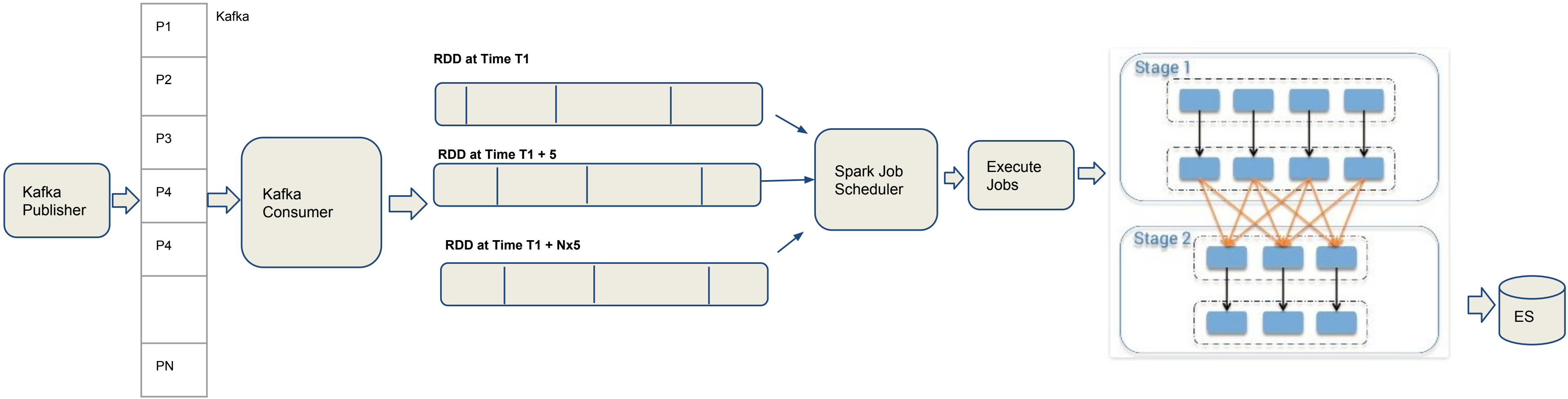
Receiver Stream has some nice features like

Receiver Handler, Back Pressure mechanism,
WAL less end to end No-Data-Loss.

Receiver Stream has auto recovery mechanism from failure situations to keep the streaming channel alway up.

Contributed back all major enhancements we did in Kafka Receiver back to spark community.

Streaming Pipeline Optimization



Too many variables to tune :

How many Kafka Partitions ?
How many RDD Partitions ?
How much Map and Reduce side partition ?
How much network Shuffle ? How many stages ?
How much spark Memory, CPU cores, JVM Heap , GC overhead , memory back-pressure,
Elasticsearch optimizations , bulk request, retry , bulk size , number of indices, number of shards ..

And so on..

Revisit the volume

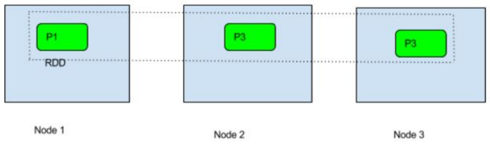
Streaming Ingestion ~ 200K RPS peak rate and growing

Streaming Aggregation on 5 min Window.

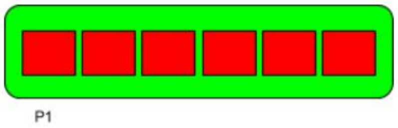
- ***60 million*** Access Log Entries within 5 min Batch
- 100 Dimensions across 3 different Granularities.
- Every log entry creates ~ 100 x 3 = 300 records
- ***~ 20 billion*** records to aggregate upon in a single window.

Key to handle such huge aggregations within 5 min window is to aggregate at stages..

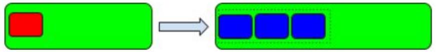
Aggregation Flow



Consumer Pulls compressed access log entries Kafka



Every compressed entries has N individual logs

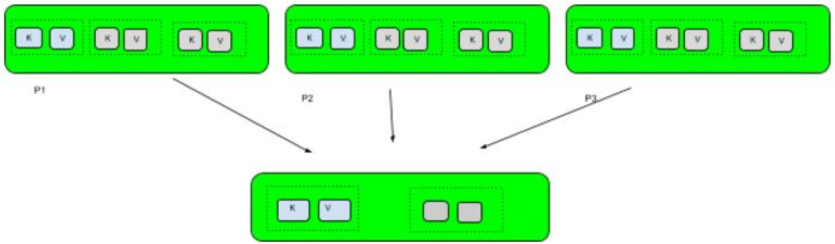


Every log fan-out to multiple records (dimensions/granularity)

Every record is (key,value) pair

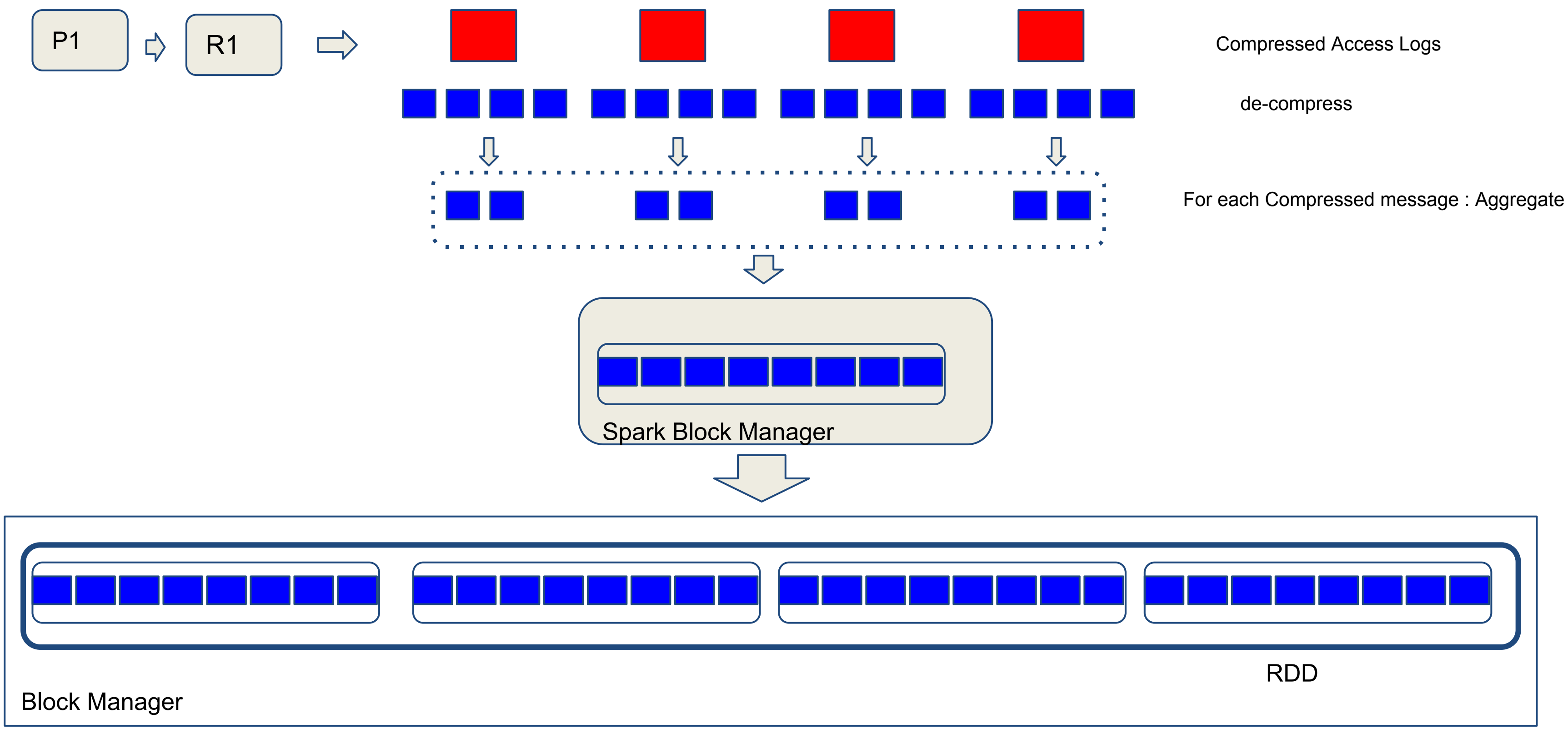
Map : Per Partition logic

Shuffle

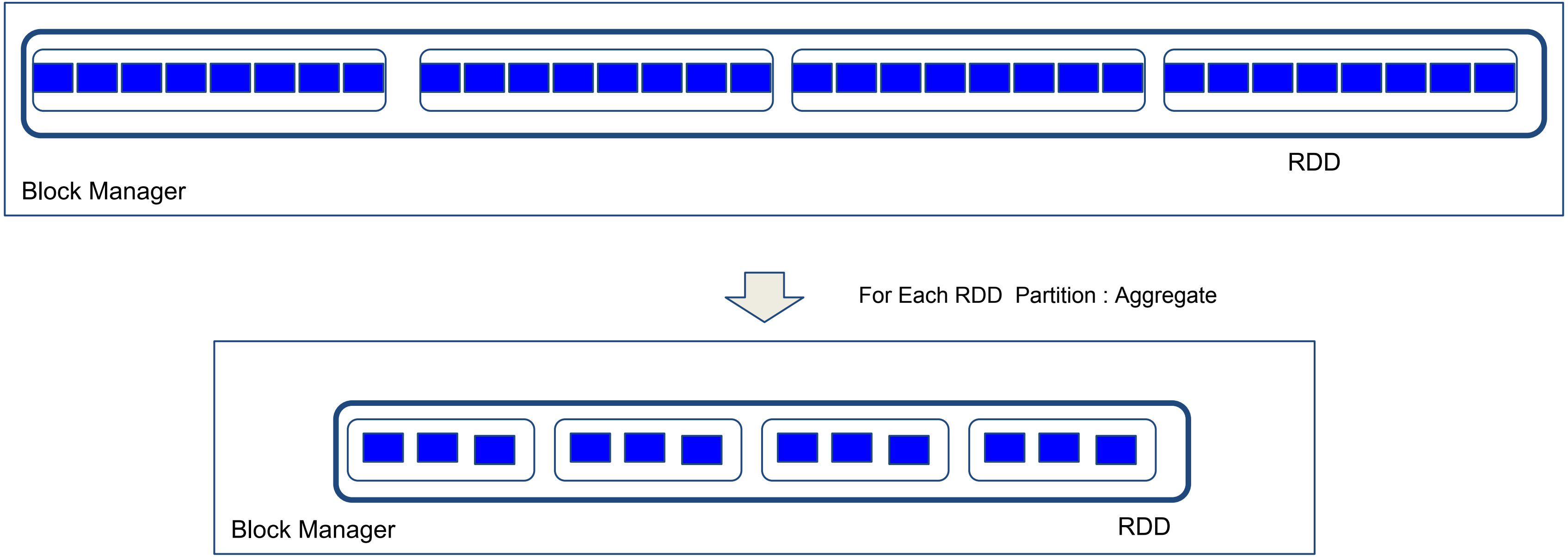


Reduce : Cross Partition logic

Stage 1 : Aggregation at Receiver Handler



Stage 2 : Per Partition Aggregation : Spark Streaming Map

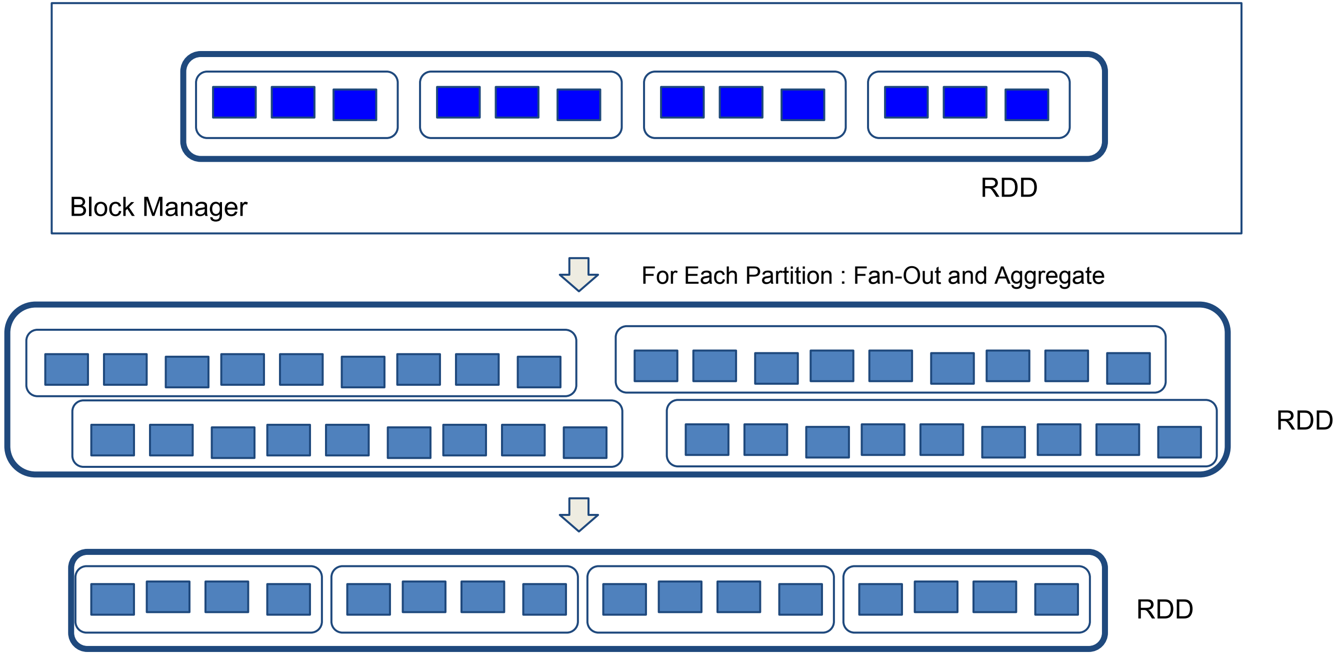


During Job run we observed Stage 1 and Stage 2 contributes to ~ 5 times reduction in object space.

E.g. with 200K RPS, 5 min batch consumes ~60 million access logs , and after Stage 1 and 2 , number of aggregated logs are around ~ 12 millions.

What is the Key to aggregate upon ?

Stage 3 : Fan-out and per-partition aggregation : Map

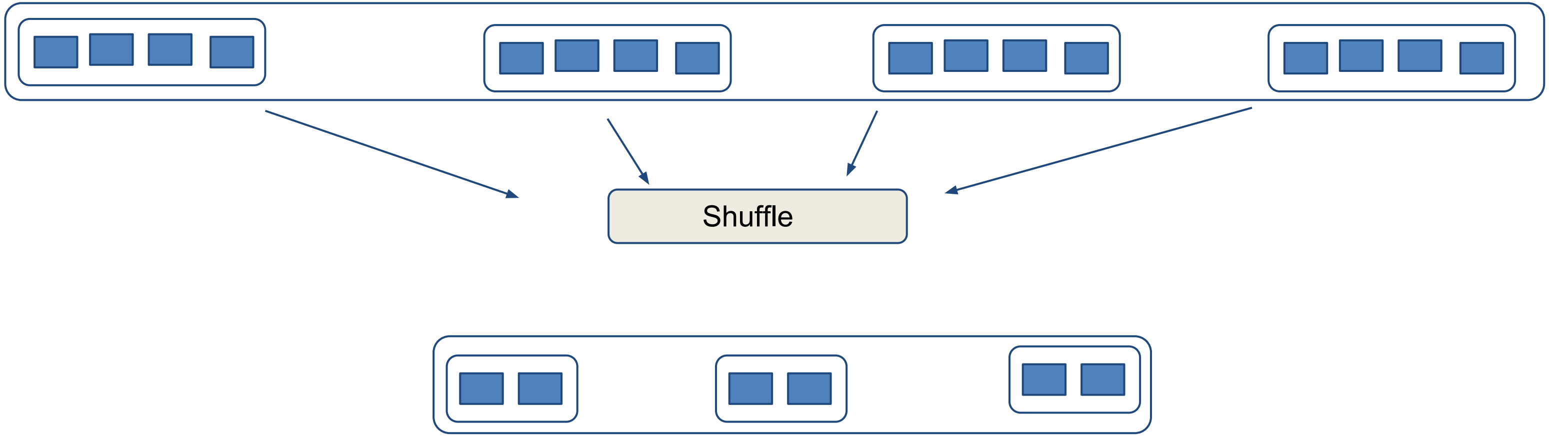


During Job run we observed Stage 3 contributes to ~ 8 times increase in object space. Note : Fan-out factor is 3 x 100 = 300

After stage 1 and stage 2 , number of aggregated records are around ~ 12 million. number of records after Stage 3 ~ 80 million

What is the key for aggregation ?

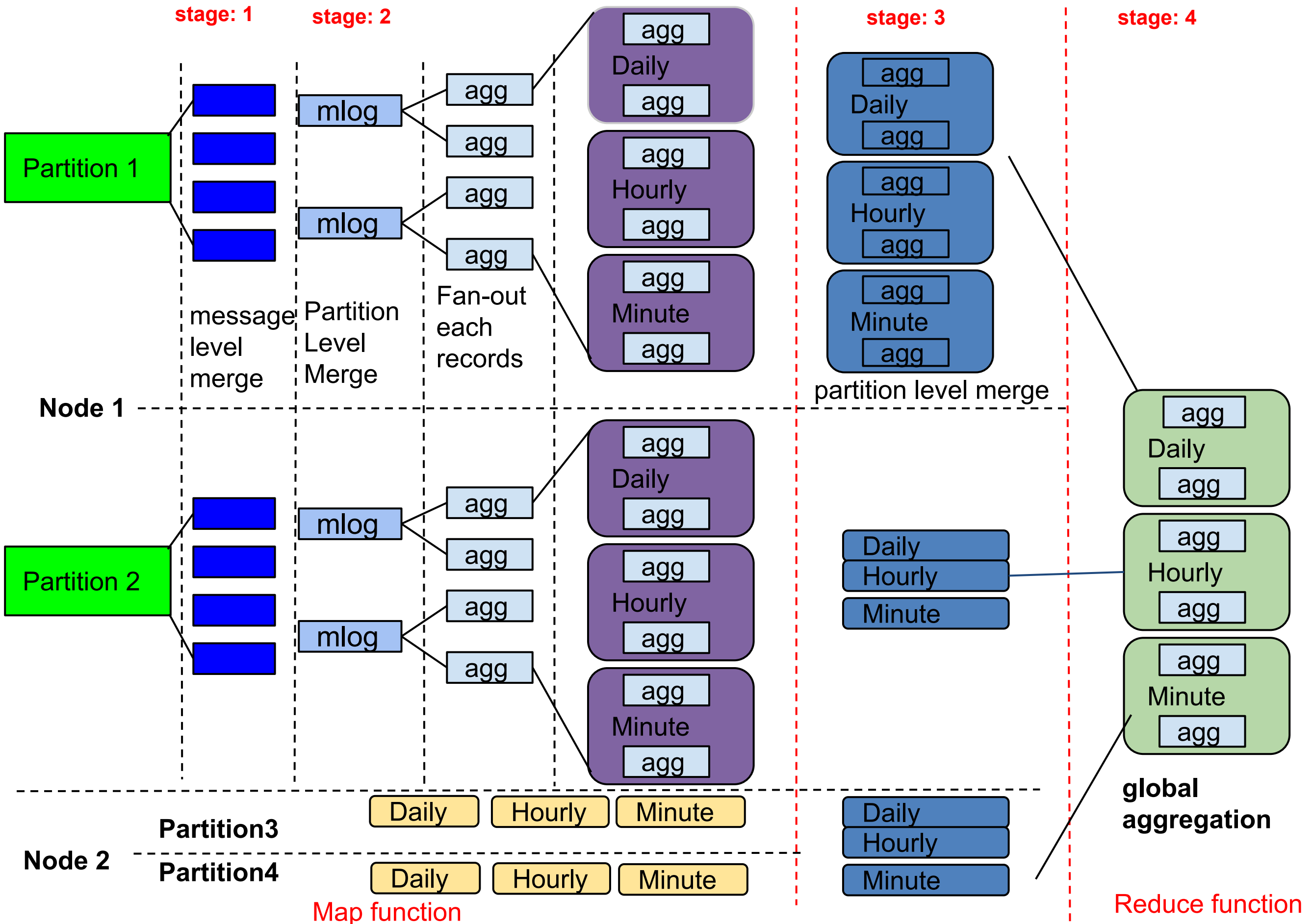
Stage 4 : cross partition aggregation : Reduce



During Job run we observed after Stage 4, number of records reduces to ~ 500K

This number tally with the write RPS at ElasticSearch..

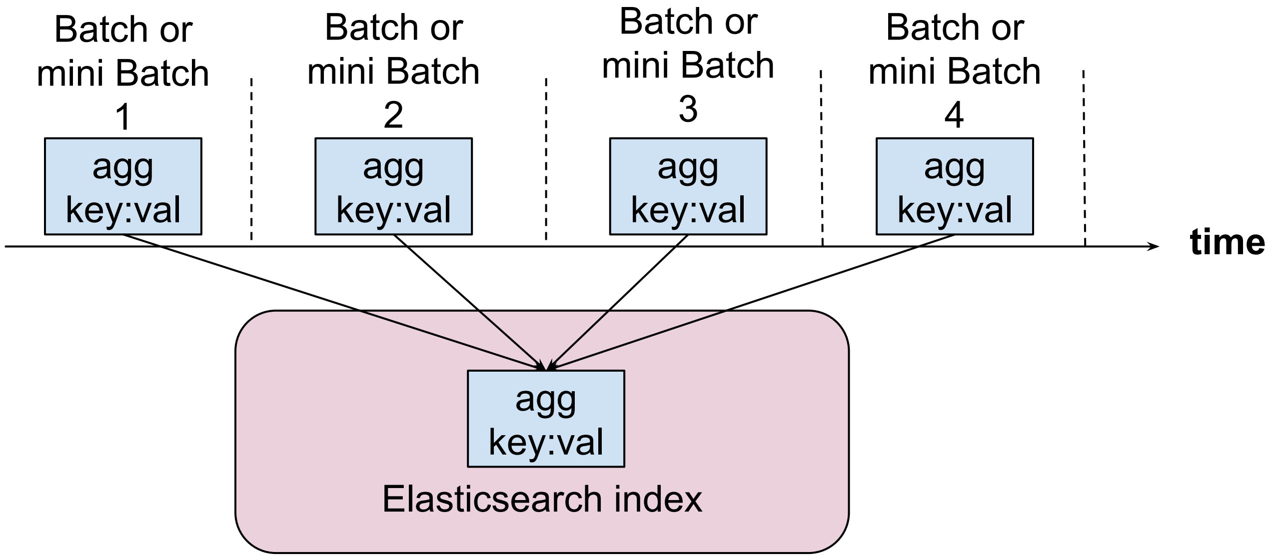
Multi-Stage Aggregation - In a Slide



instartlogic

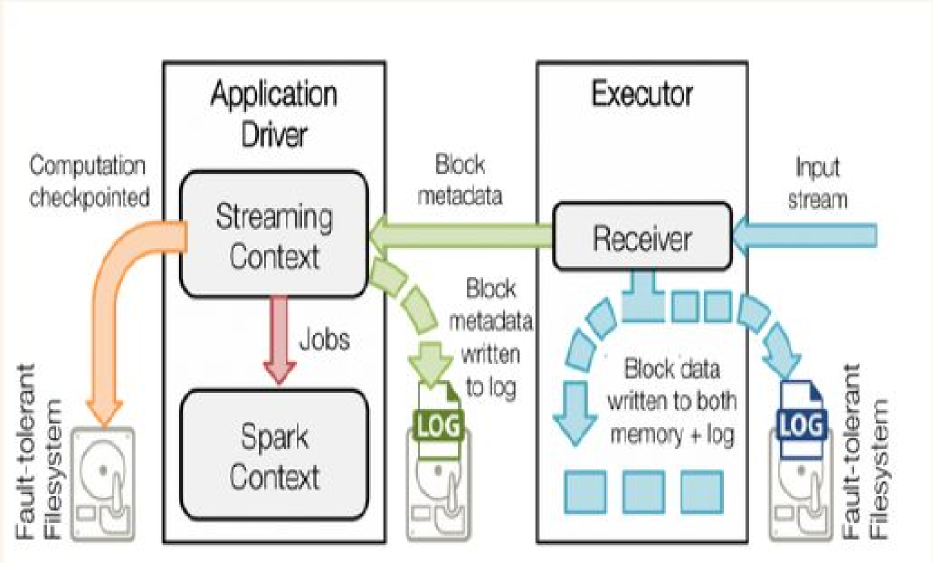
Stage 5 : Elasticsearch final Stage Aggregation

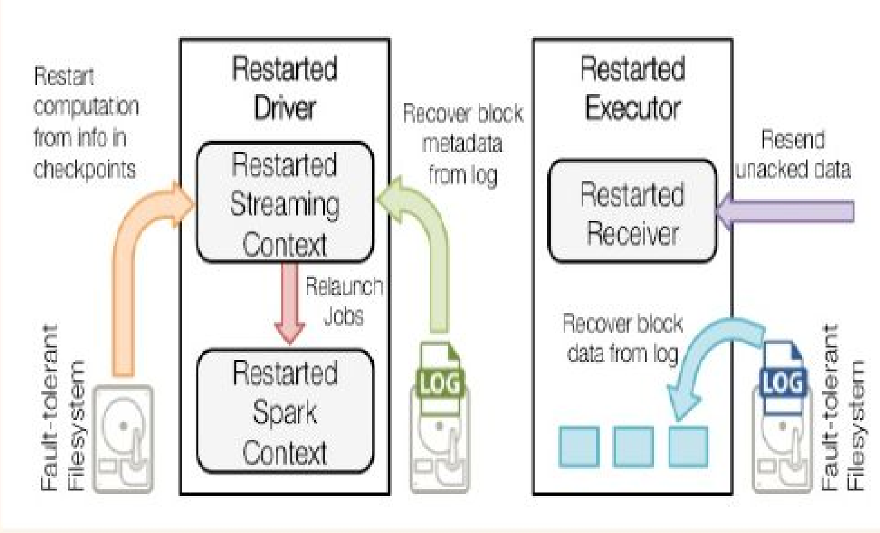
- Reason:
  - Batch Job: late arriving logs
  - Streaming Job: Each partition could have logs across multiple hours



End to End No Data Loss without WAL

Why WAL is recommended for Receiver Mode ?





How we achieved WAL Less Recovery

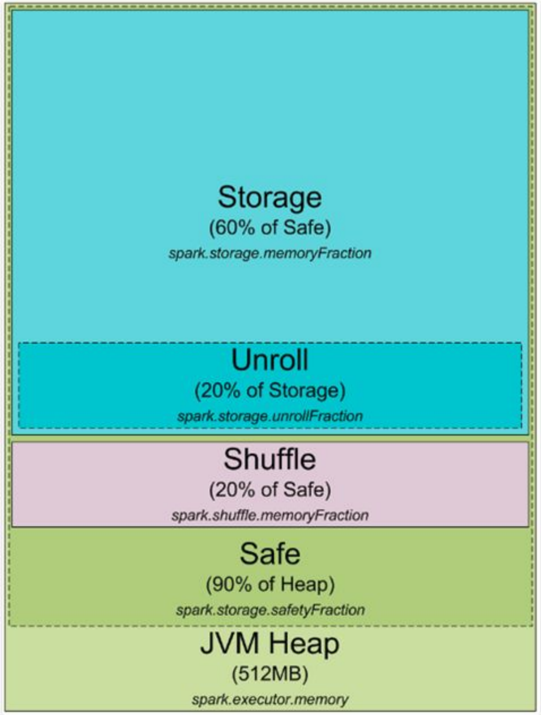
Keep Track of Consumed and Processed Offset

Every Block written by Receiver Thread belongs to one Kafka Partitions.
Every messages written has metadata related to offsets and partition

Driver reads the offset ranges for every block and find highest offset for each Partitions. Commits offset to ZK after every Batch

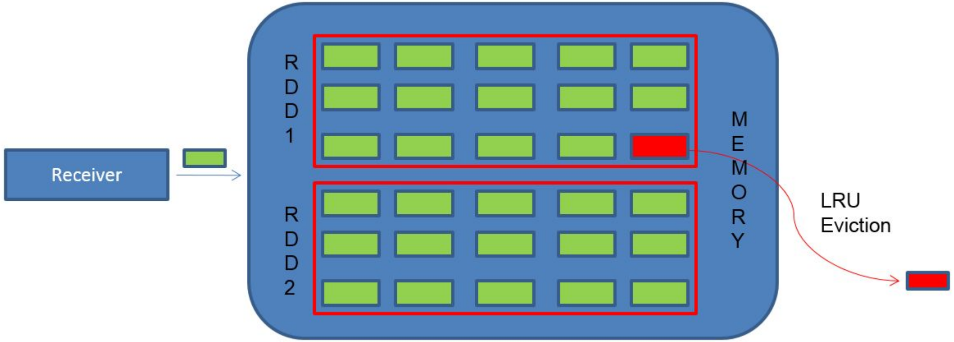
Spark Back Pressure

Spark Executors Memory : JVM Which Executes Task

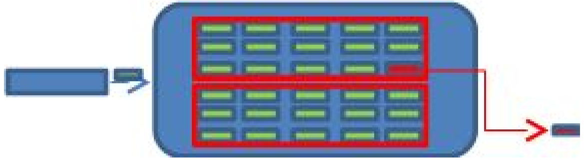


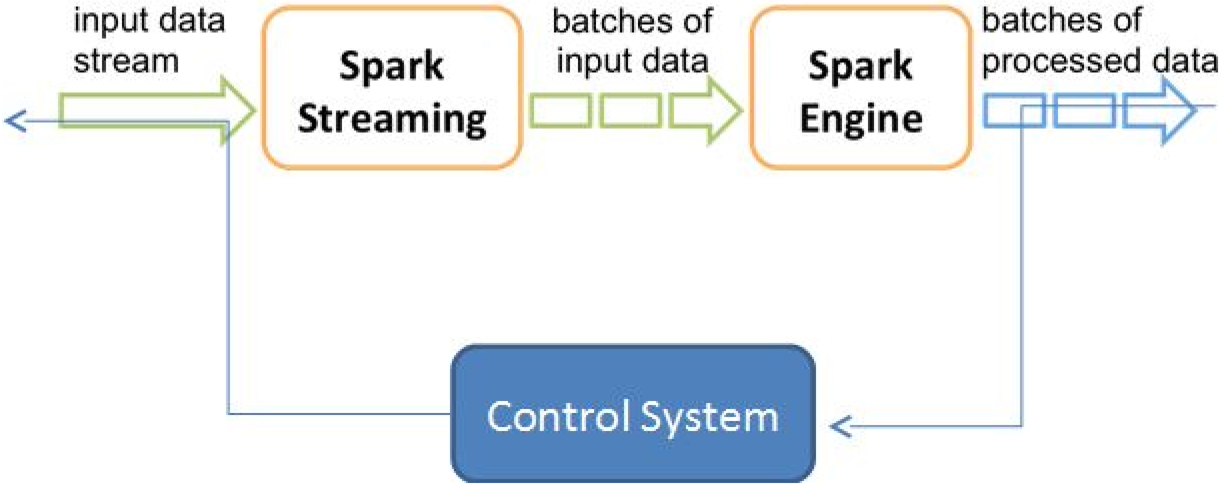
Storage Memory : Used for Incoming Blocks

Batch Time	Input Size	Status
2016/09/20 10:35:00	28544 records	queued
2016/09/20 10:30:00	29039 records	processing



Control System





It is a feedback loop from Spark Engine to Ingestion Logic

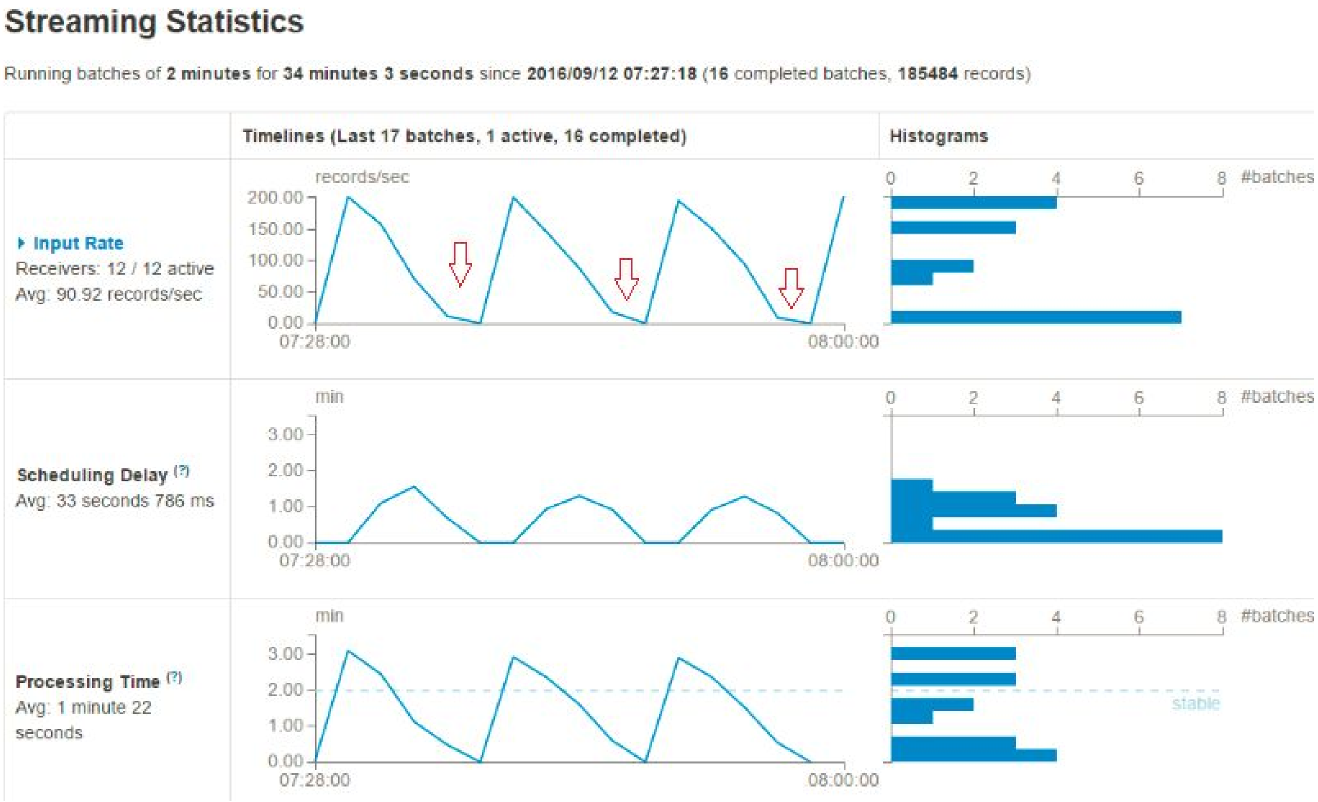
PID Controller

$$\underbrace{u(t)} = \underbrace{K_p e(t)} + \underbrace{K_i \int_0^t e(t)\,dt} + \underbrace{K_d \frac{de(t)}{dt}}$$

Output = Proportional + Integral + Derivative

Output = Error Now + Errors Past + Error Future

Input Rate throttled as Scheduling Delay and Processing Delay increases



Thank You