BUILDING A DISTRIBUTED REAL-TIME STREAM PROCESSING SYSTEM

PLUMBING THE DATA TUBES

@amylouboyle
This document and the information herein (including any information that may be incorporated by reference) is provided for informational purposes only and should not be construed as an offer, commitment, promise or obligation on behalf of New Relic, Inc. ("New Relic") to sell securities or deliver any product, material, code, functionality, or other feature. Any information provided hereby is proprietary to New Relic and may not be replicated or disclosed without New Relic’s express written permission.

Such information may contain forward-looking statements within the meaning of federal securities laws. Any statement that is not a historical fact or refers to expectations, projections, future plans, objectives, estimates, goals, or other characterizations of future events is a forward-looking statement. These forward-looking statements can often be identified as such because the context of the statement will include words such as “believes,” “anticipates,” “expects” or words of similar import.

Actual results may differ materially from those expressed in these forward-looking statements, which speak only as of the date hereof, and are subject to change at any time without notice. Existing and prospective investors, customers and other third parties transacting business with New Relic are cautioned not to place undue reliance on this forward-looking information. The achievement or success of the matters covered by such forward-looking statements are based on New Relic’s current assumptions, expectations, and beliefs and are subject to substantial risks, uncertainties, assumptions, and changes in circumstances that may cause the actual results, performance, or achievements to differ materially from those expressed or implied in any forward-looking statement. Further information on factors that could affect such forward-looking statements is included in the filings we make with the SEC from time to time. Copies of these documents may be obtained by visiting New Relic’s Investor Relations website at ir.newrelic.com or the SEC’s website at www.sec.gov.

New Relic assumes no obligation and does not intend to update these forward-looking statements, except as required by law. New Relic makes no warranties, expressed or implied, in this document or otherwise, with respect to the information provided.
WE HAVE LOTS ‘O DATA
Why Stream Processing?
Why Stream Processing?

New Relic Agents → Collector → Database → Query poller

New Relic Agents → Collector → DB writer → Query processor
What is stream processing?

- Un-bounded data
- Processed continuously
Batch

Micro-batch

Continuous
Distributed Stream processing?

Stream Processing

+ Big Data
Every minute New Relic handles:
40M+ HTTP requests
1.8B+ new data points
1000T+ events queried
Example Pipeline

Data sources → Filter/Match → Aggregate
Querying Event Data

NRQL

```
@boyle account
SELECT * FROM Transaction WHERE appName='Hovercraft Full of Eels'
```
Querying Event Data

Events Pipeline:
```
SELECT max(systemCpuPercent) FROM NodeStatus TIMESERIES WHERE appName='my-app'
```
Example Pipeline

SELECT max(systemCpuPercent) FROM NodeStatus WHERE appName='my-app'
Example Pipeline

```sql
SELECT max(systemCpuPercent) FROM NodeStatus WHERE appName='my-app'
```
Filter/map events

```
SELECT max(systemCpuPercent) FROM NodeStatus WHERE appName='my-app'
```
Aggregate events

```sql
SELECT max(systemCpuPercent) FROM NodeStatus WHERE appName='my-app'
```
Windowed data

Events Pipeline: `SELECT max(systemCpuPercent) FROM NodeStatus TIMESERIES WHERE appName='my-app'`
Time Driven Processing

- Used for Aggregation
- Output message determined by time
- Event time vs processing time
Windowing
Fixed windows
Sliding windows
Session Windows

Event Time
Fixed windows
Windowing

Stream

0:00 - 0:01

0:01 - 0:02

0:02 - 0:03
Windowing

Stream

0:00 - 0:01

0:01 - 0:02

0:02 - 0:03

@amylouboyle
Windowing

Stream

0:00 - 0:01

0:01 - 0:02

0:02 - 0:03
When to publish windows

```
SELECT max(systemCpuPercent) FROM NodeStatus WHERE appName='my-app'
SINCE 3 minutes ago UNTIL 2 minute ago
```
Late data

- Publish once
  - Drop any late events
- Publish updates
  - Replaces previous result
  - Keep already published aggregates in memory
- Publish deltas
  - Publish new result as a delta
Scale

Fundamentals

Operations

@amylouboyle
PARTITIONING DATA : WORKLOAD DISTRIBUTION
Partitioning data

Data sources → Filter/Match → Aggregate
Partitioning data

Data sources → Filter/Match
Partitioning data

Data sources → Filter/Match

@amylouboyle
Partitioning data

Data sources

Filter/Match

Filter/Match

Filter/Match
Partitioning data

Data sources → Filter/Match → Filter/Match → Filter/Match
SCALING A STREAMING SYSTEM IS ALL ABOUT PARTITIONING WORKLOAD EFFECTIVELY
Random partitioning

- Alternatively, Round robin
- most even spread of load
- great for stateless operations
Random partitioning

- Partition 0
- Partition 1
- Partition 2
Random partitioning

Data sources -> Filter/Match -> Aggregate
Partition by aggregated attribute

- If you need all data to end up in the same aggregate
- Can be very unbalanced
Partition by aggregated attribute

Partition 0

Partition 1

Partition 2
Partition by aggregated attribute
Partition by resource

- DB shards
- minimize write locations
Partition by resource

Partition 0

Partition 1

Partition 2
Partition by resource

Partition 0

Partition 1

Partition 2
DATA SERIALIZATION
Debatch/pre-filter → Filter/Match → Aggregate
Measuring the gains

Events Pipeline: SELECT sum(jvmCpuPercent * cpuCount) as 'CPUs Used' FROM NodeStatus TIMESERIES COMPARE WITH 3 weeks ago

CPU before and after
Since 60 minutes ago, compared with 3 weeks earlier

@amylouboyle
SERIALIZATION IS EXPENSIVE
BE LAZY
Backpressure

Stream application

consumer buffer

stream logic

producer buffer

durable + replayable

durable + replayable
@amylouboyle

Fundamentals

Scale

Operations
FAILURE RECOVERY AND DEPLEYS
THINGS WILL GO WRONG, BE PREPARED
Redundancy

• Provisioning headroom
Redundancy

Stream service

Stream service

Stream service

Stream service
Redundancy

Stream service

Stream service

Stream service

Operations
Partition rebalances

- Random
- Sticky
- Fixed
Random assignment
PARTITION REBALANCES

Random assignment

Data source

Filter/Match

Operations
PARTITION REBALANCES

Random assignment

Data source

Filter/Match

Filter/Match
PARTITION REBALANCES

Sticky assignment

Data source

Filter/Match
Filter/Match
Filter/Match

@amylouboyle
PARTITION REBALANCES

Sticky assignment

Data source

Filter/Match

Filter/Match
Fixed assignment

PARTITION REBALANCES

Data source

Filter/Match

Filter/Match

Filter/Match

@amylouboyle
Fixed assignment

PARTITION REBALANCES

Operations
PARTITION REBALANCES

Fixed assignment

Operations

@amylouboyle
Stateless vs Stateful

STATELESS SERVICES
• Event-driven
• Random partition assignment
• Random rebalances

STATEFUL SERVICES
• Aggregation
• partition by attribute of data
• Sticky/Fixed rebalances
Snapshotting state

Data sources → Filter/Match → Aggregate

@amylouboyle
Snapshotting state
Snapshotting state

Data sources → Filter/Match → Aggregate

@amylouboyle
Snapshotting state

Filter/Match

input topic

Aggregate

Aggregate

Aggregate

Operations
Snapshotting state

Filter/Match

Aggregate

input topic

snapshot topic

Operations
Snapshotting state

Filter/Match

input topic

Aggregate

Aggregate

Aggregate

snapshot topic

0

1

2

@amylouboyle
Snapshotting state

Filter/Match

input topic

Aggregate

Aggregate

Aggregate

snapshot topic

@amylouboyle
Snapshotting state

Operations

Filter/Match

input topic

Aggregate

Aggregate

Aggregate

snapshot topic

@amylouboyle
MONITORING :
WHAT TO WATCH
You have a problem. You: Ah! I know! I use a distributed system. Now you still have a problem, but you no longer know where.
Correlating state across the cluster

- How do you know all data is being consumed?
- How do you know you’re not double consuming data?
MONITOR DATA FLOW THROUGH THE SYSTEM
Tracking data flow

This shouldn't happen

Fix deployed
Tracking data flow

System Throughput metrics

Batches Consumed
Since 60 minutes ago
- 30 M
- 20 M
- 10 M

Pre-filtered events
Since 60 minutes ago
- 500 M
- 400 M
- 300 M
- 200 M
- 100 M

Filtered messages produced
Since 60 minutes ago
- 80 M
- 60 M
- 40 M
- 20 M

Aggregator messages consumed
Since 60 minutes ago
- 80 M
- 60 M
- 40 M
- 20 M

Aggregator Late Events
Since 60 minutes ago
- 6 M
- 4 M
- 2 M

Aggregator messages produced
Since 60 minutes ago
- 150 k
- 100 k
- 50 k
Tracking data flow
Tracking data flow
Data Lag
Data Lag

Since 10 hours ago until 9 hours ago

Data lag

Recovery

NewRelic Administration: `SELECT max(offsetsLog) FROM ConsumerLag WHERE topic='important-service' FACET partition`
BUILD YOUR MONITORING AS YOU BUILD YOUR SYSTEM
Must-have dashboards

- Data lag
- Data ownership
- System throughput
- SLIs
RUN YOUR CODE WITH REAL DATA IN A TEST ENVIRONMENT BEFORE DEPLOYING TO PRODUCTION
FINAL THOUGHTS
USING A STREAM PROCESSING FRAMEWORK WILL PROBABLY MAKE YOUR LIFE EASIER
IF YOU REMEMBER NOTHING ELSE...

- Windowing for unordered and late data
- Partition data effectively
- Serialization is expensive
- Prepare for failure
- Keep track of data flow and bottlenecks