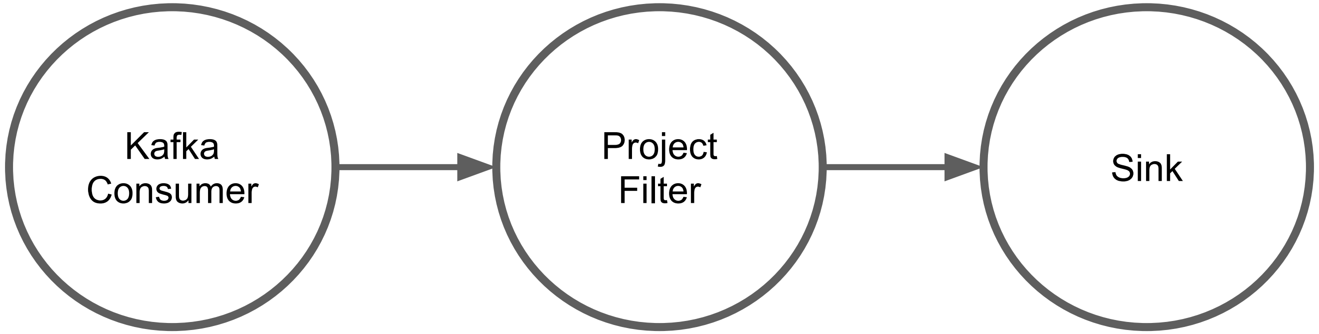
Autoscaling

Timothy Farkas
Senior Software Engineer @ Netflix

Problem Definition

Our Pain



- Thousands of stateless single source and single sink Flink routers.
- All operators are chained.
- When lag for a router exceeds a threshold we are paged.

Definitions

- **Workload:** Events being produced to a kafka topic. Two main knobs to turn:
  - Message Rate
  - Message Size
- **Lag:** The time it would take for a router to process all the remaining unprocessed events that are buffered in its kafka topic.
- **Healthy Router:** A router is healthy if it’s **lag** is always under ~5 minutes.
- **Autoscaling Solution:** Adjust the number of nodes in the router dynamically based on the **workload** to keep the router **healthy**. Attempt to use the smallest number of nodes that are required to keep the pipeline **healthy**.

Solution Space

- **Claim:** There is no perfect solution. Any autoscaling algorithm can be defeated by one or more workloads.
- **Proof:** Take any autoscaling algorithm **A**. Provide **A** with a workload **W** that does the exact opposite of what **A** expects whenever **A** decides to resize the cluster. **=>**
  A will always make the wrong decision for **W** by definition. **=>**
  Q.E.D.

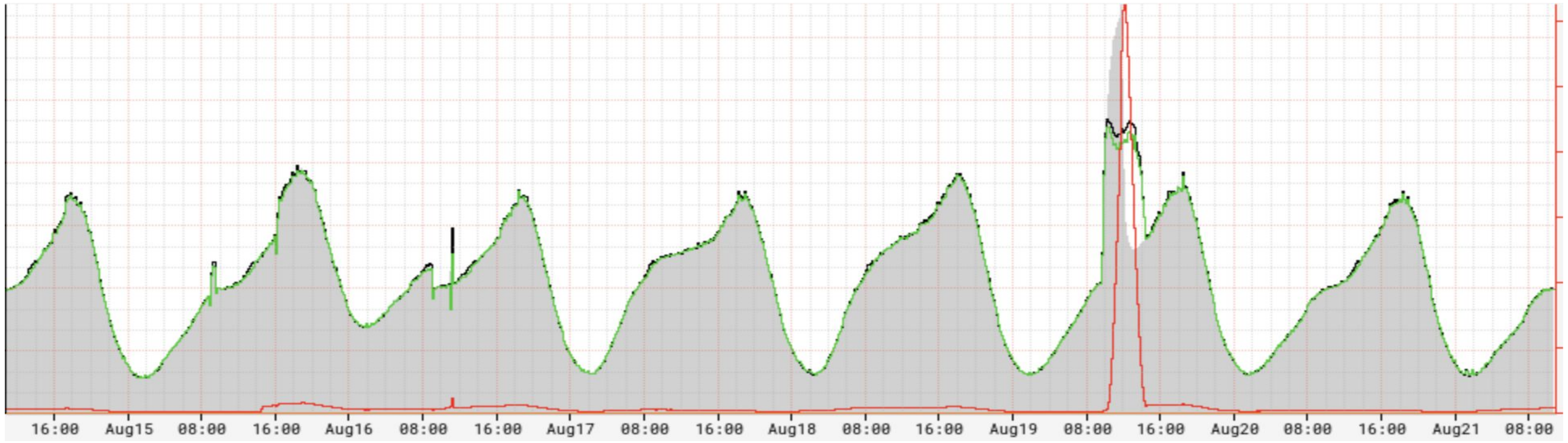
    - Understand our limitations.
    - Make assumptions about our workloads.
    - Make a solution that works well when taking both into account.

Limitations

- Rescaling introduces processing pauses.
- Scaling down a Flink job suspends processing for 1 - 3 minutes and possibly more.
  - **CHEAP:** Graceful shutdown with savepoint.
  - **EXPENSIVE:** Remove TMs.
  - **CHEAP:** Restart from savepoint with reduced parallelism.
- Scaling up a Flink job suspends processing for period < 1 minute.
  - **EXPENSIVE:** Add TMs.
  - **CHEAP:** Graceful shutdown with savepoint.
  - **CHEAP:** Restart from savepoint with increased parallelism.
- There is a two minute delay for propagating metrics through Netflix's metrics infrastructure.

Assumptions

- Better to accidentally over allocate than to under allocate.
- Average message size changes infrequently.
- Large spikes in the workload happen, but not frequently.
- Workloads tend to smoothly increase or decrease, when they don’t have a large spike.



Solution

Desirable Characteristics

- Minimal amount of state
- Deterministic behavior
- Easy to unit test
- Easy to control

Approaches

- Historical Prediction
- Rule Based
- PID Controller
- **Statistical Short Term Prediction + Policies**

Autoscaling - High Level Steps

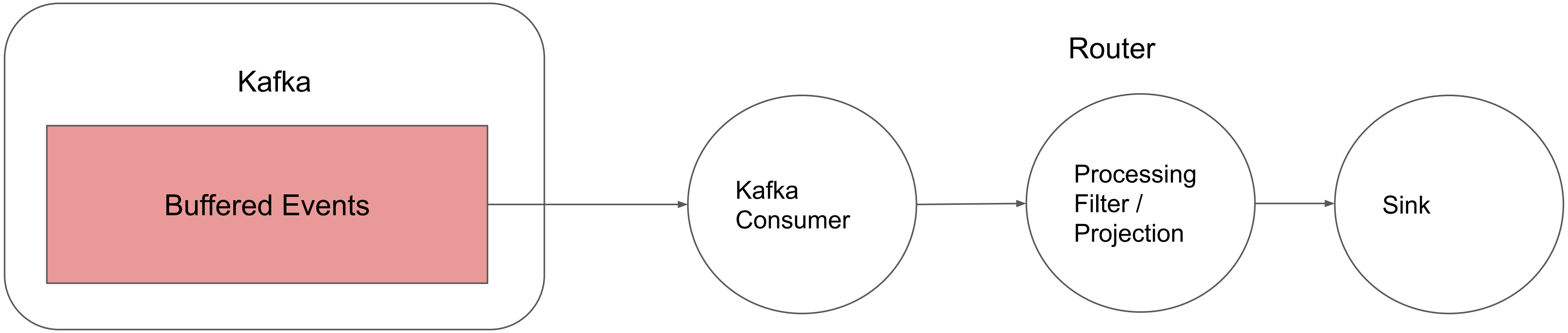
- **Collect:** Receive a batch of metrics for the current 1 minute timebucket.
- **Pre-Decision Policy:** Apply policies which decide whether the cluster can be rescaled or whether performance information can be collected about the cluster.
- **Decide:** Based on latest batch of events, decide whether to:
  - Scale up
  - Scale down
  - Stay the same
  - Also collects cluster performance information
- **Calculate Size:** If scaling up or down, decide how many nodes need to be added or removed.
- **Post-Decision Policy:** Apply policies which can modify scale up and scale down decisions.

Metrics Collection

Each minute collect the following

- Kafka consumer lag
- Records processed per second
- Cpu utilization
- Max Message Latency
- Kafka messages in per second
- Net in / out utilization
- Sink health metrics

Store the metrics for the past **N** minutes to inform scaling decisions and to do regression to predict the workload.



Pre-Decision Policy

Abort autoscaling process if:

- The router has recent task failures
- The router is currently redeploying

Decide - Scale Up

Scale up if:

- There is significant lag AND sink is healthy
- Utilization exceeds the safe threshold AND sink is healthy

Key Insight - Collect cluster performance information:

- If the cluster needs to be scaled up that means the cluster is saturated.
- This is effectively a benchmark for the performance of the cluster at the current size.
- Save this information in a **Performance Table** for future scaling decisions.
- More on this in the **Performance Table** section later.

Decide - Scale Down

Scale down if:

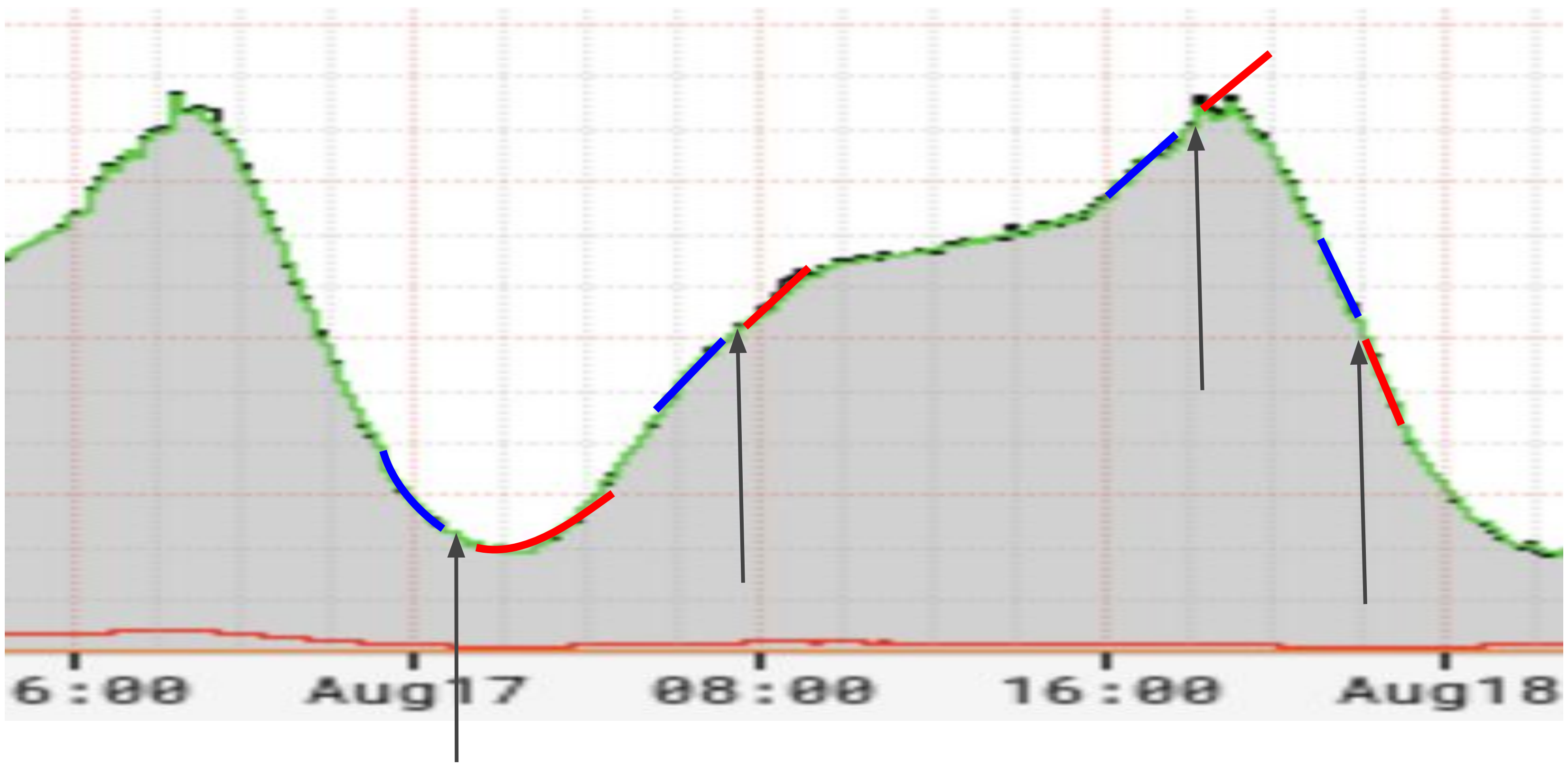
- There is no lag AND we do not anticipate an increase in incoming messages
- More on how we anticipate incoming message rate in the **Predict Workload** section later.

Calculate Size

- **Predict Workload:** Predict the future workload (messages in per second), while taking spikes into account.
- **Target Events Per Second:** Compute target events / sec that the pipeline will need to handle **X** minutes from now.
- **Cluster Size Lookup:** Use the target events / sec to estimate the desired cluster size, which can handle the workload up to **X** minutes from now.

Predict Workload

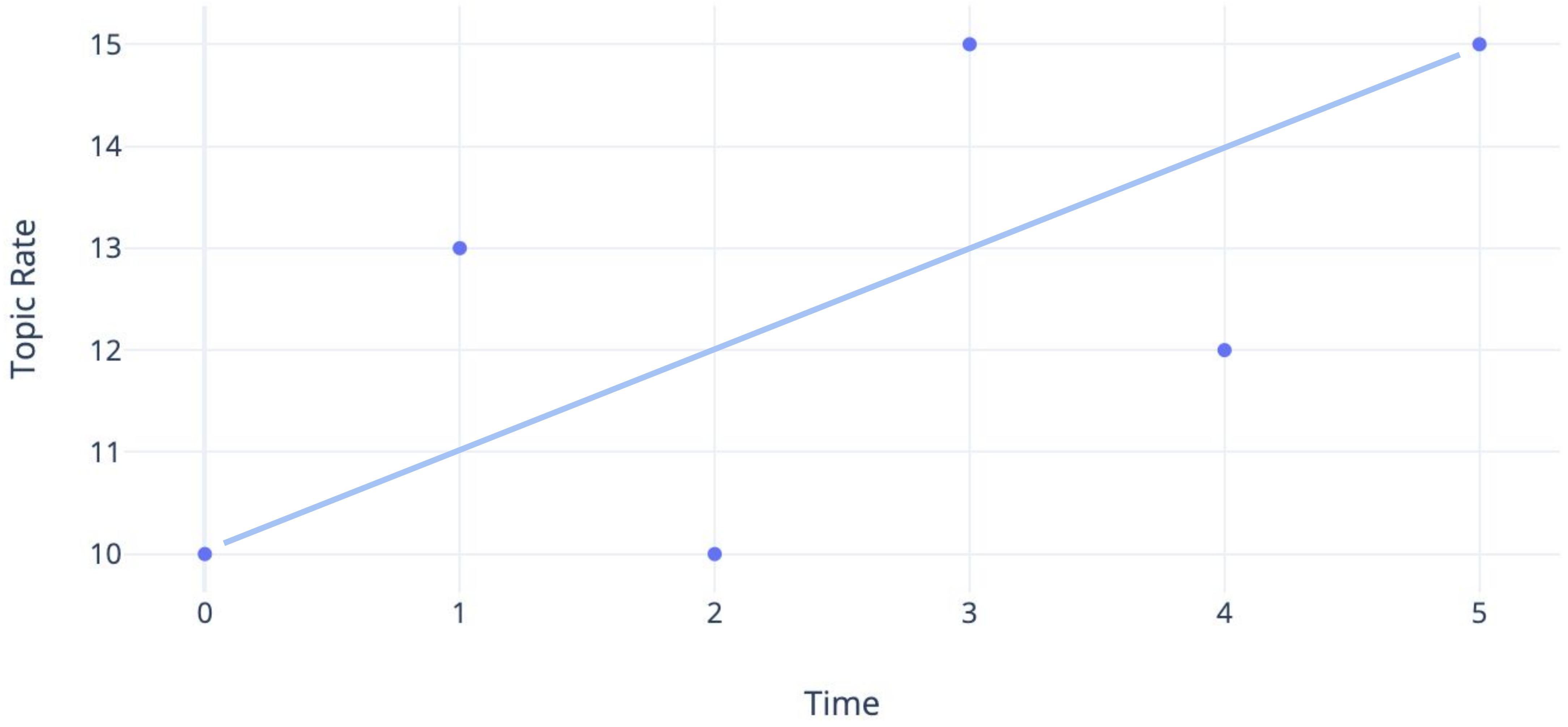
- Quadratic regression for troughs: **ax^2 + bx + c**
- Linear regression for everything else: **ax + b**



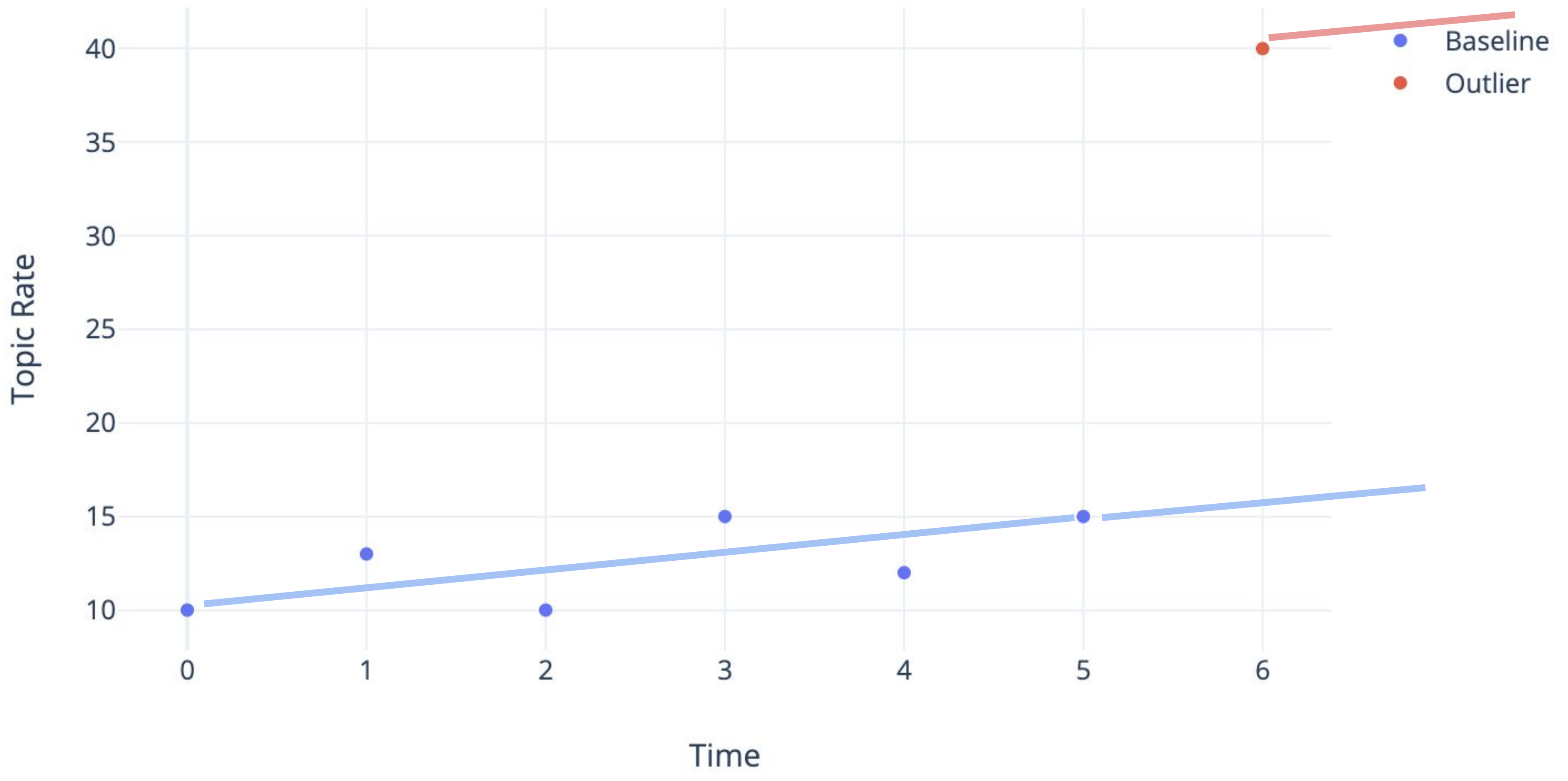
Spike Detection

- Assume error for regression is normally distributed and centered at 0.
- Find standard deviation of error.
- Any error greater than 3 * sigma is an outlier.
- After enough consecutive outliers are observed, the baseline is reset.

Spike Detection - Baseline



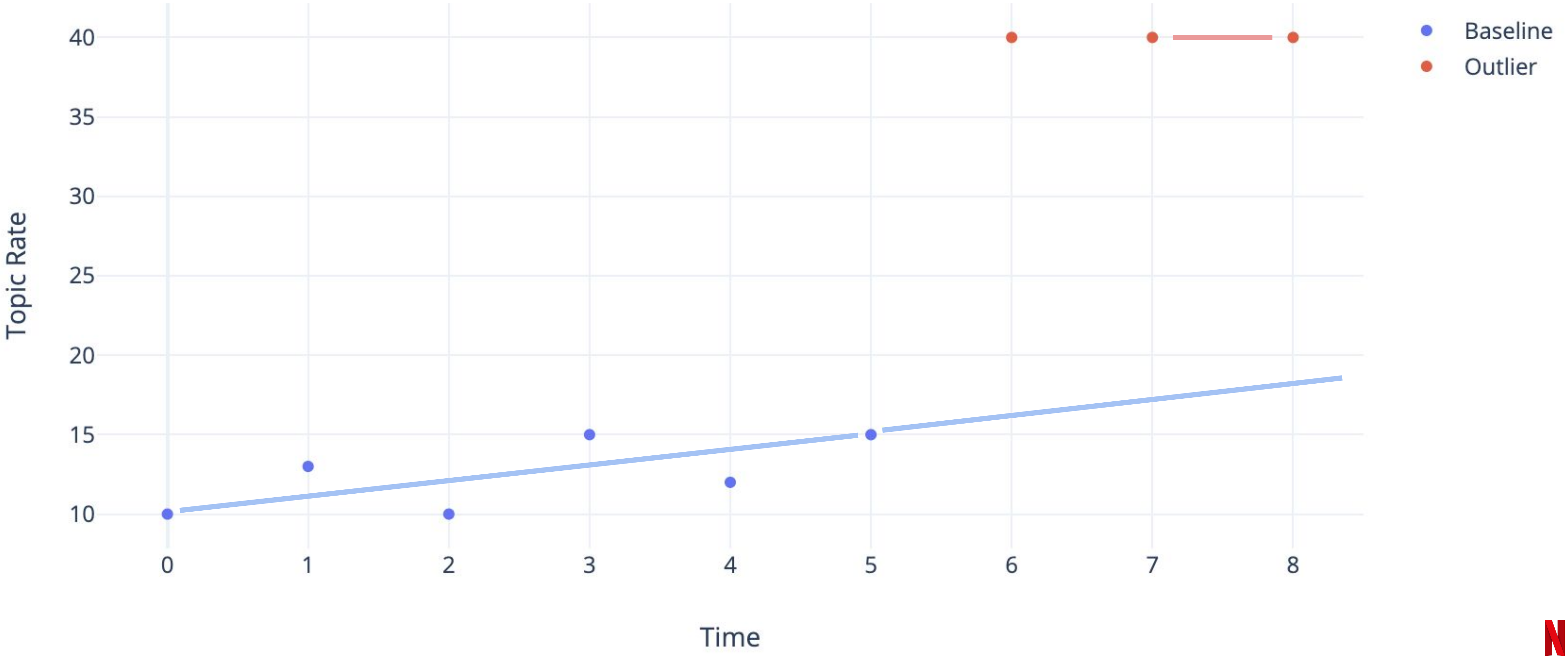
Spike Detection - First Outlier



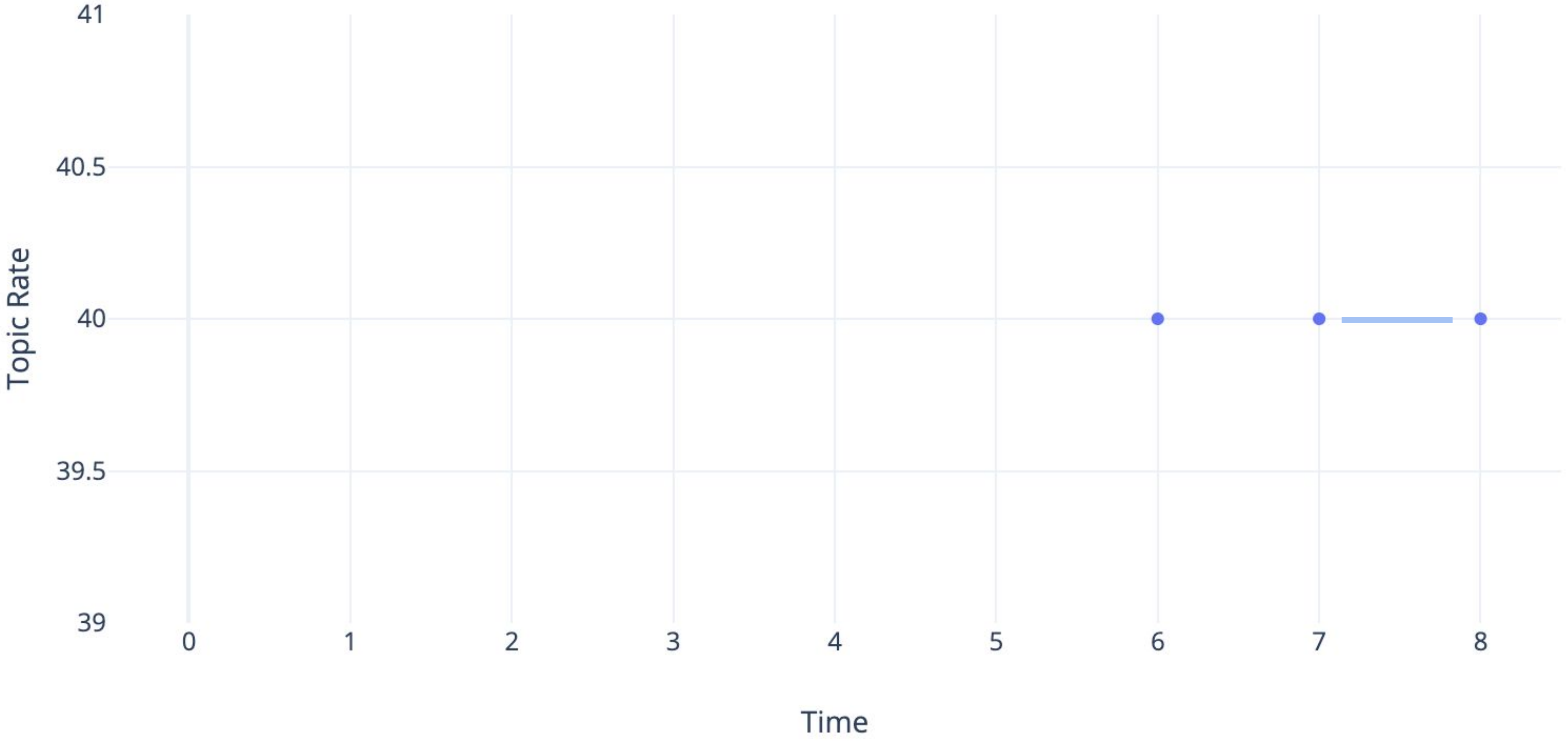
$$\mathrm{Var}(X) = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2,$$

Var = (1 + 1 + 1 + 1) / 6 std = sqrt(4 / 6) 3 *std = 2.45

Spike Detection - Outliers



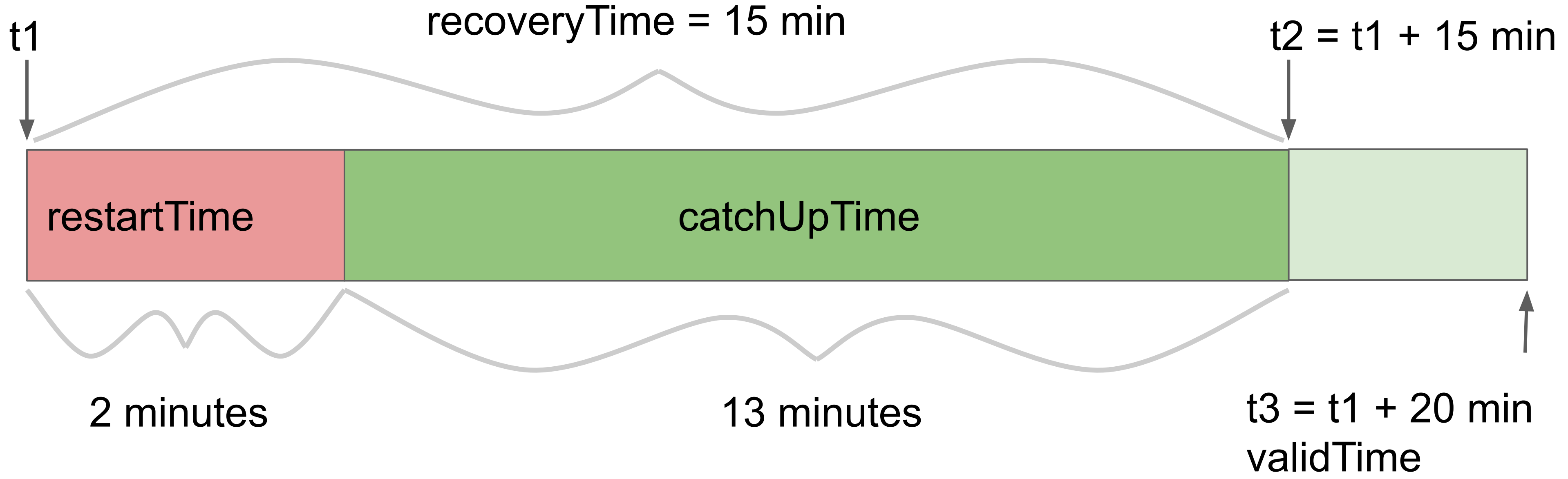
Spike Detection - Baseline Reset



Calculate Size

- **Predict Workload:** Predict the future workload (messages in per second), while taking spikes into account.
- **Target Events Per Second:** Compute target events / sec that the pipeline will need to handle **X** minutes from now.
- **Cluster Size Lookup:** Use the target events / sec to estimate the desired cluster size, which can handle the workload up to **X** minutes from now.

Compute Target Processing Rate



3. $$targetRate = max(recoveryRate, workloadRate)$$

2. $$recoveryRate = \frac{totalEvents}{catchUpTime} \qquad workloadRate = r(t_3)$$

1. $$totalEvents = \int_{t_1}^{t_2} r(t)\,\mathrm{dt} + bufferedEvents$$

Calculate Size

- **Predict Workload:** Predict the future workload (messages in per second), while taking spikes into account.
- **Target Events Per Second:** Compute target events / sec that the pipeline will need to handle **X** minutes from now.
- **Cluster Size Lookup:** Use the target events / sec to estimate the desired cluster size, which can handle the workload up to **X** minutes from now.

Cluster Size Lookup - The Performance Table

- Lag and resource usage is high =>
- Pipeline is saturated =>
- We decide to scale up =>
- We know the maximum throughput of the current cluster at the current size =>
- Record the performance in a lookup table

Cluster Size Lookup - The Performance Table

- Given a target rate find the performance records above and below it.
- Do linear interpolation to find the suitable cluster size.

Performance Table

Num Nodes	**Max Rate**
4	10,000
10	20,000
18	35,000

targetRate = 15,000

ratio = $\frac{(15000 - 10000)}{(20000 - 10000)}$

ratio = .5

clusterSize = .5 (4) + .5 (10)

clusterSize = 7

Cluster Size Lookup - Corner Case

Performance Table

Num Nodes	**Max Rate**
4	10,000
10	20,000
18	35,000

targetRate = 40,000

clusterSize = $\frac{(40{,}000)}{(35{,}000 / 18)}$ = 20.57

ceil(clusterSize) = 21

Cluster Size Lookup - Complexities

- Few more corner cases
- Utilization also needs to be taken into account
- Want new cluster size to have reasonable resource utilization 60% or less

Calculate Size
Scale Up vs Scale Down

- Flow and logic is the same
- Minor differences in implementation details

Post-Decision Policy

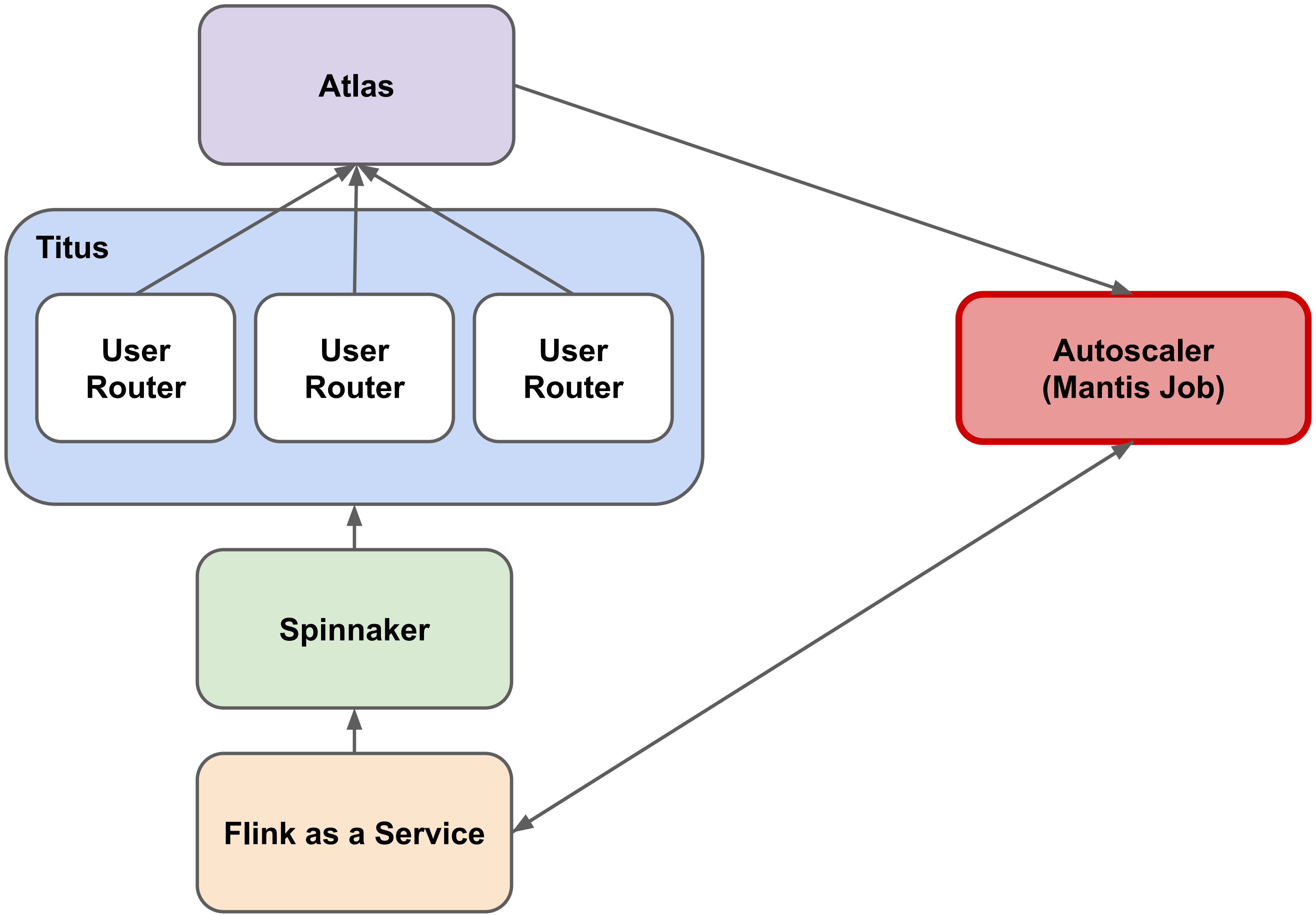
- Minimum cluster size based on partition count of Kafka topic
- Maximum cluster size based on partition count of Kafka topic
- Cooldown period for scale ups
- Cooldown period for scale downs
- Disable scale downs during region failover (see **Region Failover** section)
- Safety limit for max scale up. Ex. cannot add more than 50 nodes during a scale up
- Safety limit for max cluster size

Running In Production

Architecture Options

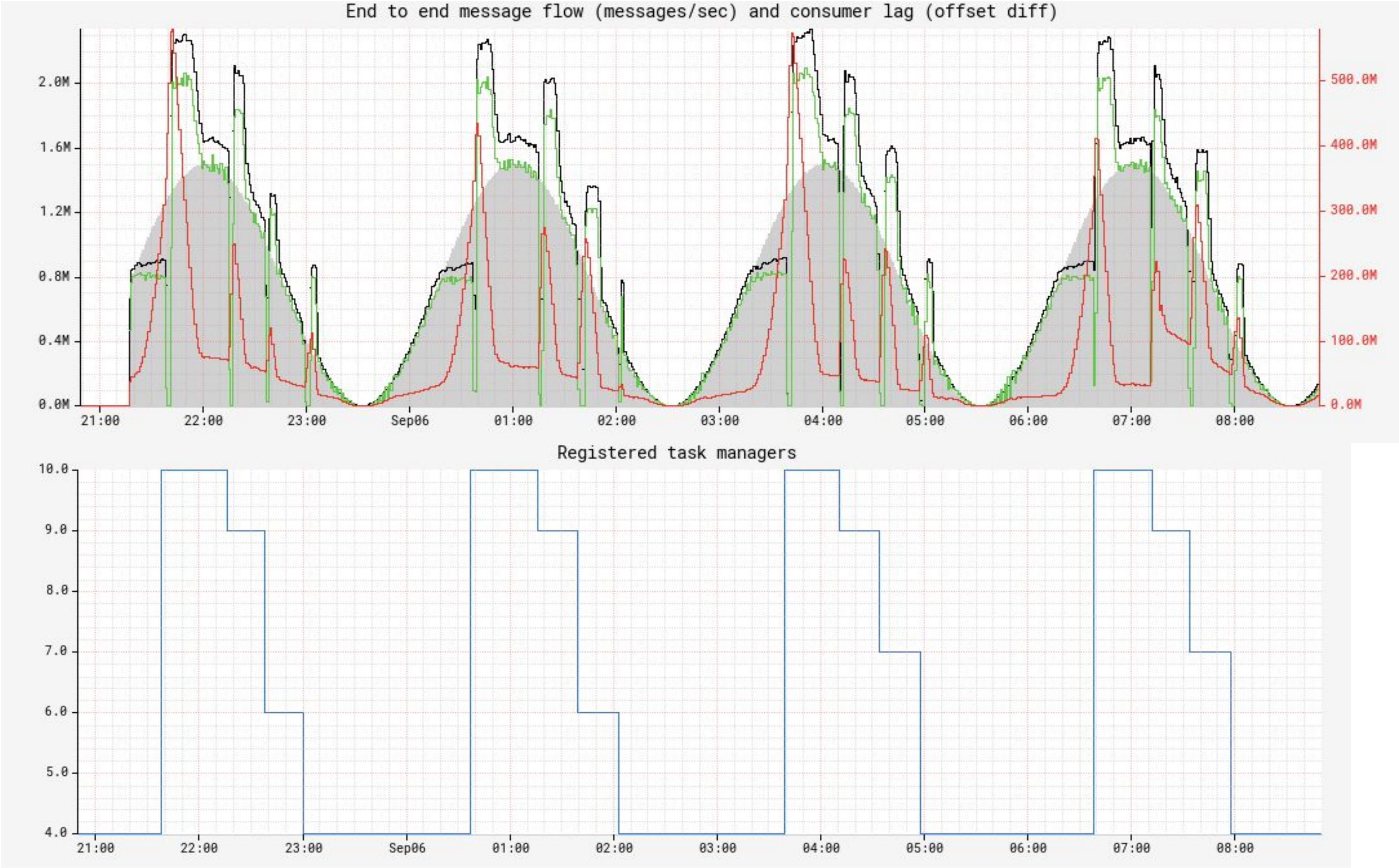
- Embed autoscaling in Flink
  - **Pros:**
    - Lower latency for retrieving metrics.
  - **Cons:**
    - Complex resource manager interactions get pushed down into Flink.
    - Rescale operation not easily integrated into operations history for the job.
    - Autoscaling changes requires redeploy of the job.
- **Run autoscaling as a Mantis pipeline**
  - **Pros:**
    - Flink service control plane handles all resource manager interactions already and it can be re-used for rescaling the job.
    - Flink service control plane keeps history of all rescale actions.
    - Autoscaling can be changed without redeploying jobs.
  - **Cons:**
    - 2 minute latency for getting metrics.

Autoscaling Architecture



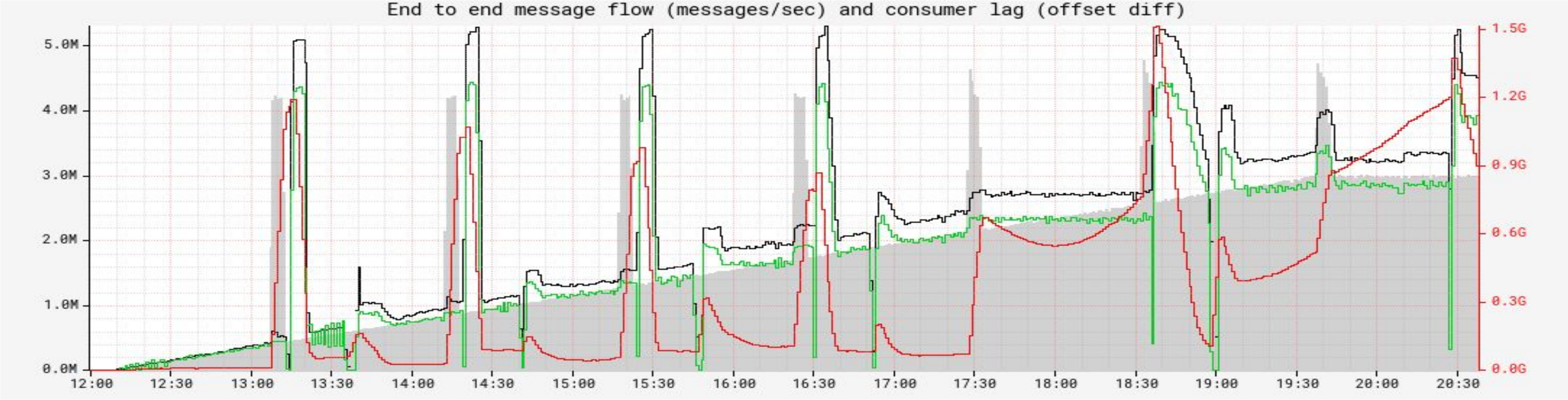
Results

Sine Wave

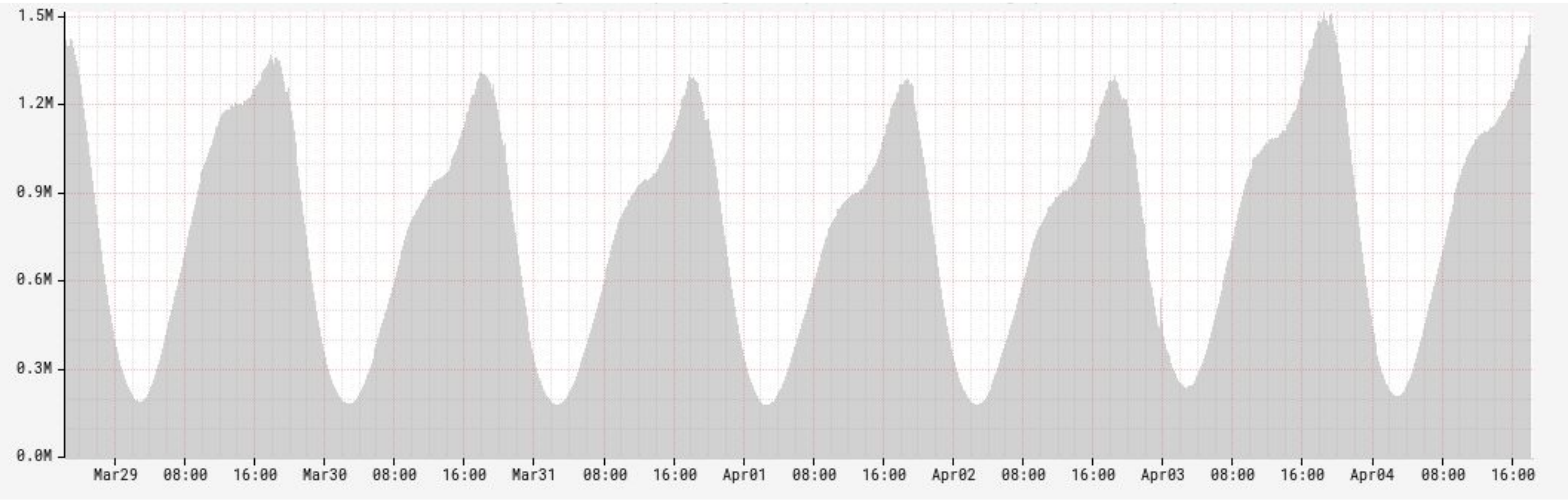


End to end message flow (messages/sec) and consumer lag (offset diff)
2.0M
1.6M
1.2M
0.8M
0.4M
0.0M
500.0M
400.0M
300.0M
200.0M
100.0M
0.0M
21:00
22:00
23:00
Sep06
01:00
02:00
03:00
04:00
05:00
06:00
07:00
08:00
Registered task managers
10.0
9.0
8.0
7.0
6.0
5.0
4.0

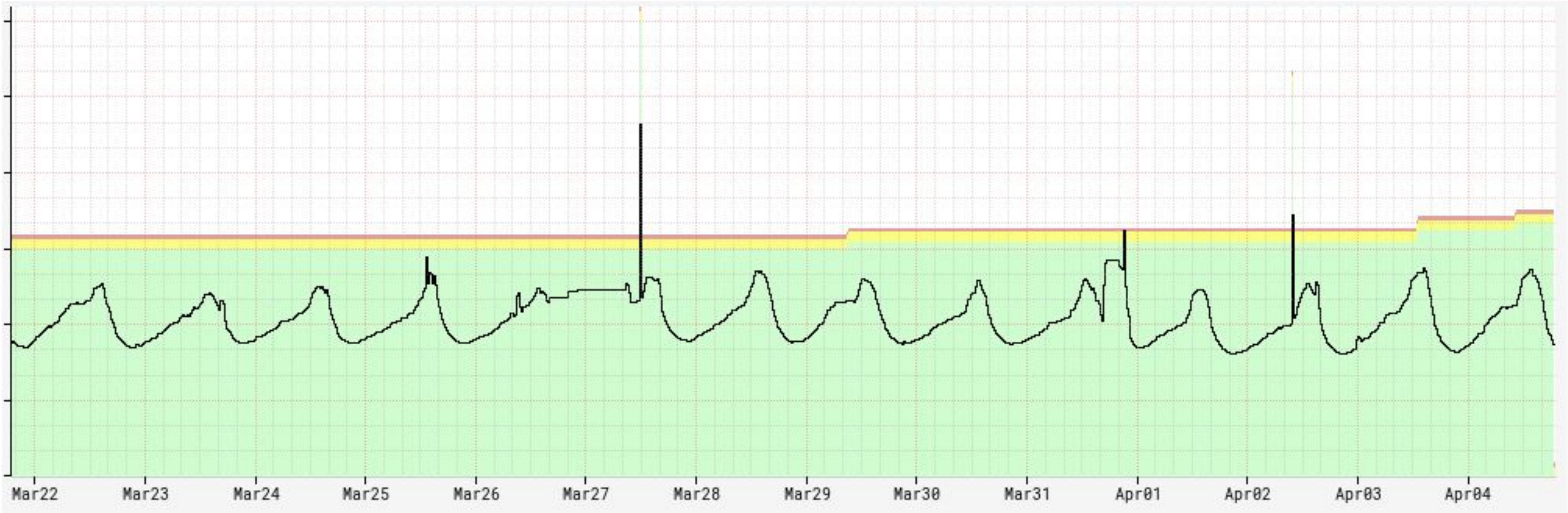
Linear Spikey



Production Router



Fleet Resource Usage



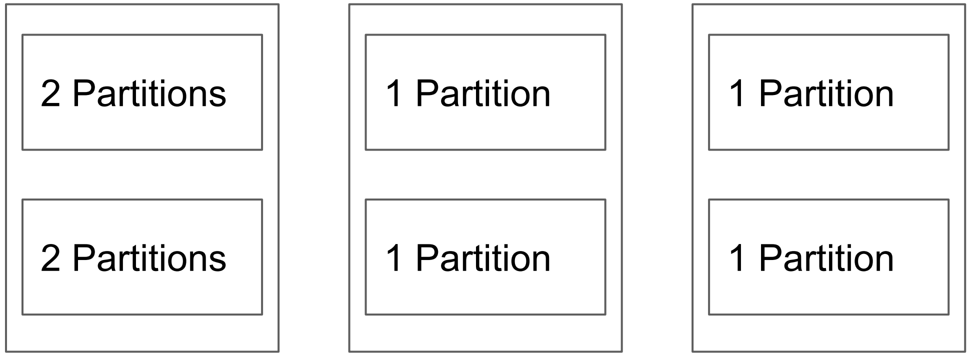
Additional Considerations

Memory Requirements

- Direct memory has to be preserved for Kafka Consumer and Kafka Producer
- Direct memory cannot be changed for TMs that are already running
- Smaller clusters require more Direct memory (each node handles more partitions)
- Larger clusters require less Direct memory (each node handles fewer partitions)
- Deploy cluster with Direct memory that works for the minimum cluster size

Partition Balancing

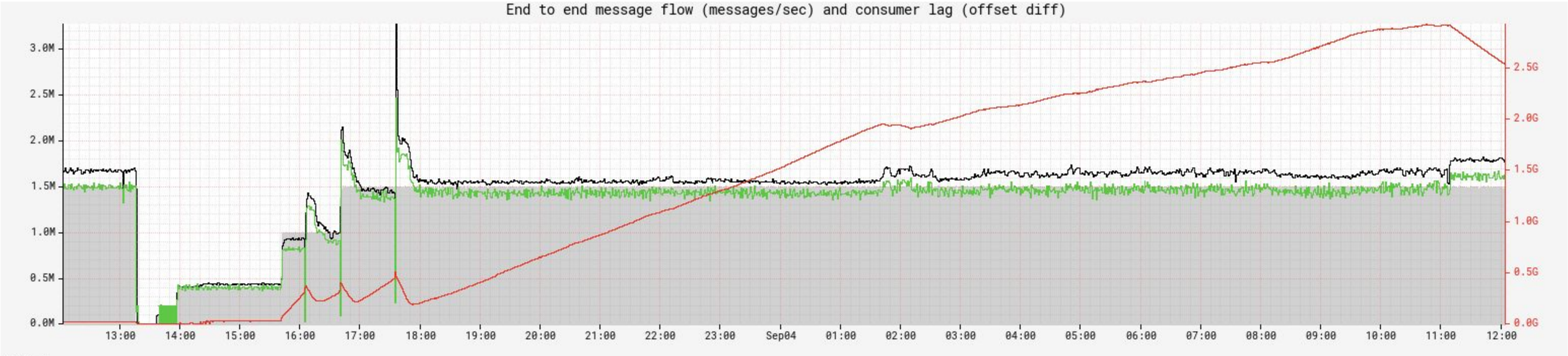
- 3 TMs
- 2 Task Slots per TM
- Topic with 8 partitions

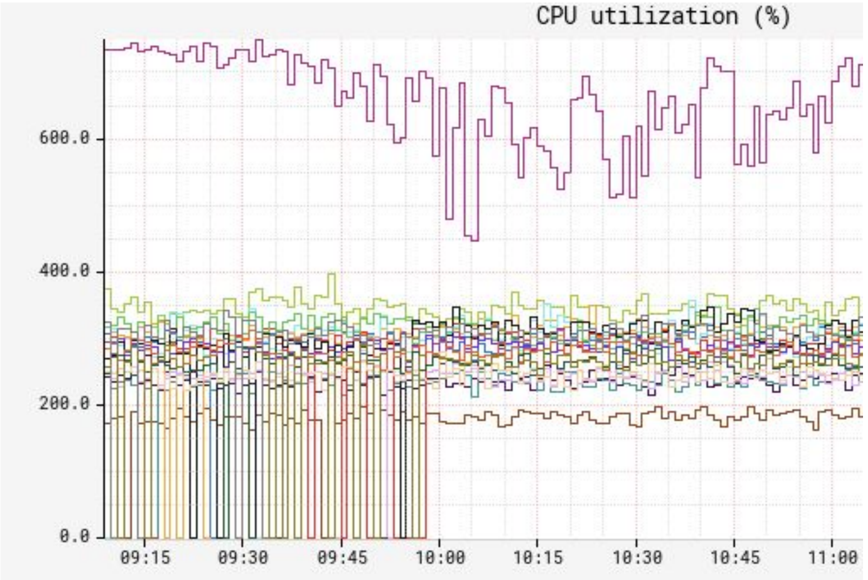


TMs with N Task Slots in the worst case can be assigned N more partitions than other TMs

- Aggregate consumer lag and average message latency may be low.
- A few partitions may have high latency due to unbalanced distribution of partitions.
- Round up the cluster so that the maximum possible partitions per subtask is reduced by 1.
- Note this is a much looser requirement on cluster size than requiring equal distribution of partitions to subtasks. This allows finer grained cluster size control.

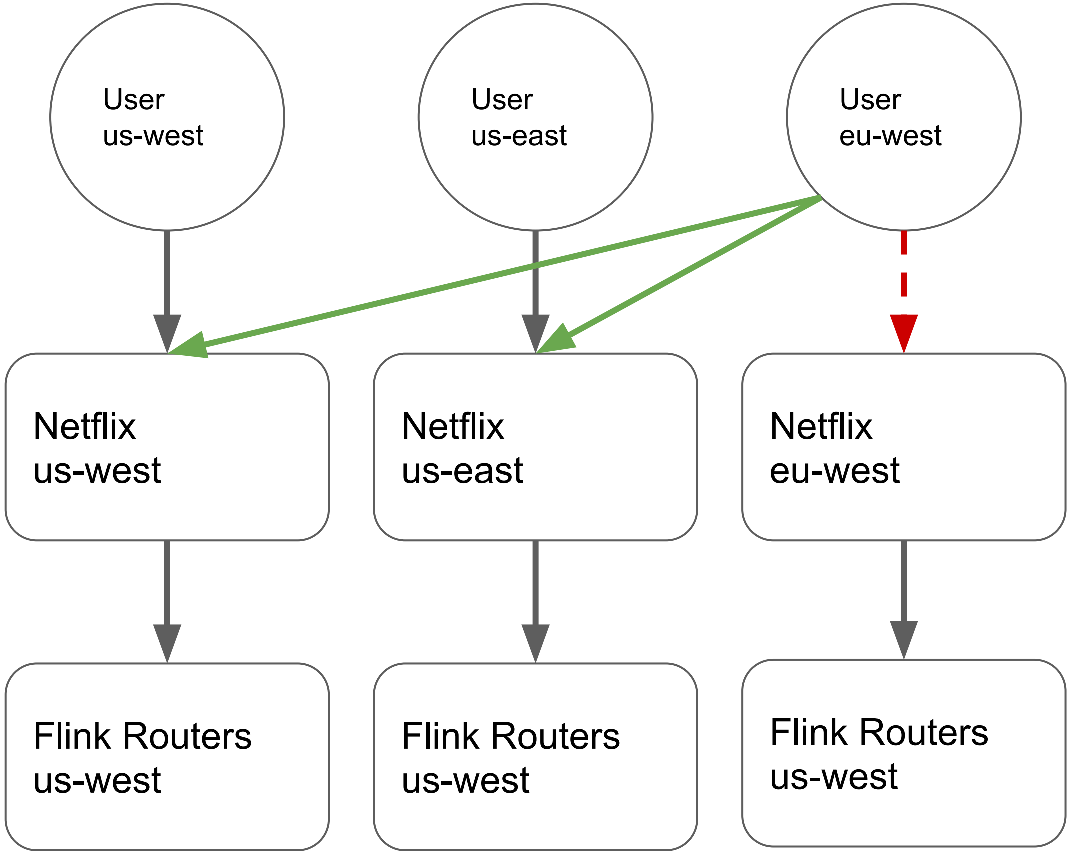
Outlier Containers






```
[Wed Sep  4 17:50:03 2019] EDAC skx MC2: HANDLING MCE MEMORY ERROR
[Wed Sep  4 17:50:03 2019] EDAC skx MC2: CPU 24: Machine Check Event: 0 Bank 13:
cc063f80000800c0
[Wed Sep  4 17:50:03 2019] EDAC skx MC2: TSC 0
[Wed Sep  4 17:50:03 2019] EDAC skx MC2: ADDR 57c24fb4c0
[Wed Sep  4 17:50:03 2019] EDAC skx MC2: MISC 908511010100086
[Wed Sep  4 17:50:03 2019] EDAC skx MC2: PROCESSOR 0:50654 TIME 1567619410 SOCKET 1 APIC 40
[Wed Sep  4 17:50:03 2019] EDAC MC2: 6398 CE memory scrubbing error on
CPU_SrcID#1_MC#0_Chan#0_DIMM#0 (channel:0 slot:0 page:0x57c24fb offset:0x4c0 grain:32
syndrome:0x0 -  OVERFLOW err_code:0008:00c0 socket:1 imc:0 rank:1 bg:3 ba:2 row:1aacf col:338)
```

Region Failover



Disable scaledowns in the evacuated region until traffic comes back.

Future Work

Eager Scale Up

- **Current Scale Up Decision:**
  - There is significant consumer lag AND sink is healthy
  - Utilization exceeds the safe threshold AND sink is healthy

This is not ideal since in most cases latency builds up in the job before a scale up is triggered.

- **Eager Scale Up Decision:**
  - Use the performance table to determine the maximum processing rate of the current cluster.
  - Use regression to determine if the workload will exceed the processing rate of the cluster in the near future.
  - If this is the case do a scale up before any lag builds up.

Downscale Optimization

- Current downscale operation
  - **CHEAP:** Graceful shutdown with savepoint.
  - **EXPENSIVE:** Remove TMs.
  - **CHEAP:** Restart from savepoint with reduced parallelism.
- Optimized downscale operation
  - **CHEAP:** Graceful shutdown with savepoint.
  - **CHEAP:** Blacklist TMs that will be removed.
  - **CHEAP:** Restart from savepoint with reduced parallelism.
  - **EXPENSIVE:** Remove TMs.

Complex DAGs

- Extend the algorithm to support multiple sources and sinks.
- Handle jobs where all operators are not chained.

Acknowledgements

- Steven Wu
- Andrew Nguonly
- Neeraj Joshi
- Mark Cho
- Netflix Flink Team
- Netflix Mantis Team
- Netflix Data Pipeline Team
- Netflix RTDI Team

- https://www.spinnaker.io/
- https://github.com/Netflix/mantis