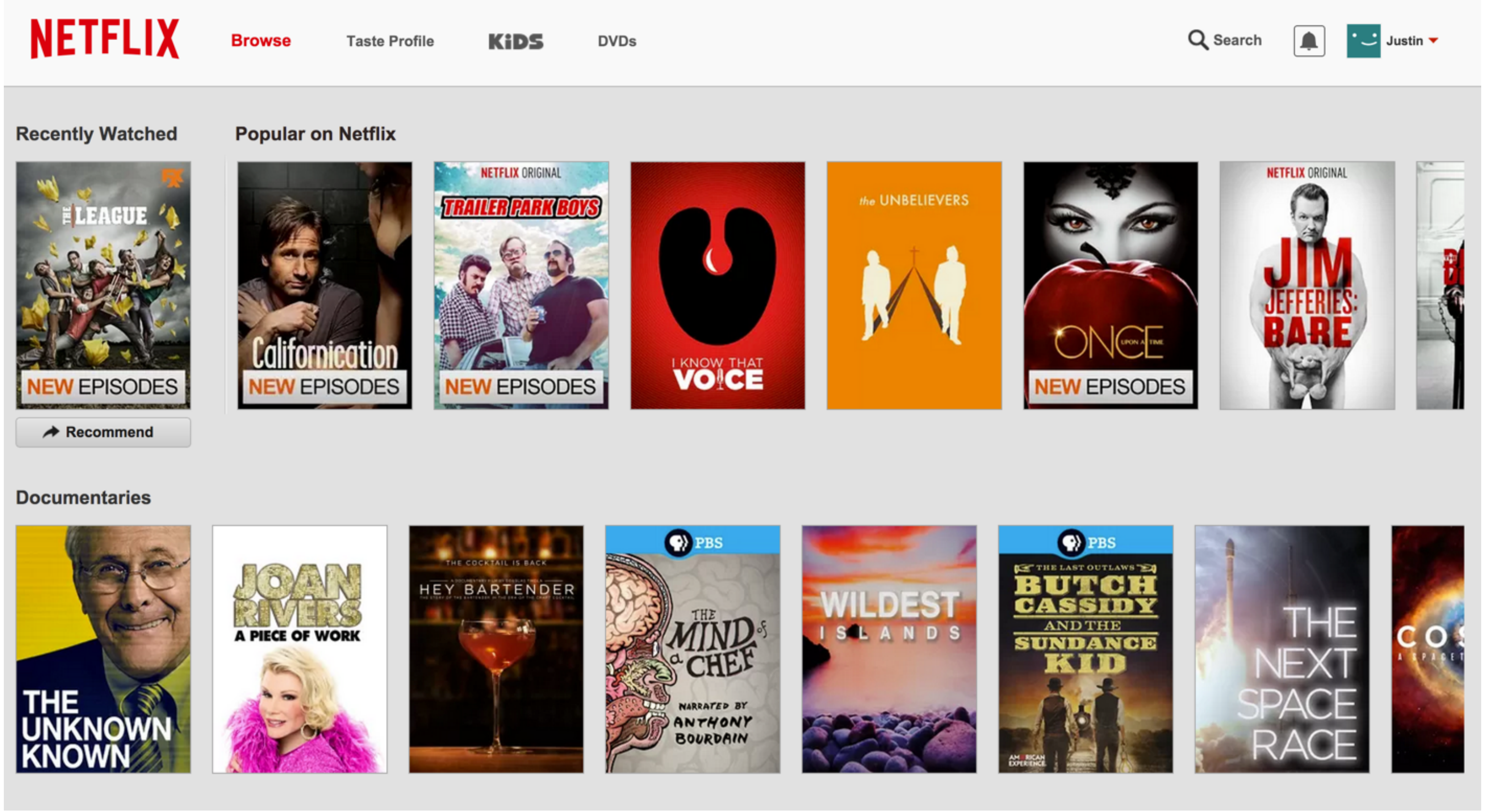
Mantis: Netflix's Event Stream Processing System

Justin Becker Danny Yuan
10/26/2014

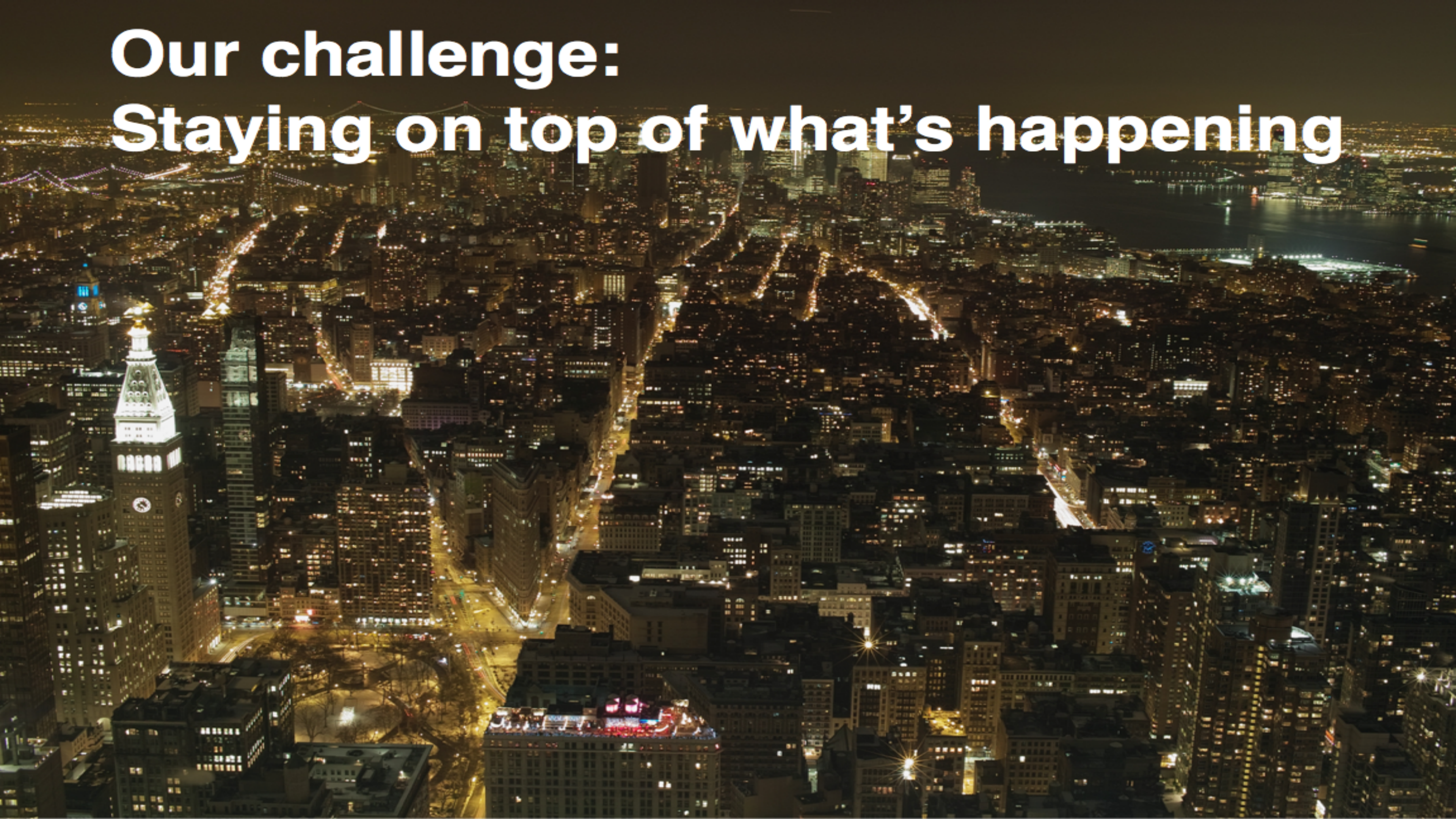
Motivation

Traditional TV just works

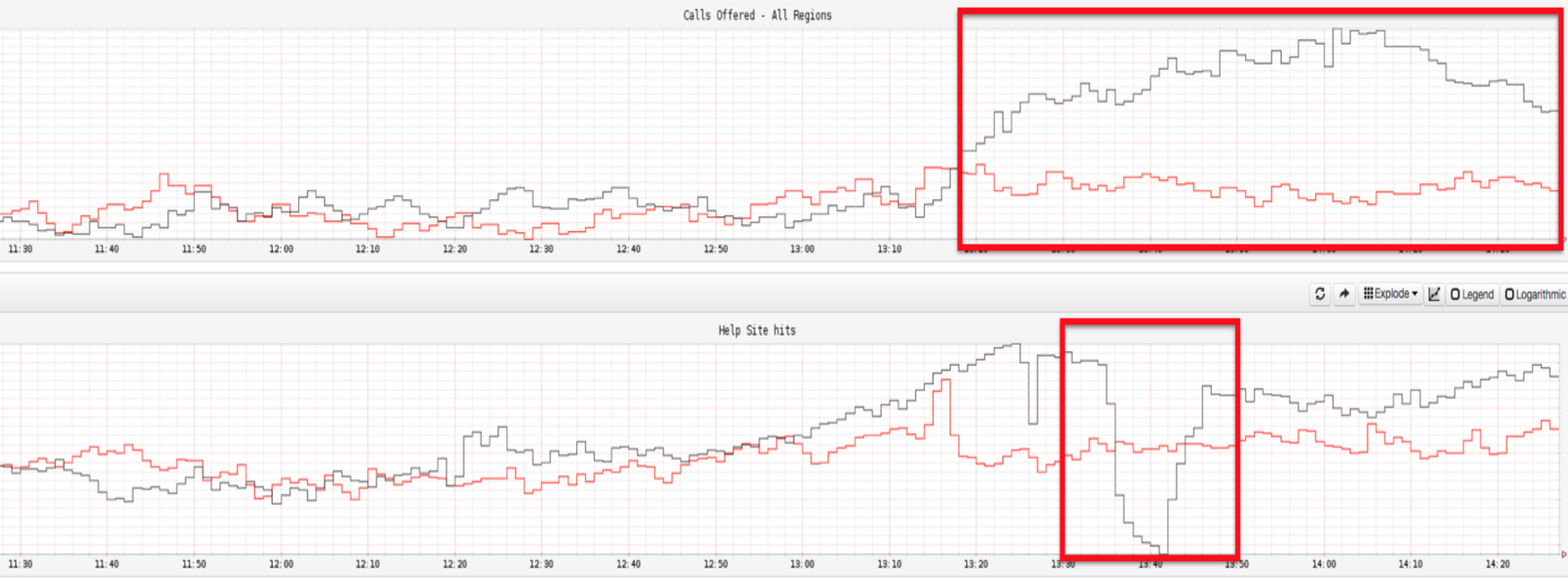
Netflix wants Internet TV to work just as well



Our challenge:
Staying on top of what’s happening

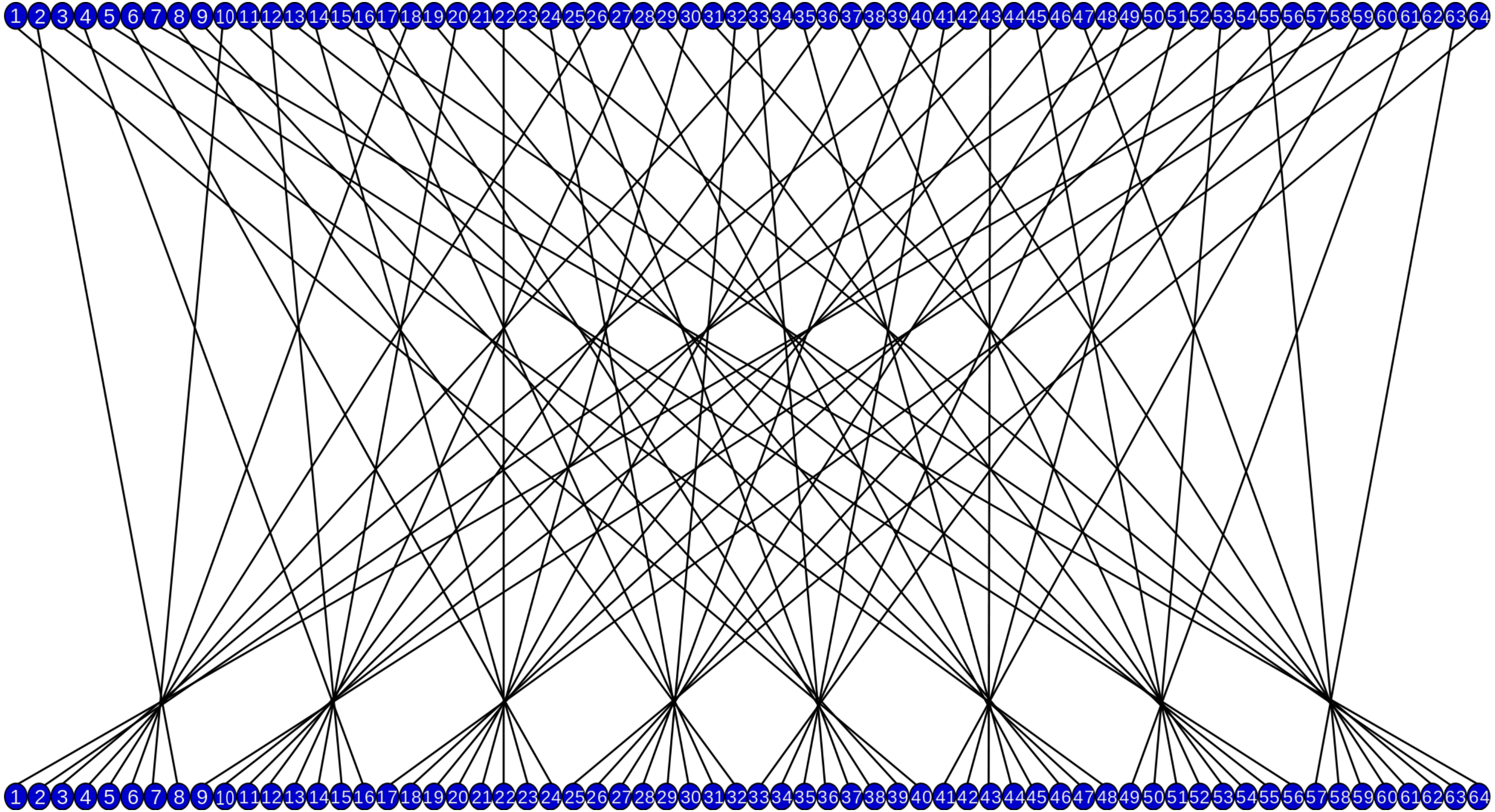


Especially when things aren’t working


Calls Offered - All Regions
11:30
11:40
11:50
12:00
12:10
12:20
12:30
12:40
12:50
13:00
13:10
Explode ▾
Legend
Logarithmic
Help Site hits
11:30
11:40
11:50
12:00
12:10
12:20
12:30
12:40
12:50
13:00
13:10
13:20
13:30
13:40
13:50
14:00
14:10
14:20

Tracking “big signals” is not enough

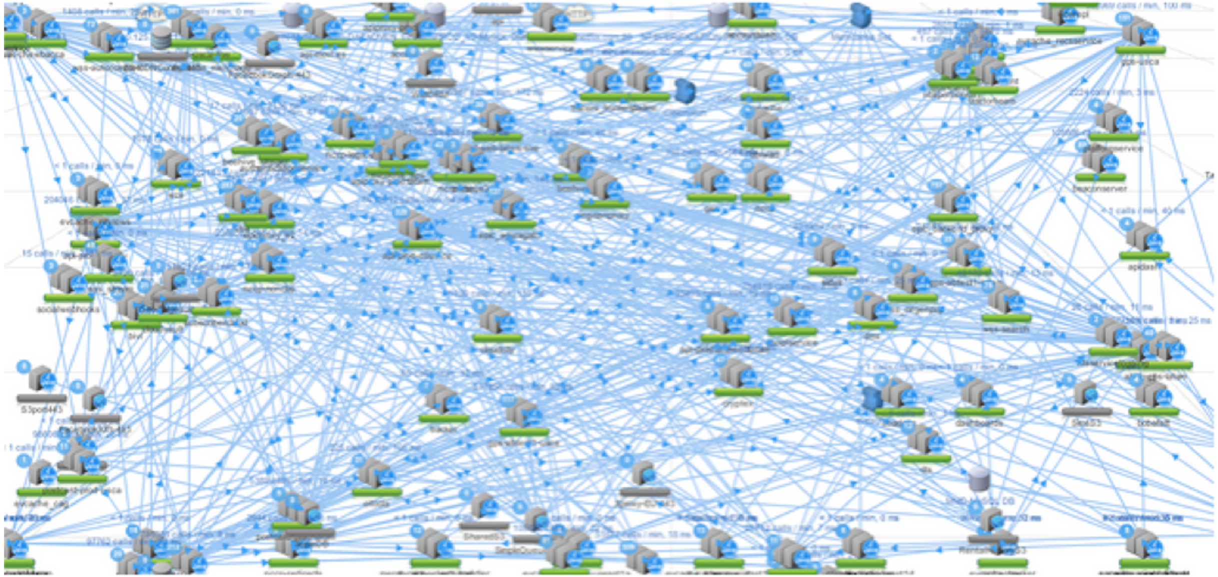
Need to track all kinds of permutations



Detect quickly, to resolve ASAP

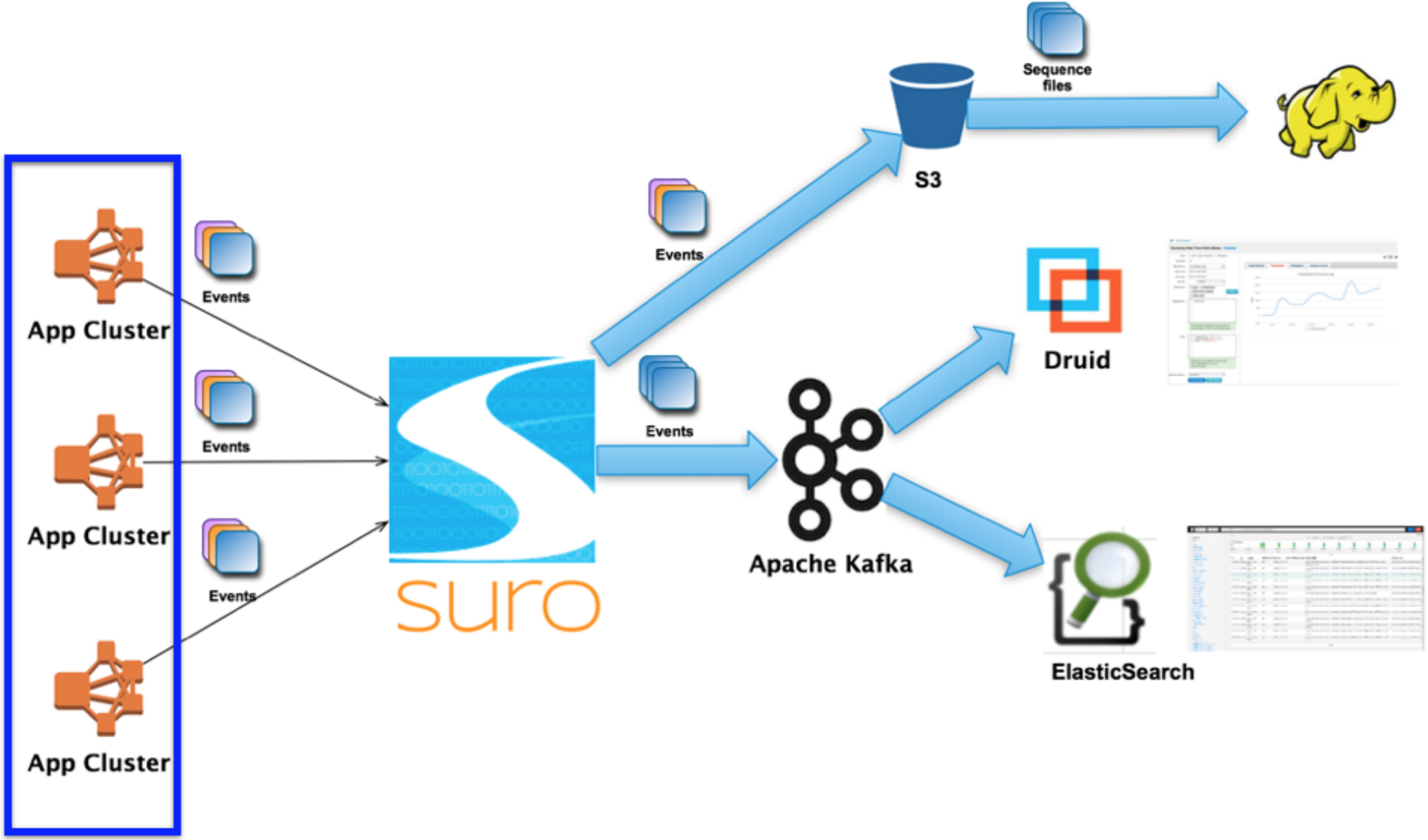
Cheaper than product services

Here comes the problem



12 ~ 20 million metrics updates per second

750 billion metrics updates per day



App Cluster

App Cluster

App Cluster

Events

Events

Events

suro

Events

Events

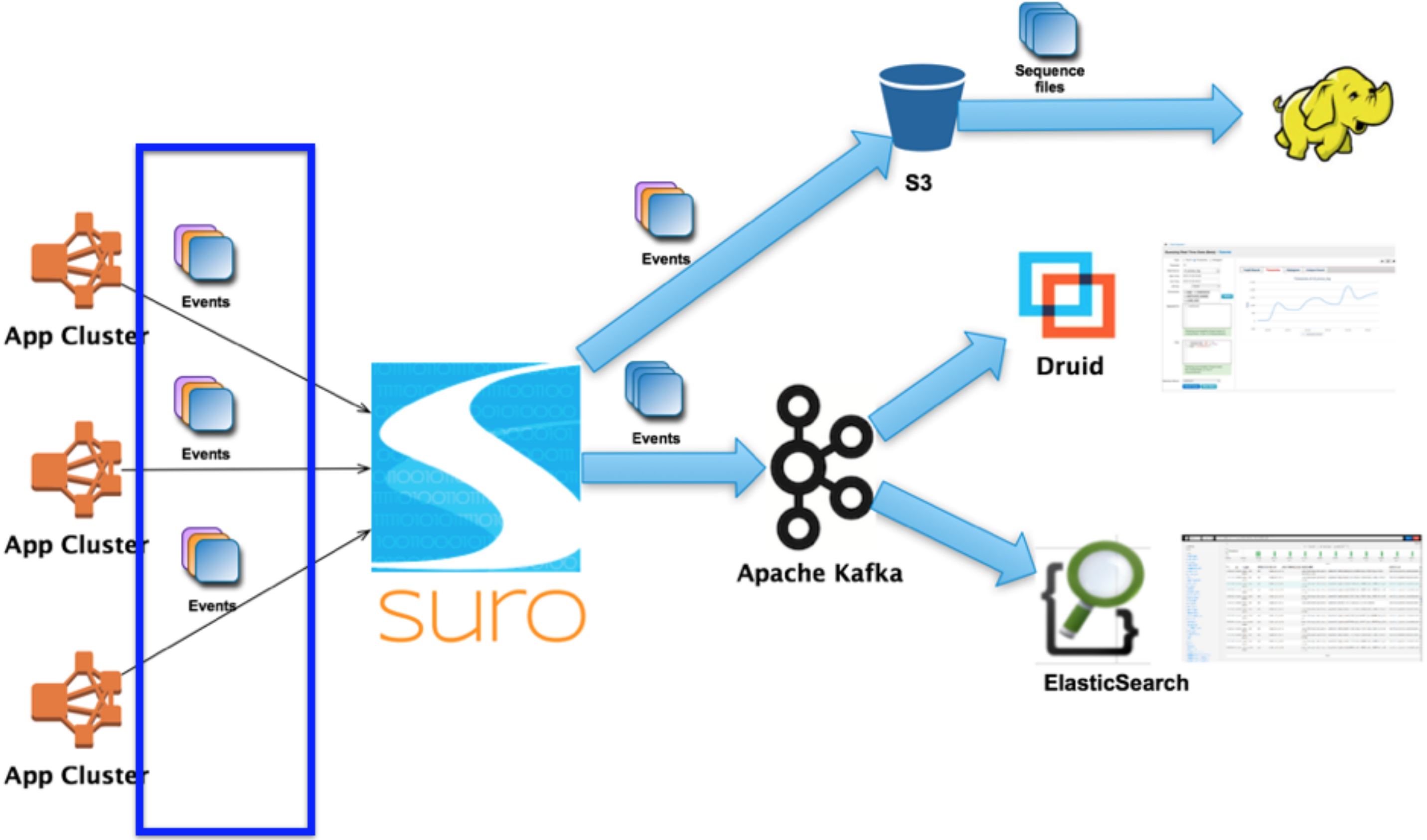
S3

Sequence files

Apache Kafka

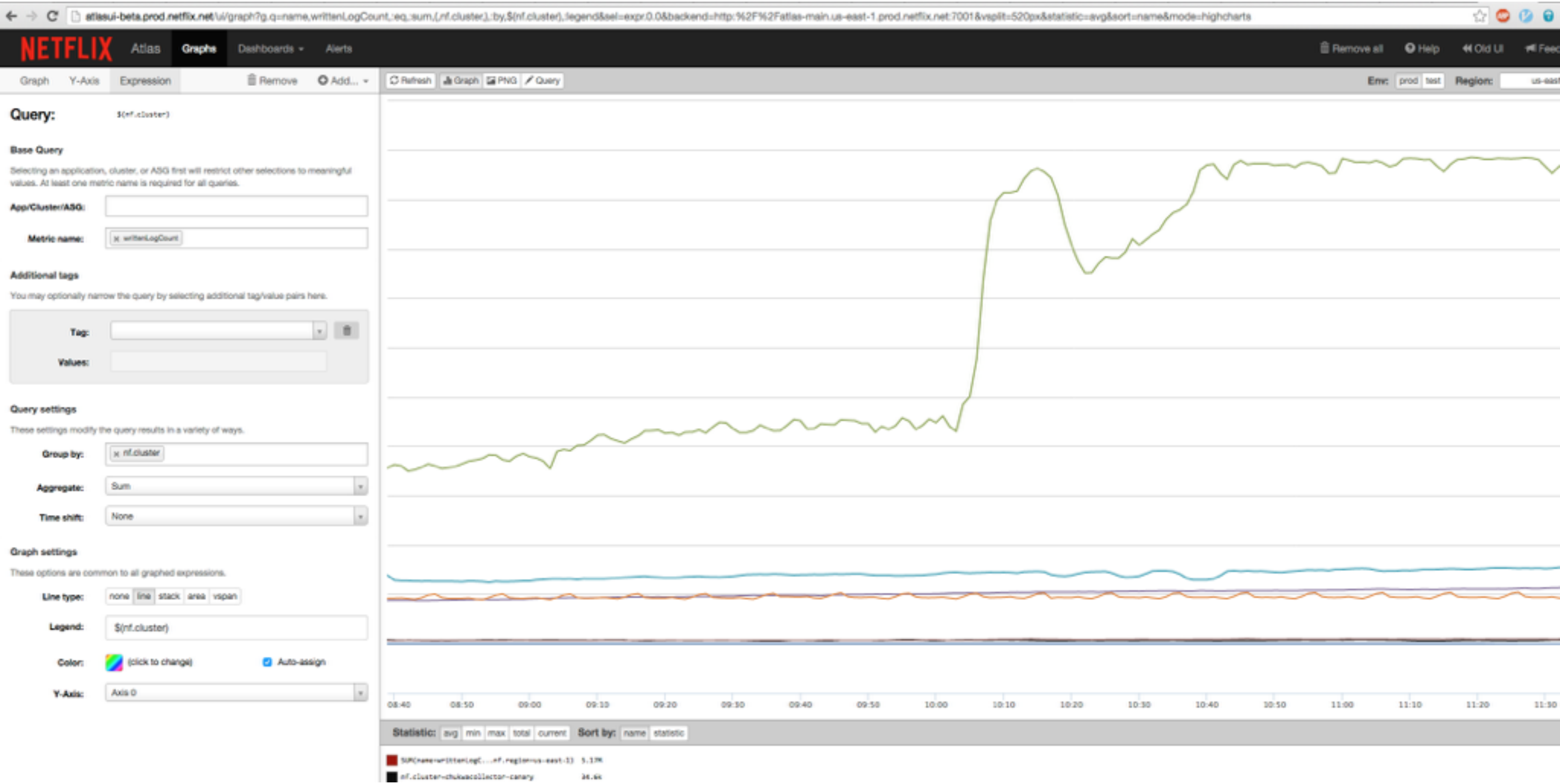
Druid

ElasticSearch


App Cluster
App Cluster
App Cluster
Events
Events
Events
suro
Events
S3
Sequence files
Events
Apache Kafka
Druid
ElasticSearch

4 ~ 18 Million App Events Per Second

> 300 Billion Events Per Day



Query: $(nf.cluster)

Base Query

Selecting an application, cluster, or ASG first will restrict other selections to meaningful values. At least one metric name is required for all queries.

App/Cluster/ASG:

Metric name: × writtenLogCount

Additional tags

You may optionally narrow the query by selecting additional tag/value pairs here.

Tag:

Values:

Query settings

These settings modify the query results in a variety of ways.

Group by: × nf.cluster

Aggregate: Sum

Time shift: None

Graph settings

These options are common to all graphed expressions.

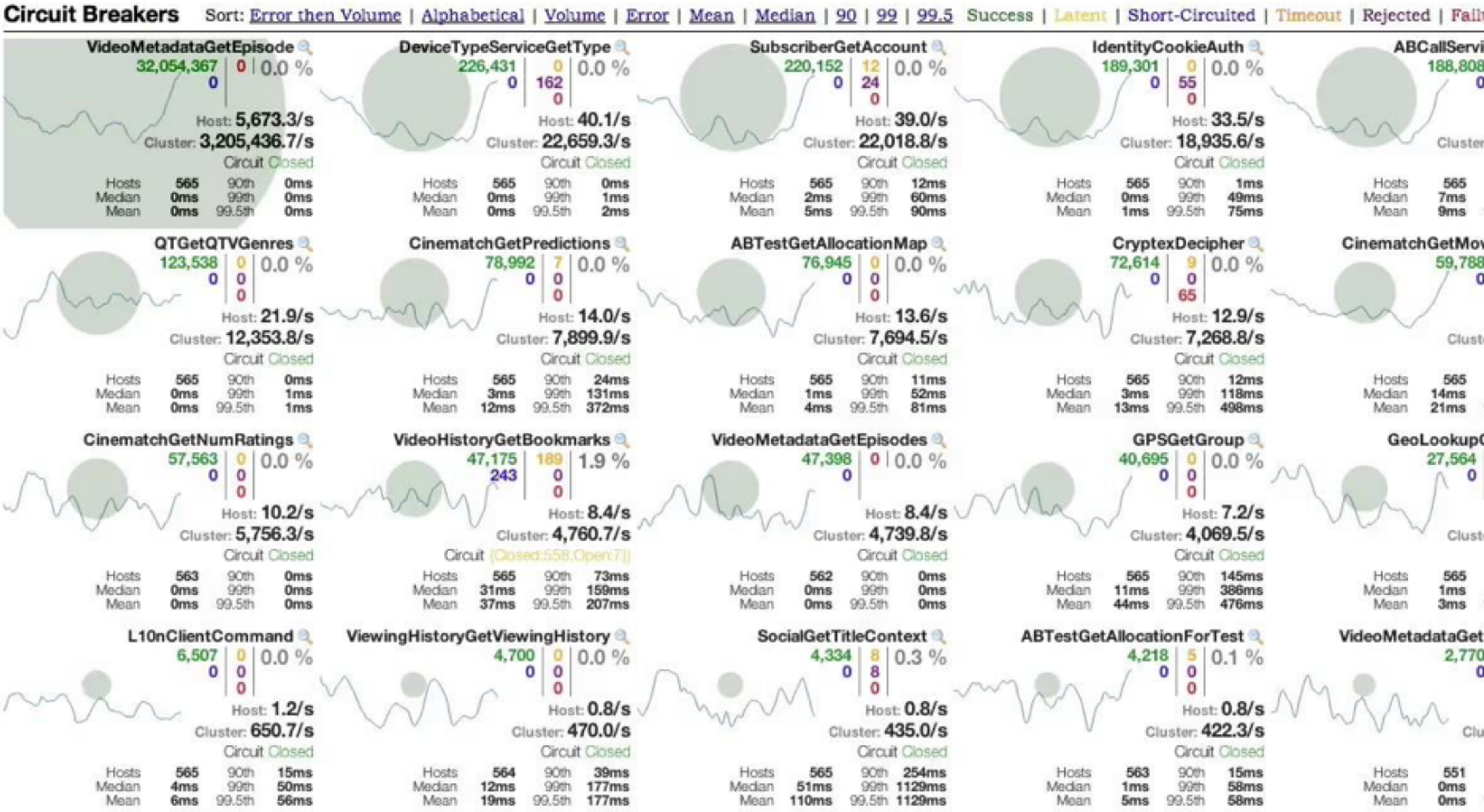
Line type: none [✓ line] stack area vspan

Legend: $(nf.cluster)

Color: (click to change) ☑ Auto-assign

Y-Axis: Axis 0

🟥	SUM(name=writtenLogC...nf.region=us-east-1)	5.17M
⬛	nf.cluster=chukwacollector-canary	36.6k



Circuit Breakers Sort: Error then Volume | Alphabetical | Volume | Error | Mean | Median | 90 | 99 | 99.5 Success | Latent | Short-Circuited | Timeout | Rejected | Failu

VideoMetadataGetEpisode
32,054,367 | 0 | 0.0 %
0
Host: 5,673.3/s
Cluster: 3,205,436.7/s
Circuit Closed
Hosts 565 90th 0ms
Median 0ms 99th 0ms
Mean 0ms 99.5th 0ms

DeviceTypeServiceGetType
226,431 | 0 | 0.0 %
0 | 162
0
Host: 40.1/s
Cluster: 22,659.3/s
Circuit Closed
Hosts 565 90th 0ms
Median 0ms 99th 1ms
Mean 0ms 99.5th 2ms

SubscriberGetAccount
220,152 | 12 | 0.0 %
0 | 24
0
Host: 39.0/s
Cluster: 22,018.8/s
Circuit Closed
Hosts 565 90th 12ms
Median 2ms 99th 60ms
Mean 5ms 99.5th 90ms

IdentityCookieAuth
189,301 | 0 | 0.0 %
0 | 55
0
Host: 33.5/s
Cluster: 18,935.6/s
Circuit Closed
Hosts 565 90th 1ms
Median 0ms 99th 49ms
Mean 1ms 99.5th 75ms

ABCallServi
188,808
0
Cluster
Hosts 565
Median 7ms
Mean 9ms

QTGetQTVGenres
123,538 | 0 | 0.0 %
0 | 0
0
Host: 21.9/s
Cluster: 12,353.8/s
Circuit Closed
Hosts 565 90th 0ms
Median 0ms 99th 1ms
Mean 0ms 99.5th 1ms

CinematchGetPredictions
78,992 | 7 | 0.0 %
0 | 0
0
Host: 14.0/s
Cluster: 7,899.9/s
Circuit Closed
Hosts 565 90th 24ms
Median 3ms 99th 131ms
Mean 12ms 99.5th 372ms

ABTestGetAllocationMap
76,945 | 0 | 0.0 %
0 | 0
0
Host: 13.6/s
Cluster: 7,694.5/s
Circuit Closed
Hosts 565 90th 11ms
Median 1ms 99th 52ms
Mean 4ms 99.5th 81ms

CryptexDecipher
72,614 | 9 | 0.0 %
0 | 0
65
Host: 12.9/s
Cluster: 7,268.8/s
Circuit Closed
Hosts 565 90th 12ms
Median 3ms 99th 118ms
Mean 13ms 99.5th 498ms

CinematchGetMov
59,788
0
Clust
Hosts 565
Median 14ms
Mean 21ms

CinematchGetNumRatings
57,563 | 0 | 0.0 %
0 | 0
0
Host: 10.2/s
Cluster: 5,756.3/s
Circuit Closed
Hosts 563 90th 0ms
Median 0ms 99th 0ms
Mean 0ms 99.5th 0ms

VideoHistoryGetBookmarks
47,175 | 189 | 1.9 %
243 | 0
0
Host: 8.4/s
Cluster: 4,760.7/s
Circuit (Closed:558,Open:7))
Hosts 565 90th 73ms
Median 31ms 99th 159ms
Mean 37ms 99.5th 207ms

VideoMetadataGetEpisodes
47,398 | 0 | 0.0 %
0
Host: 8.4/s
Cluster: 4,739.8/s
Circuit Closed
Hosts 562 90th 0ms
Median 0ms 99th 0ms
Mean 0ms 99.5th 0ms

GPSGetGroup
40,695 | 0 | 0.0 %
0 | 0
0
Host: 7.2/s
Cluster: 4,069.5/s
Circuit Closed
Hosts 565 90th 145ms
Median 11ms 99th 386ms
Mean 44ms 99.5th 476ms

GeoLookup
27,564
0
Clust
Hosts 565
Median 1ms
Mean 3ms

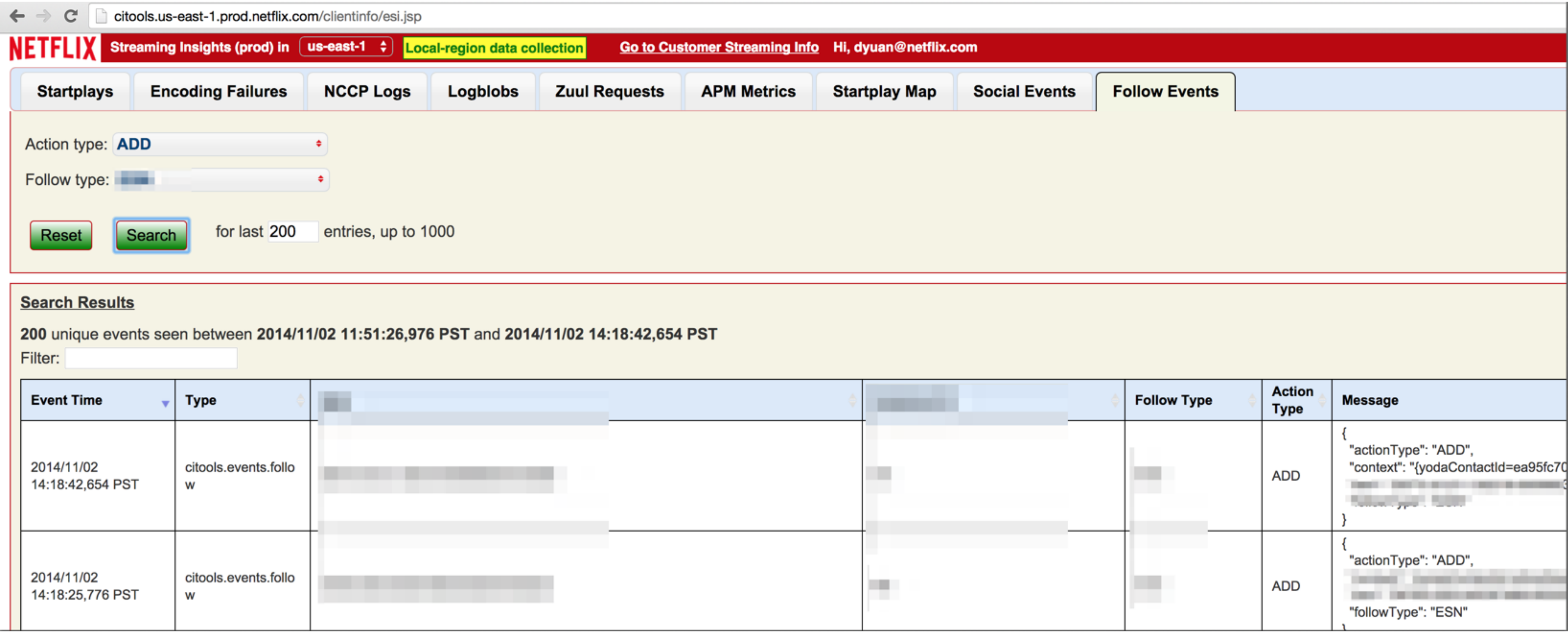
L10nClientCommand
6,507 | 0 | 0.0 %
0 | 0
0
Host: 1.2/s
Cluster: 650.7/s
Circuit Closed
Hosts 565 90th 15ms
Median 4ms 99th 50ms
Mean 6ms 99.5th 56ms

ViewingHistoryGetViewingHistory
4,700 | 0 | 0.0 %
0 | 0
0
Host: 0.8/s
Cluster: 470.0/s
Circuit Closed
Hosts 564 90th 39ms
Median 12ms 99th 177ms
Mean 19ms 99.5th 177ms

SocialGetTitleContext
4,334 | 8 | 0.3 %
0 | 8
0
Host: 0.8/s
Cluster: 435.0/s
Circuit Closed
Hosts 565 90th 254ms
Median 51ms 99th 1129ms
Mean 110ms 99.5th 1129ms

ABTestGetAllocationForTest
4,218 | 5 | 0.1 %
0 | 0
0
Host: 0.8/s
Cluster: 422.3/s
Circuit Closed
Hosts 563 90th 15ms
Median 1ms 99th 58ms
Mean 5ms 99.5th 58ms

VideoMetadataGet
2,770
0
Clu
Hosts 551
Median 0ms
Mean 0ms



Action type: ADD

Follow type: [redacted]

for last 200 entries, up to 1000

Search Results

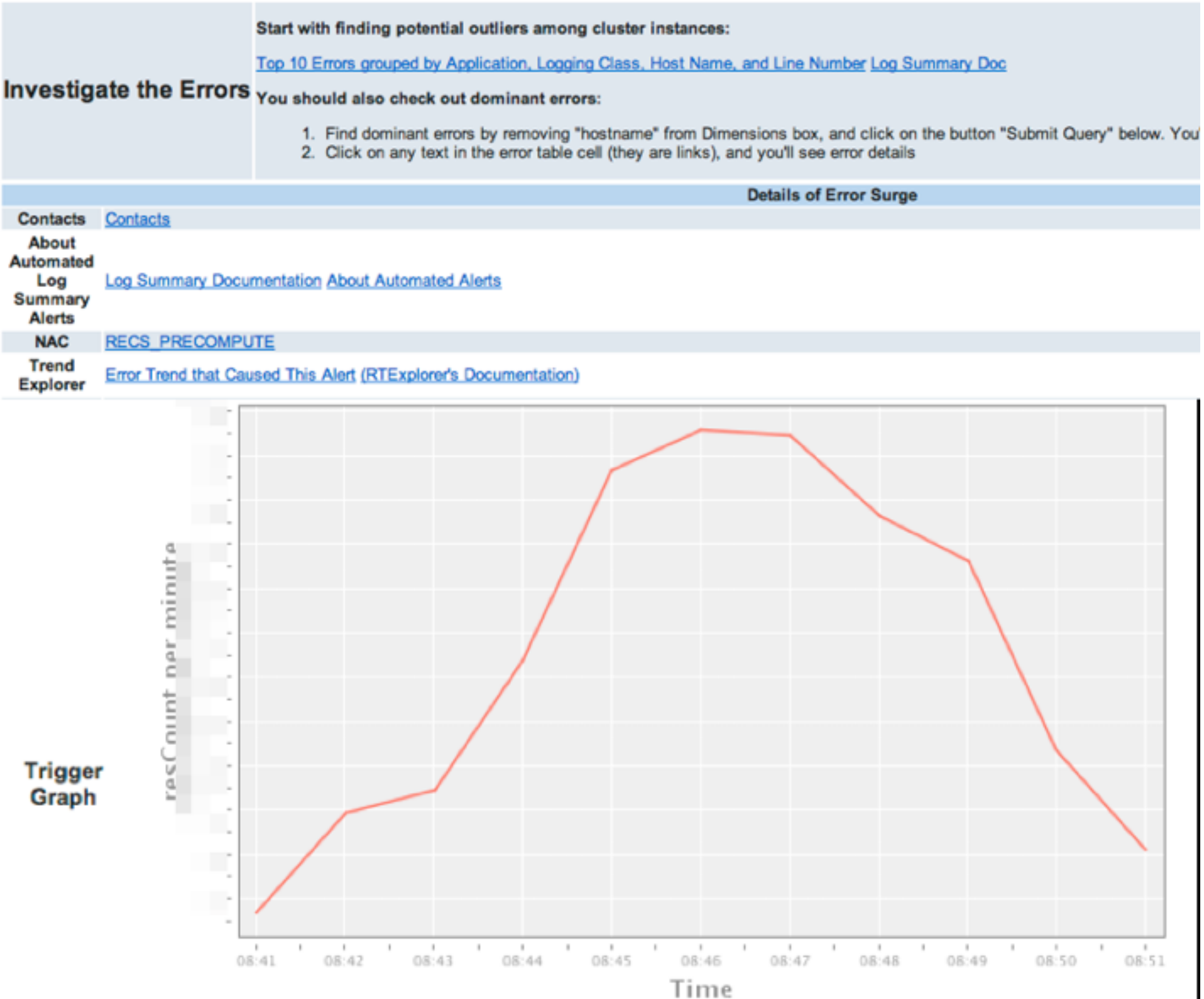
200 unique events seen between **2014/11/02 11:51:26,976 PST** and **2014/11/02 14:18:42,654 PST**

Filter:

Event Time	Type	[redacted]	[redacted]	Follow Type	Action Type	Message

2014/11/02 14:18:42,654 PST	citools.events.follow	[redacted]	[redacted]	[redacted]	ADD	{
						"actionType": "ADD",
						"context": "{yodaContactId=ea95fc70
						[redacted]
						}

2014/11/02 14:18:25,776 PST	citools.events.follow	[redacted]	[redacted]	[redacted]	ADD	{ "actionType": "ADD", [redacted] "followType": "ESN"



Investigate the Errors

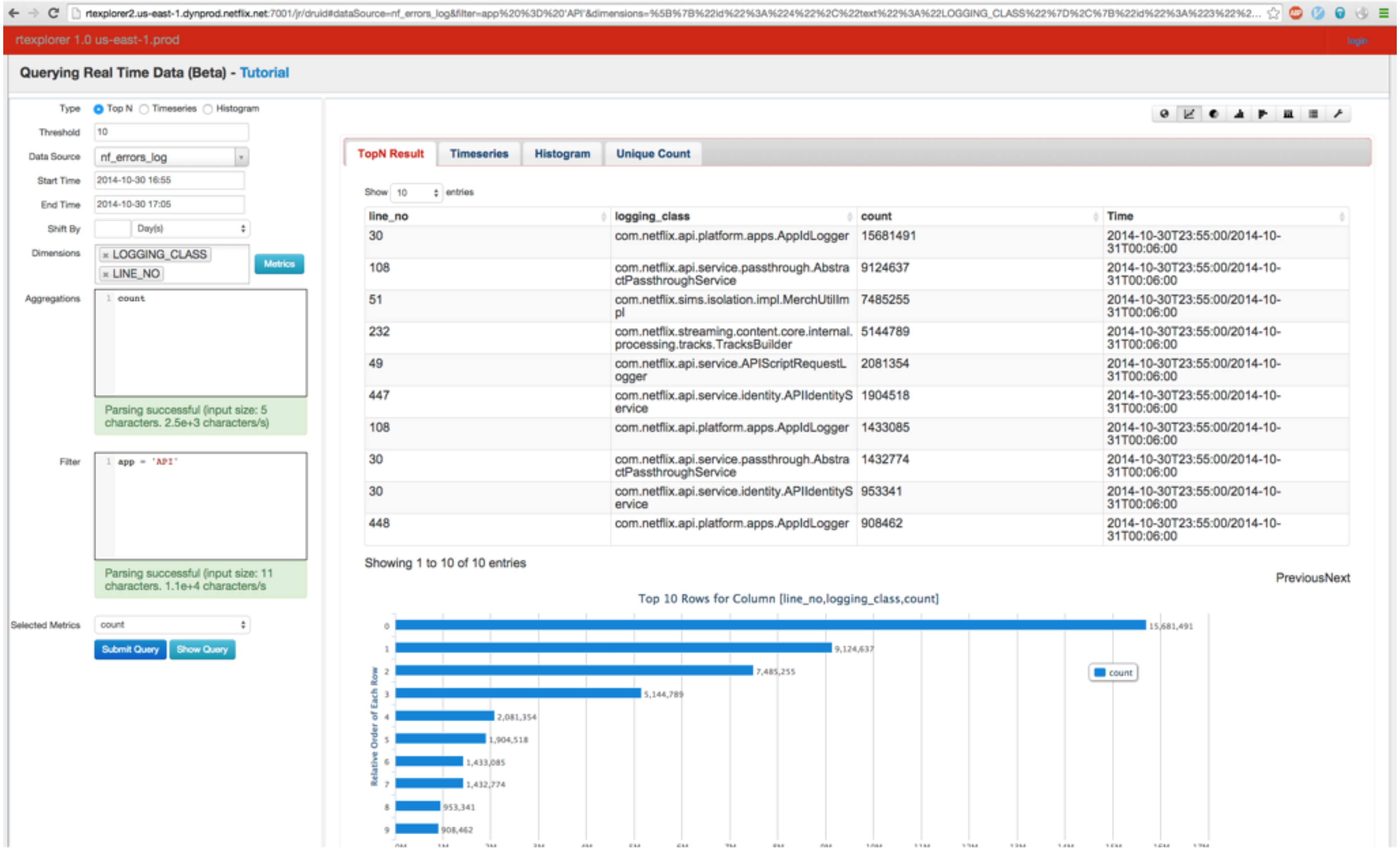
Start with finding potential outliers among cluster instances:

Top 10 Errors grouped by Application, Logging Class, Host Name, and Line Number Log Summary Doc

You should also check out dominant errors:

1. Find dominant errors by removing "hostname" from Dimensions box, and click on the button "Submit Query" below. You'
2. Click on any text in the error table cell (they are links), and you'll see error details

	Details of Error Surge
Contacts	Contacts
About Automated Log Summary Alerts	Log Summary Documentation About Automated Alerts
NAC	RECS_PRECOMPUTE
Trend Explorer	Error Trend that Caused This Alert (RTExplorer's Documentation)
Trigger Graph	



rtexplorer2.us-east-1.dynprod.netflix.net:7001/jr/druid#dataSource=nf_errors_log&filter=app%20%3D%20'API'&dimensions=%5B%7B%22id%22%3A%224%22%2C%22text%22%3A%22LOGGING_CLASS%22%7D%2C%7B%22id%22%3A%223%22%2...

rtexplorer 1.0 us-east-1.prod

login

Querying Real Time Data (Beta) - Tutorial

Type ● Top N ○ Timeseries ○ Histogram

Threshold: 10

Data Source: nf_errors_log

Start Time: 2014-10-30 16:55

End Time: 2014-10-30 17:05

Shift By: Day(s)

Dimensions: × LOGGING_CLASS × LINE_NO

Metrics

Aggregations:

```
count
```

Parsing successful (input size: 5 characters. 2.5e+3 characters/s)

Filter:

```
app = 'API'
```

Parsing successful (input size: 11 characters. 1.1e+4 characters/s

Selected Metrics: count

Submit Query

Show Query

TopN Result | Timeseries | Histogram | Unique Count

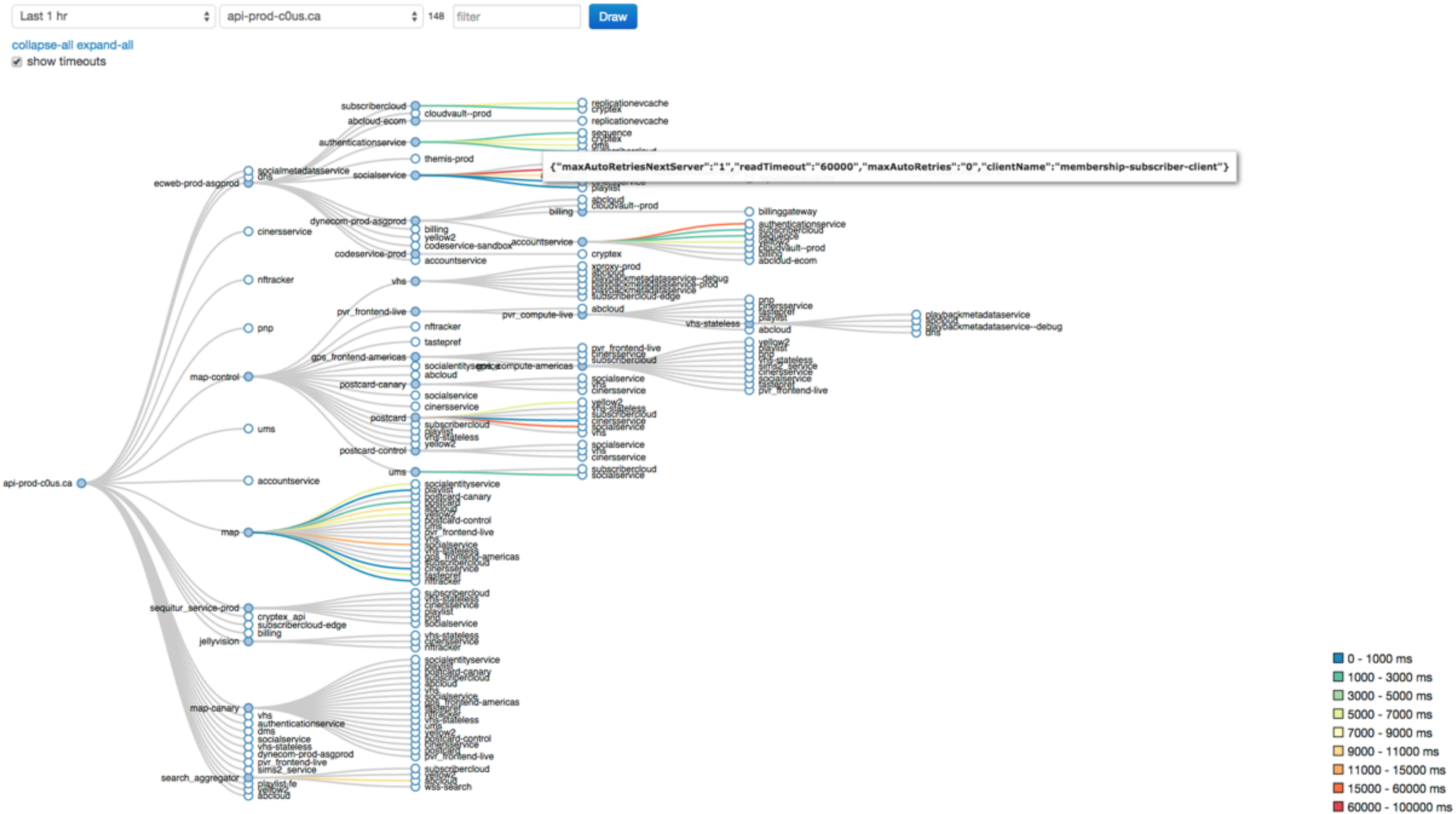
Show 10 entries

line_no	logging_class	count	Time
30	com.netflix.api.platform.apps.AppIdLogger	15681491	2014-10-30T23:55:00/2014-10-31T00:06:00
108	com.netflix.api.service.passthrough.AbstractPassthroughService	9124637	2014-10-30T23:55:00/2014-10-31T00:06:00
51	com.netflix.sims.isolation.impl.MerchUtilImpl	7485255	2014-10-30T23:55:00/2014-10-31T00:06:00
232	com.netflix.streaming.content.core.internal.processing.tracks.TracksBuilder	5144789	2014-10-30T23:55:00/2014-10-31T00:06:00
49	com.netflix.api.service.APIScriptRequestLogger	2081354	2014-10-30T23:55:00/2014-10-31T00:06:00
447	com.netflix.api.service.identity.APIIdentityService	1904518	2014-10-30T23:55:00/2014-10-31T00:06:00
108	com.netflix.api.platform.apps.AppIdLogger	1433085	2014-10-30T23:55:00/2014-10-31T00:06:00
30	com.netflix.api.service.passthrough.AbstractPassthroughService	1432774	2014-10-30T23:55:00/2014-10-31T00:06:00
30	com.netflix.api.service.identity.APIIdentityService	953341	2014-10-30T23:55:00/2014-10-31T00:06:00
448	com.netflix.api.platform.apps.AppIdLogger	908462	2014-10-30T23:55:00/2014-10-31T00:06:00

Showing 1 to 10 of 10 entries

PreviousNext

Top 10 Rows for Column [line_no,logging_class,count]



Last 1 hr ⬍ api-prod-c0us.ca ⬍ 148 filter Draw

collapse-all expand-all

☑ show timeouts

```
{"maxAutoRetriesNextServer":"1","readTimeout":"60000","maxAutoRetries":"0","clientName":"membership-subscriber-client"}
```

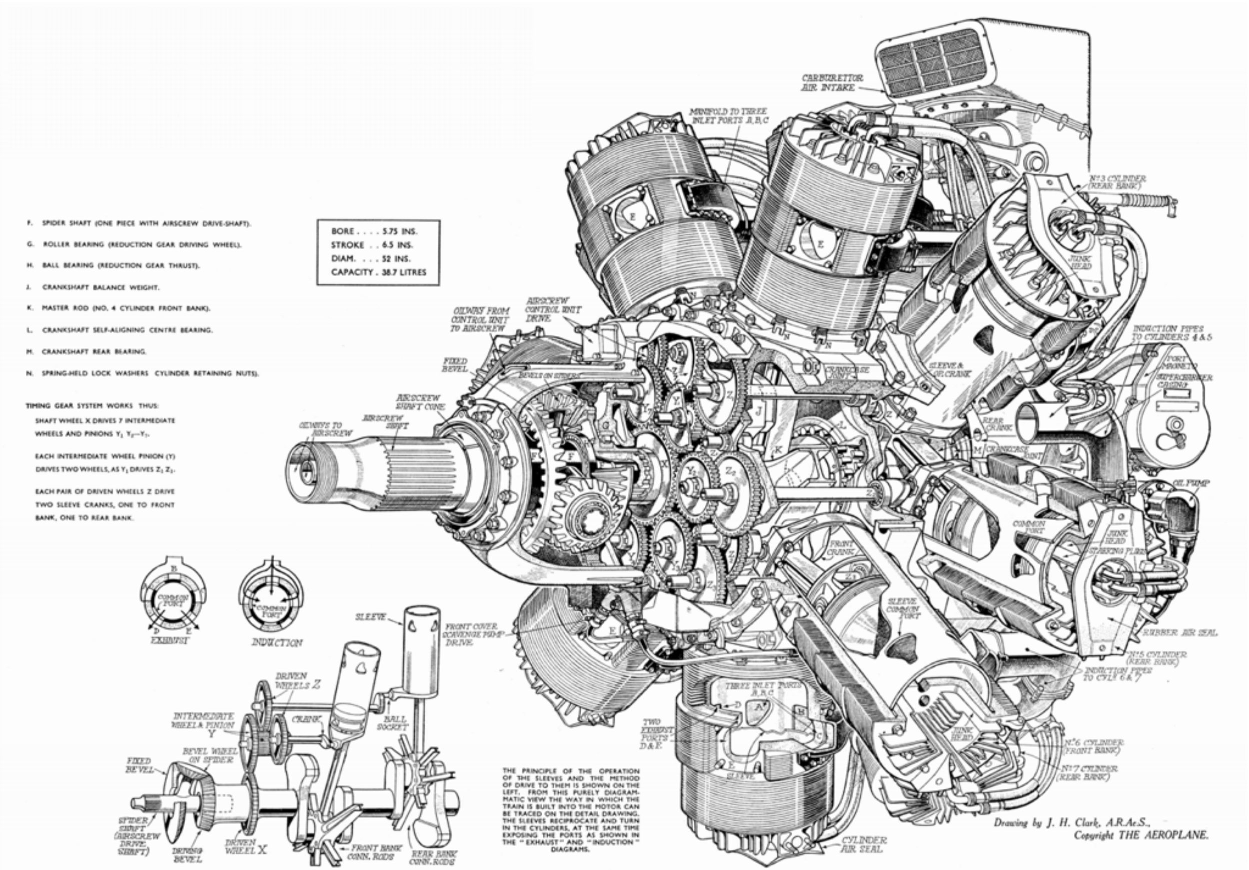
🟦 0 - 1000 ms
🟩 1000 - 3000 ms
🟩 3000 - 5000 ms
🟨 5000 - 7000 ms
⬜ 7000 - 9000 ms
🟨 9000 - 11000 ms
🟧 11000 - 15000 ms
🟧 15000 - 60000 ms
🟥 60000 - 100000 ms

Something is still missing

The Solutions Are Fragmented

They Solve Specific Problems

They Require Lots of Domain Knowledge



F. SPIDER SHAFT (ONE PIECE WITH AIRSCREW DRIVE-SHAFT).

G. ROLLER BEARING (REDUCTION GEAR DRIVING WHEEL).

H. BALL BEARING (REDUCTION GEAR THRUST).

J. CRANKSHAFT BALANCE WEIGHT.

K. MASTER ROD (NO. 4 CYLINDER FRONT BANK).

L. CRANKSHAFT SELF-ALIGNING CENTRE BEARING.

M. CRANKSHAFT REAR BEARING.

N. SPRING-HELD LOCK WASHERS CYLINDER RETAINING NUTS).

TIMING GEAR SYSTEM WORKS THUS:

SHAFT WHEEL X DRIVES 7 INTERMEDIATE WHEELS AND PINIONS Y_1 Y_2—Y_7.

EACH INTERMEDIATE WHEEL PINION (Y) DRIVES TWO WHEELS, AS Y_1 DRIVES Z_1 Z_2.

EACH PAIR OF DRIVEN WHEELS Z DRIVE TWO SLEEVE CRANKS, ONE TO FRONT BANK, ONE TO REAR BANK.

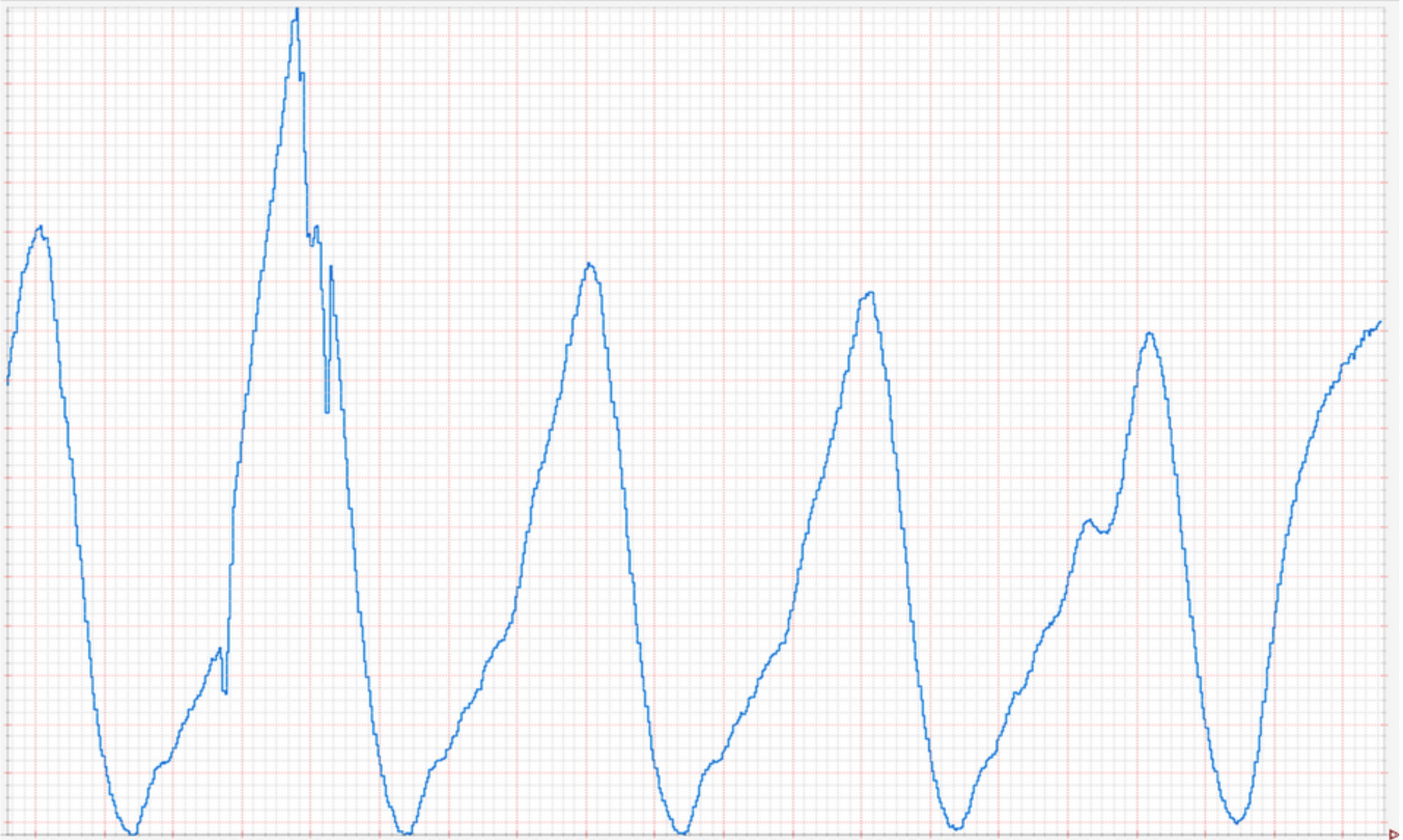
BORE . . . . 5.75 INS.
STROKE . . 6.5 INS.
DIAM. . . . 52 INS.
CAPACITY . 38.7 LITRES

THE PRINCIPLE OF THE OPERATION OF THE SLEEVES AND THE METHOD OF DRIVE TO THEM IS SHOWN ON THE LEFT. FROM THIS PURELY DIAGRAMMATIC VIEW THE WAY IN WHICH THE TRAIN IS BUILT INTO THE MOTOR CAN BE TRACED ON THE DETAIL DRAWING. THE SLEEVES RECIPROCATE AND TURN IN THE CYLINDERS, AT THE SAME TIME EXPOSING THE PORTS AS SHOWN IN THE "EXHAUST" AND "INDUCTION" DIAGRAMS.

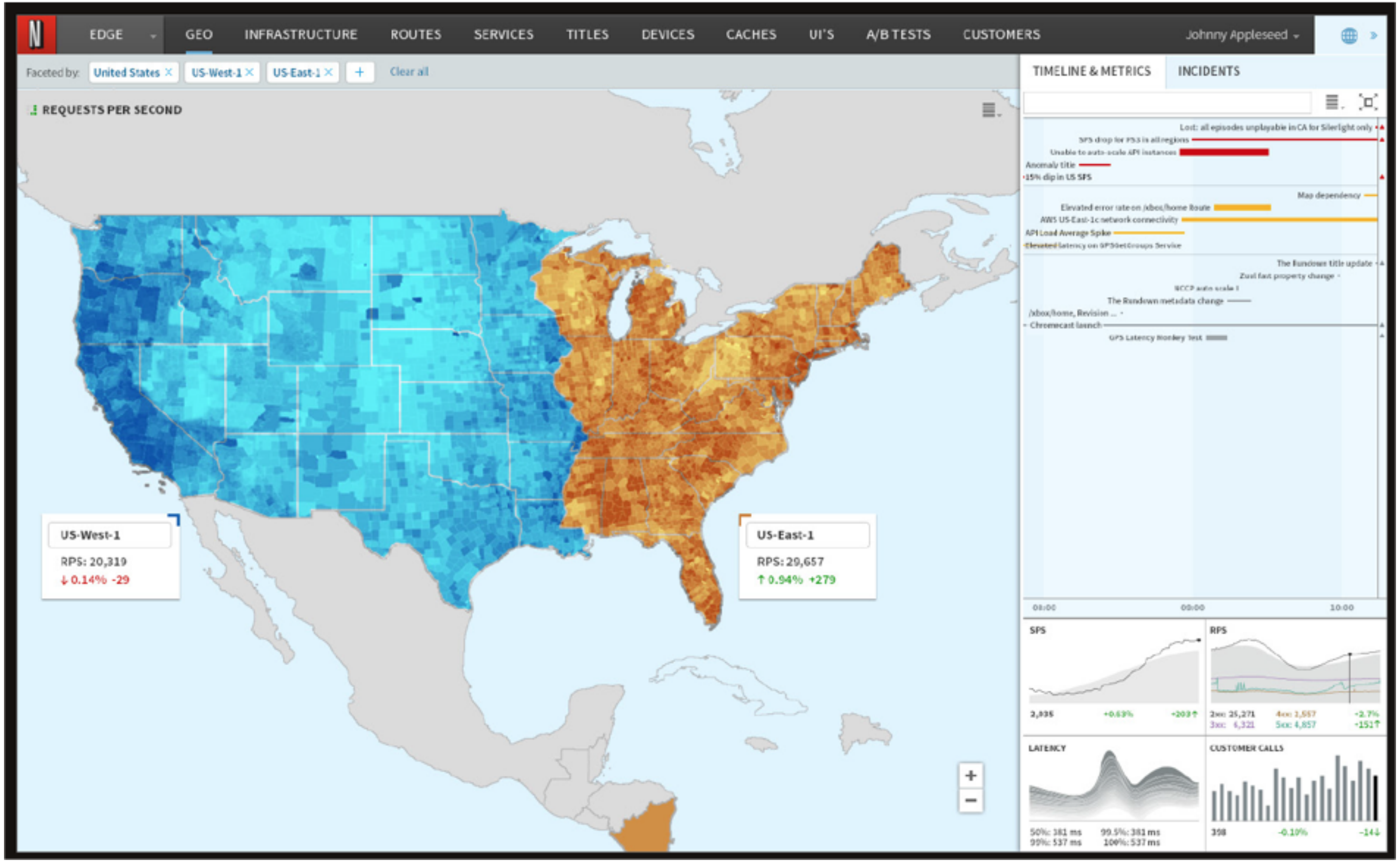
Drawing by J. H. Clark, A.R.Ae.S.,
Copyright THE AEROPLANE.

A System to Rule Them All

But…Requirements Change



But…Requirements Change



US-West-1
RPS: 20,319
↓0.14% -29

US-East-1
RPS: 29,657
↑0.94% +279

Lost: all episodes unplayable in CA for Silerlight only
SPS drop for PS3 in all regions
Unable to auto-scale API instances
Anomaly title
15% dip in US SPS
Map dependency
Elevated error rate on /xbox/home Route
AWS US-East-1c network connectivity
API Load Average Spike
Elevated latency on GPSGetGroups Service
The Rundown title update
Zuul fast property change
NCCP auto scale I
The Rundown metadata change
/xbox/home, Revision ...
Chromecast launch
GPS Latency Monkey Test

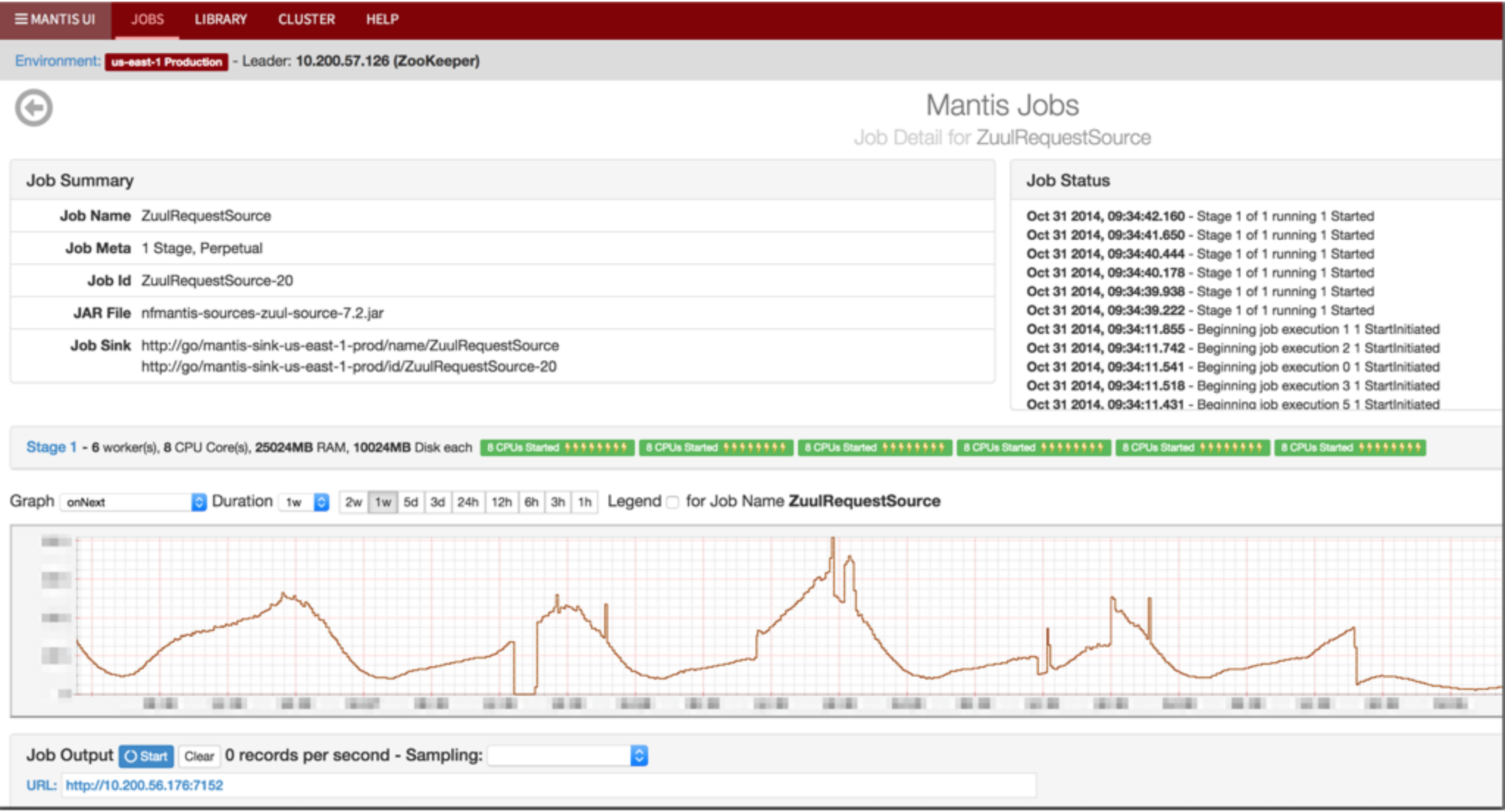
SPS: 2,935 +0.63% +203↑

RPS: 2xx: 25,271 4xx: 1,557 +2.7%
3xx: 6,321 5xx: 4,857 +151↑

LATENCY: 50%: 381 ms 99.5%: 381 ms
95%: 537 ms 100%: 537 ms

CUSTOMER CALLS: 358 -0.10% -14↓

Mantis: A Stream Processing System



Environment: us-east-1 Production - Leader: **10.200.57.126 (ZooKeeper)**

Mantis Jobs

Job Detail for ZuulRequestSource

Job Summary

Job Name	ZuulRequestSource
Job Meta	1 Stage, Perpetual
Job Id	ZuulRequestSource-20
JAR File	nfmantis-sources-zuul-source-7.2.jar
Job Sink	http://go/mantis-sink-us-east-1-prod/name/ZuulRequestSource http://go/mantis-sink-us-east-1-prod/id/ZuulRequestSource-20

Job Status

```
Oct 31 2014, 09:34:42.160 - Stage 1 of 1 running 1 Started
Oct 31 2014, 09:34:41.650 - Stage 1 of 1 running 1 Started
Oct 31 2014, 09:34:40.444 - Stage 1 of 1 running 1 Started
Oct 31 2014, 09:34:40.178 - Stage 1 of 1 running 1 Started
Oct 31 2014, 09:34:39.938 - Stage 1 of 1 running 1 Started
Oct 31 2014, 09:34:39.222 - Stage 1 of 1 running 1 Started
Oct 31 2014, 09:34:11.855 - Beginning job execution 1 1 StartInitiated
Oct 31 2014, 09:34:11.742 - Beginning job execution 2 1 StartInitiated
Oct 31 2014, 09:34:11.541 - Beginning job execution 0 1 StartInitiated
Oct 31 2014, 09:34:11.518 - Beginning job execution 3 1 StartInitiated
Oct 31 2014, 09:34:11.431 - Beginning job execution 5 1 StartInitiated
```

Stage 1 - **6** worker(s), **8** CPU Core(s), **25024MB** RAM, **10024MB** Disk each 8 CPUs Started 8 CPUs Started 8 CPUs Started 8 CPUs Started 8 CPUs Started 8 CPUs Started

Graph: onNext Duration: 1w Legend □ for Job Name **ZuulRequestSource**

Job Output 0 records per second - Sampling:

URL: http://10.200.56.176:7152

3

1. Versatile User Demands

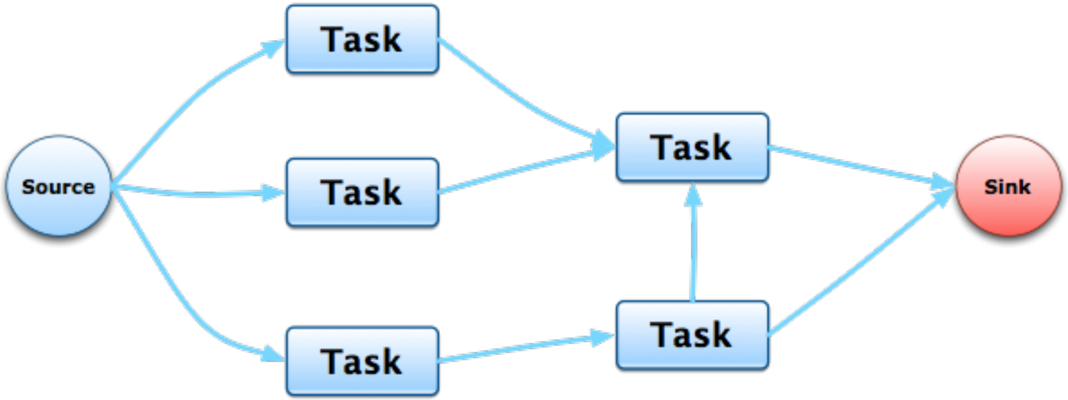
transformation detection

correlation anomaly

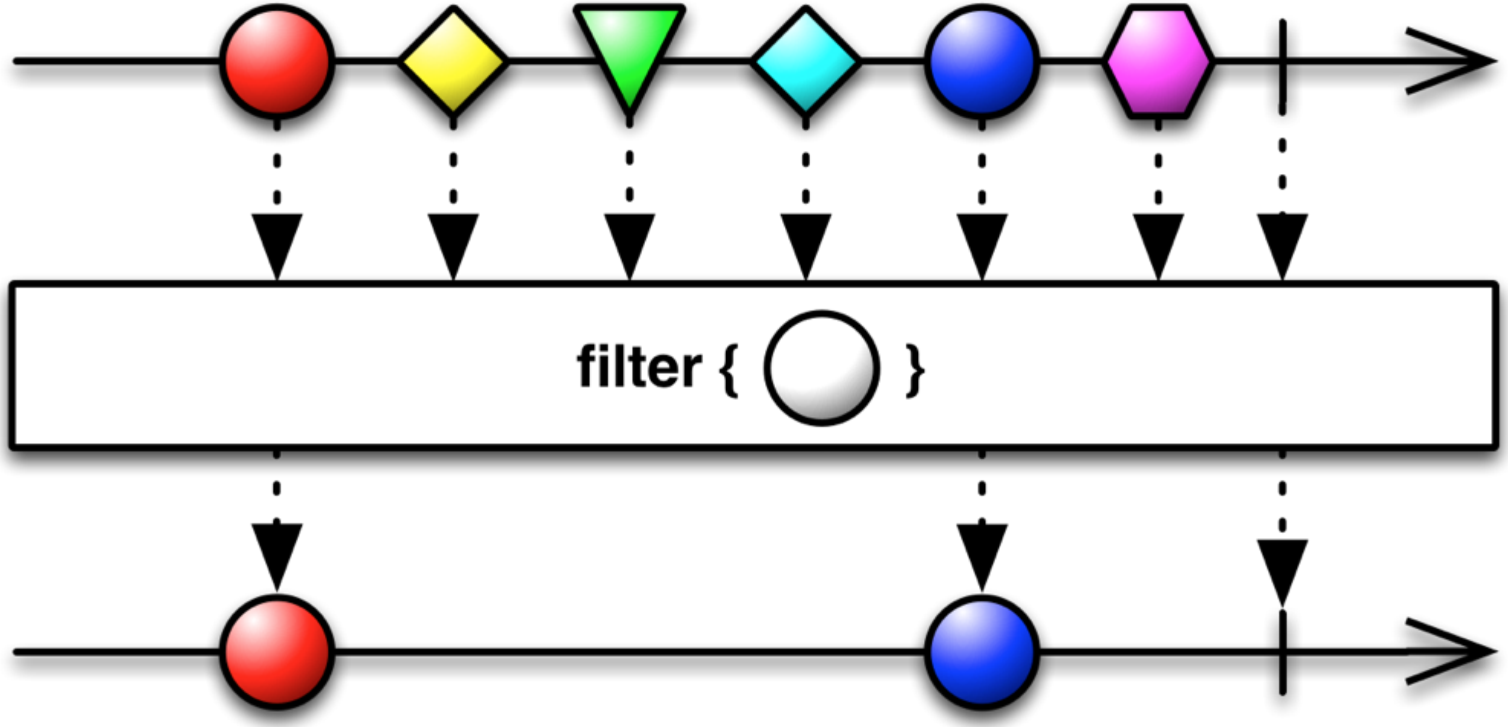
prediction outlier join validation

filtering

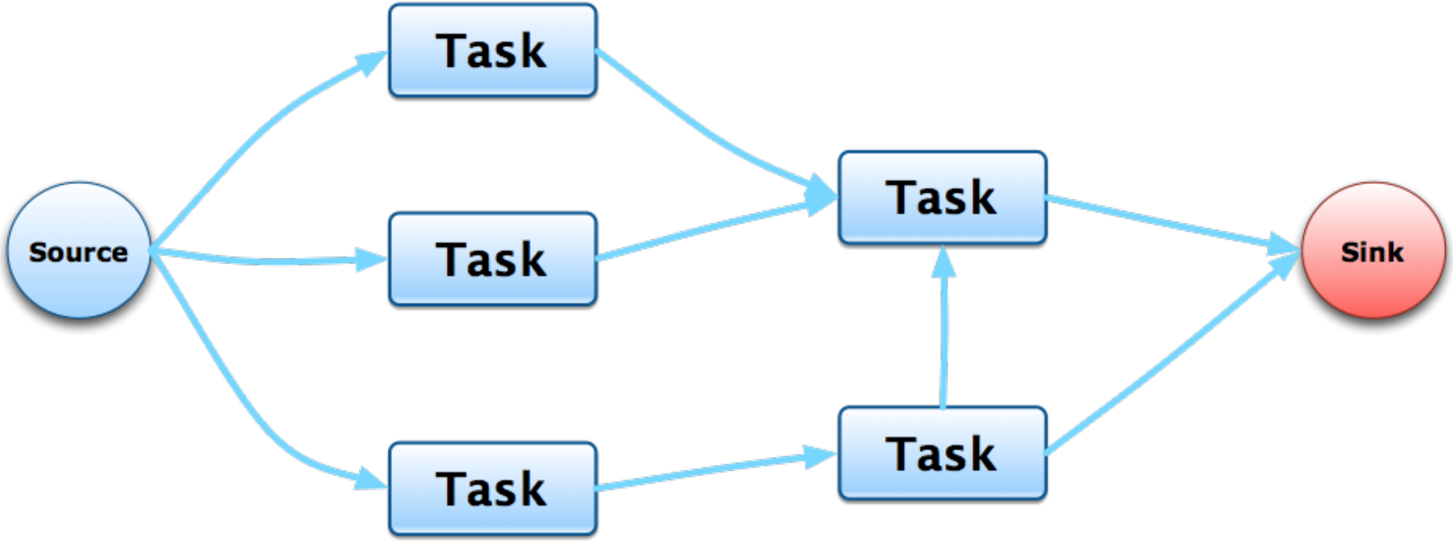
Solution:
Flexible Programming Model



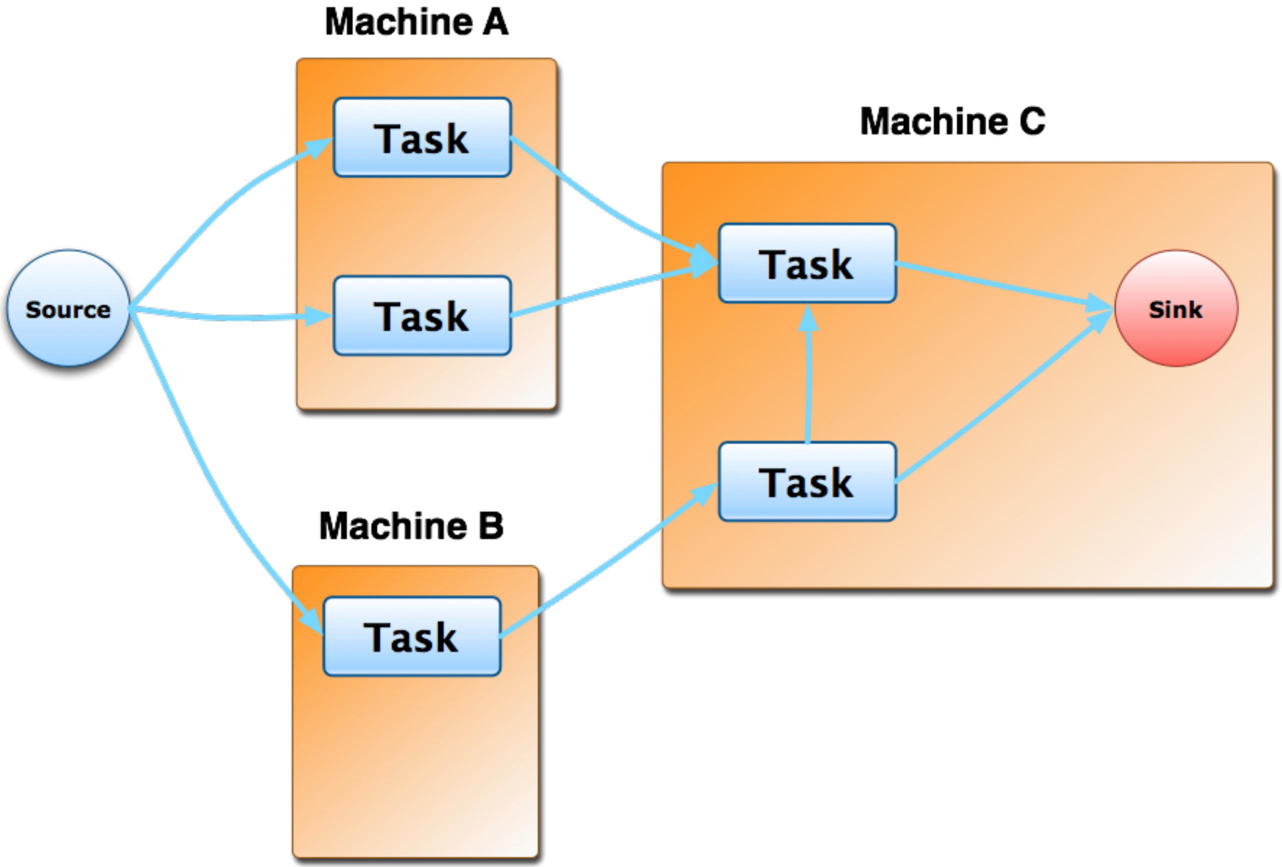
Users Deal with Data Stream Sequentially



Models computation as distributed DAG

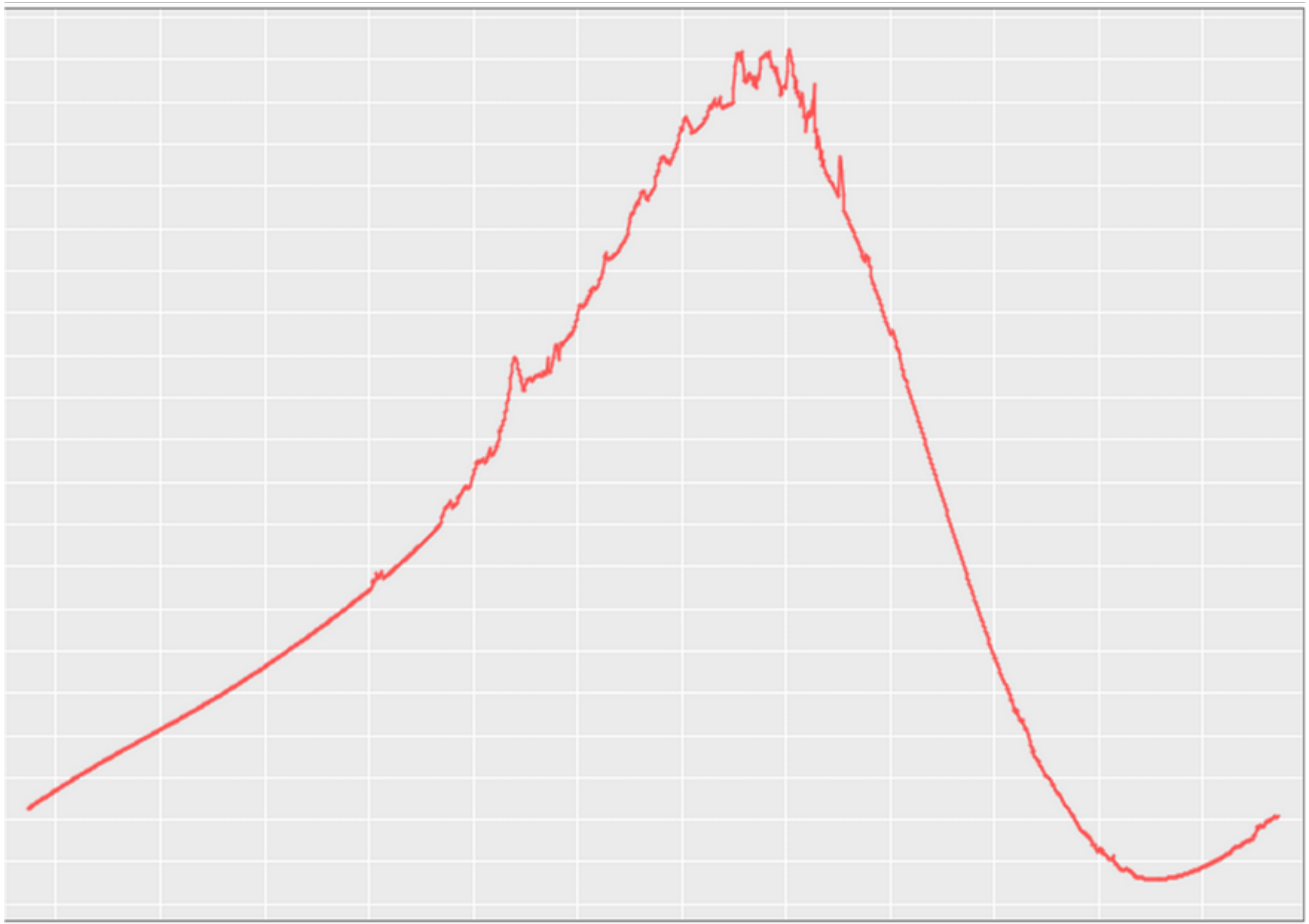


Models computation as distributed DAG

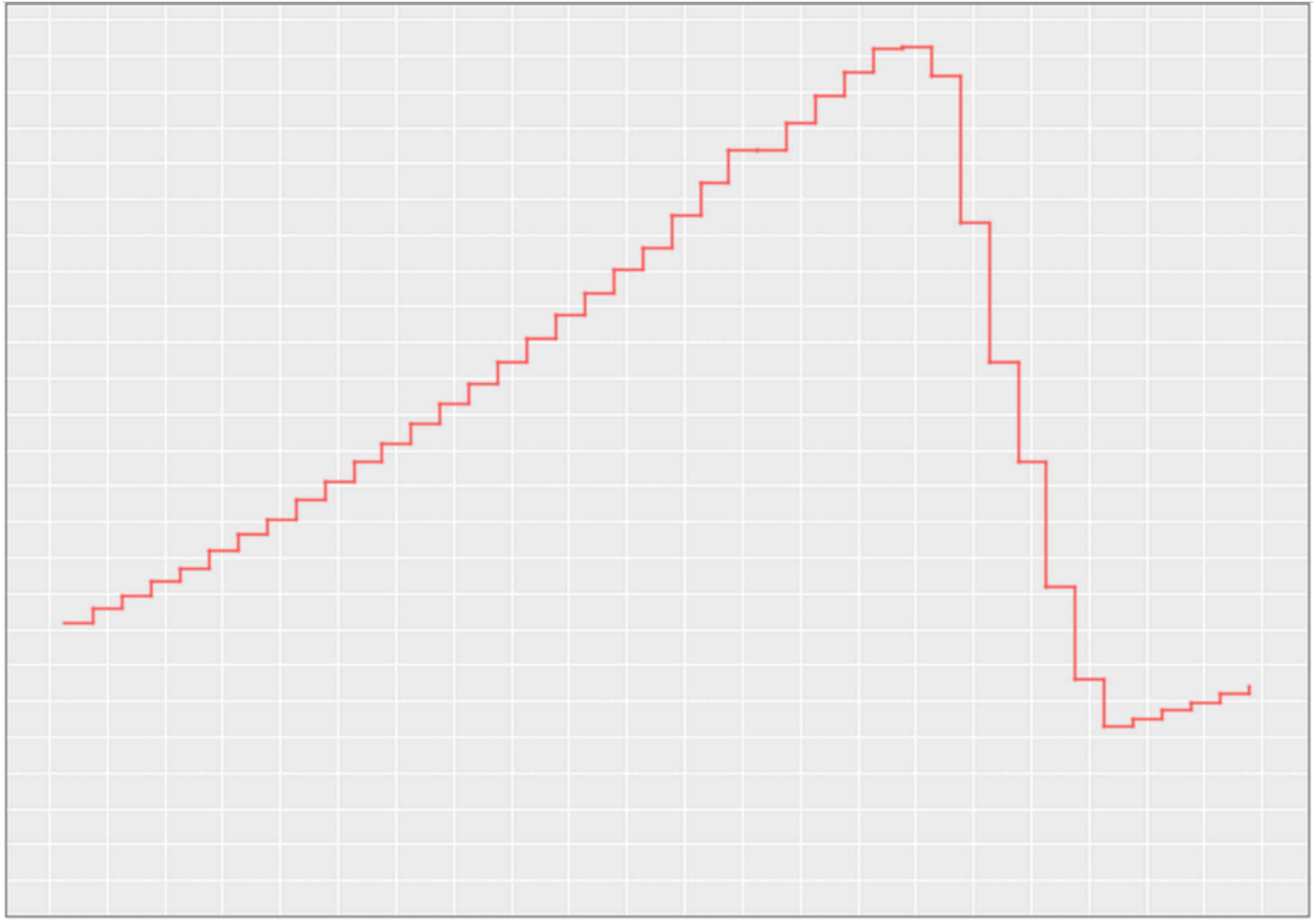


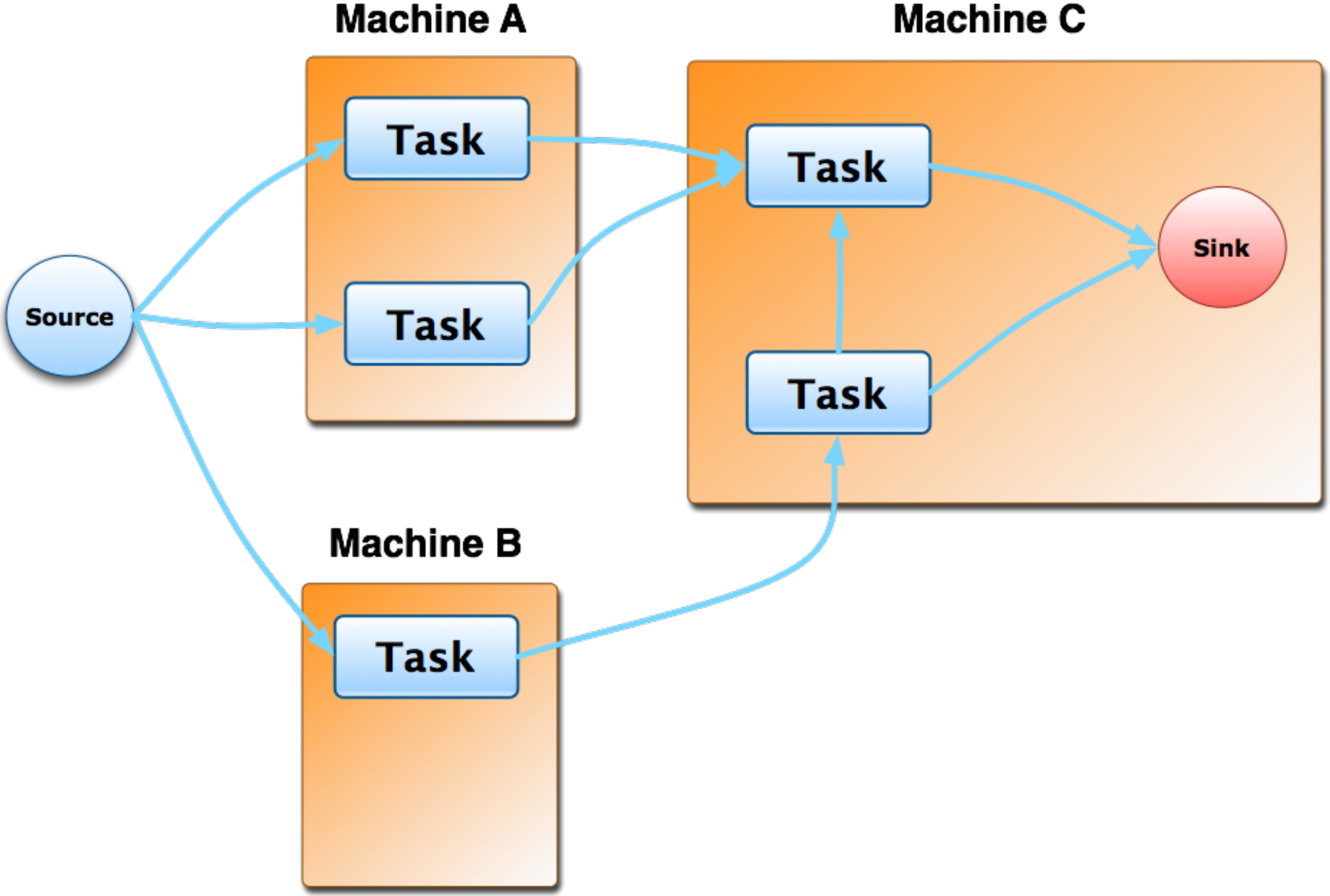
Asynchronous Computation Everywhere

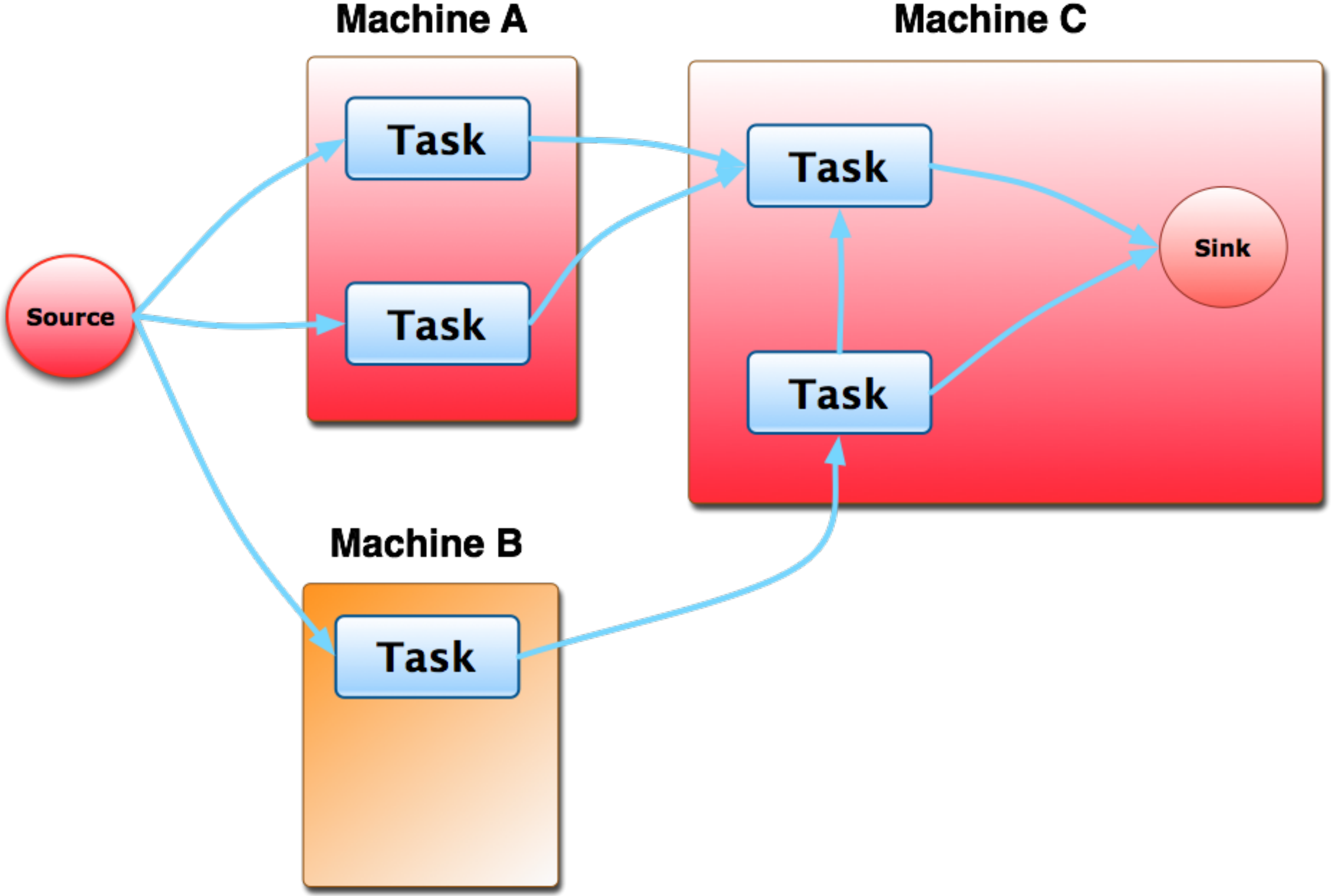
2: Traffic Fluctuates, A Lot

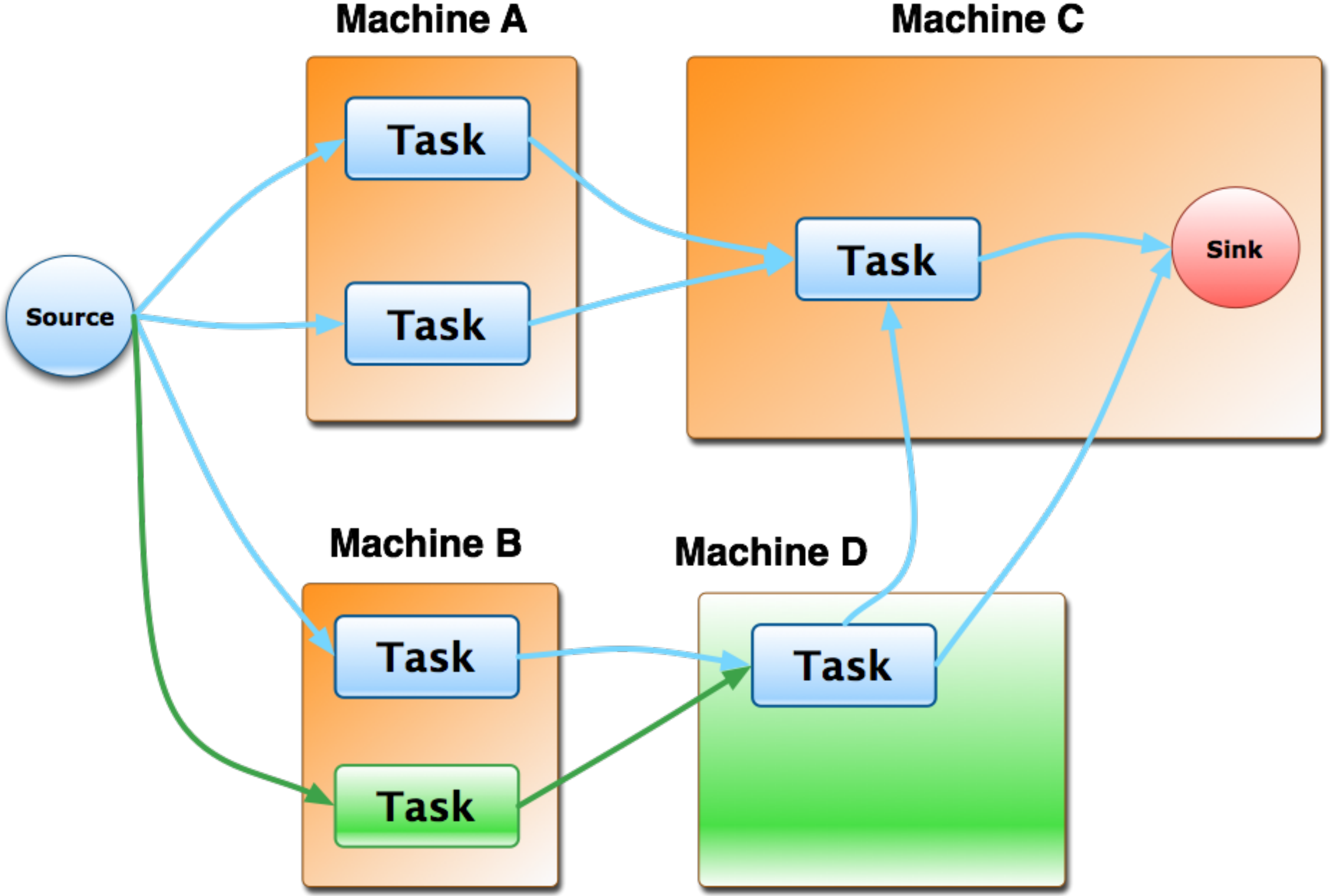


Solution: Auto Scaling



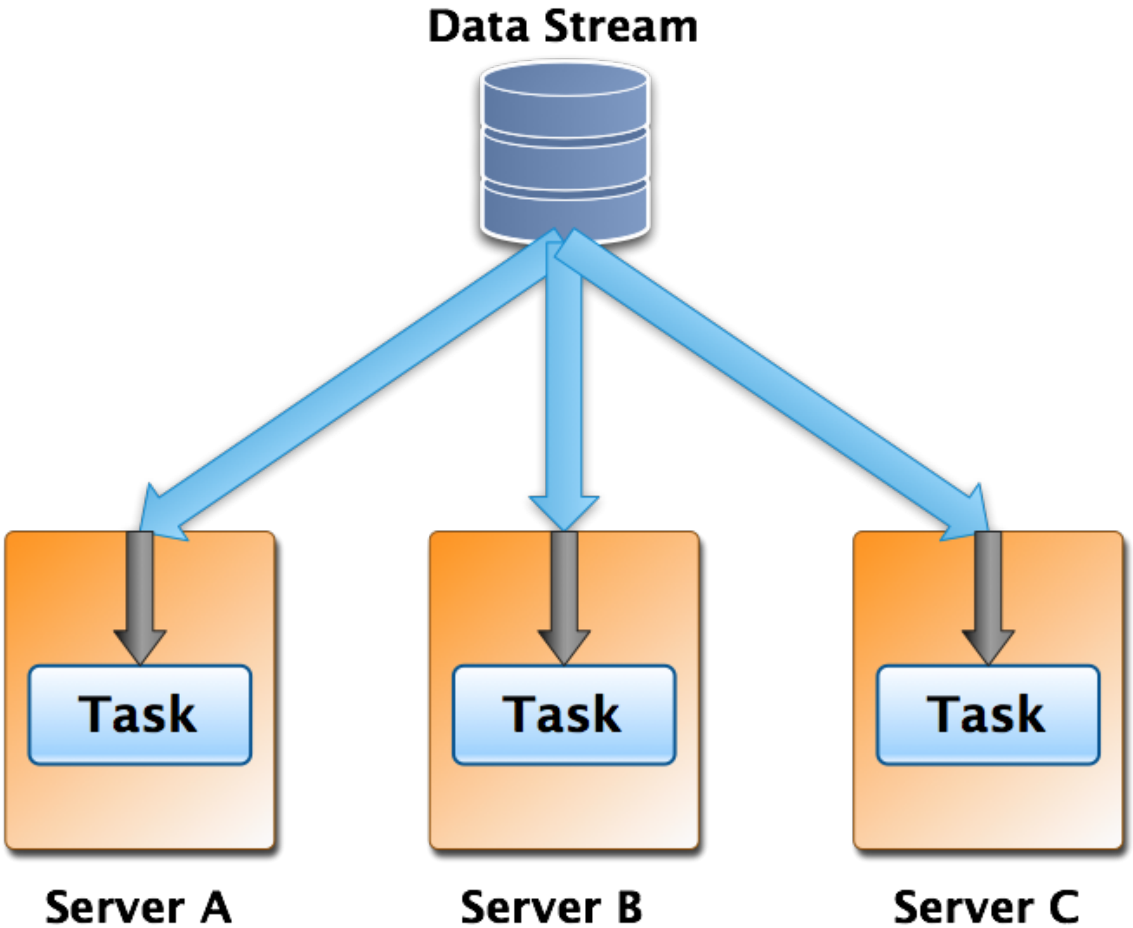

Machine A
Machine C
Task
Task
Task
Task
Source
Sink
Machine B
Task


Machine A
Machine C
Task
Task
Task
Task
Source
Sink
Machine B
Task

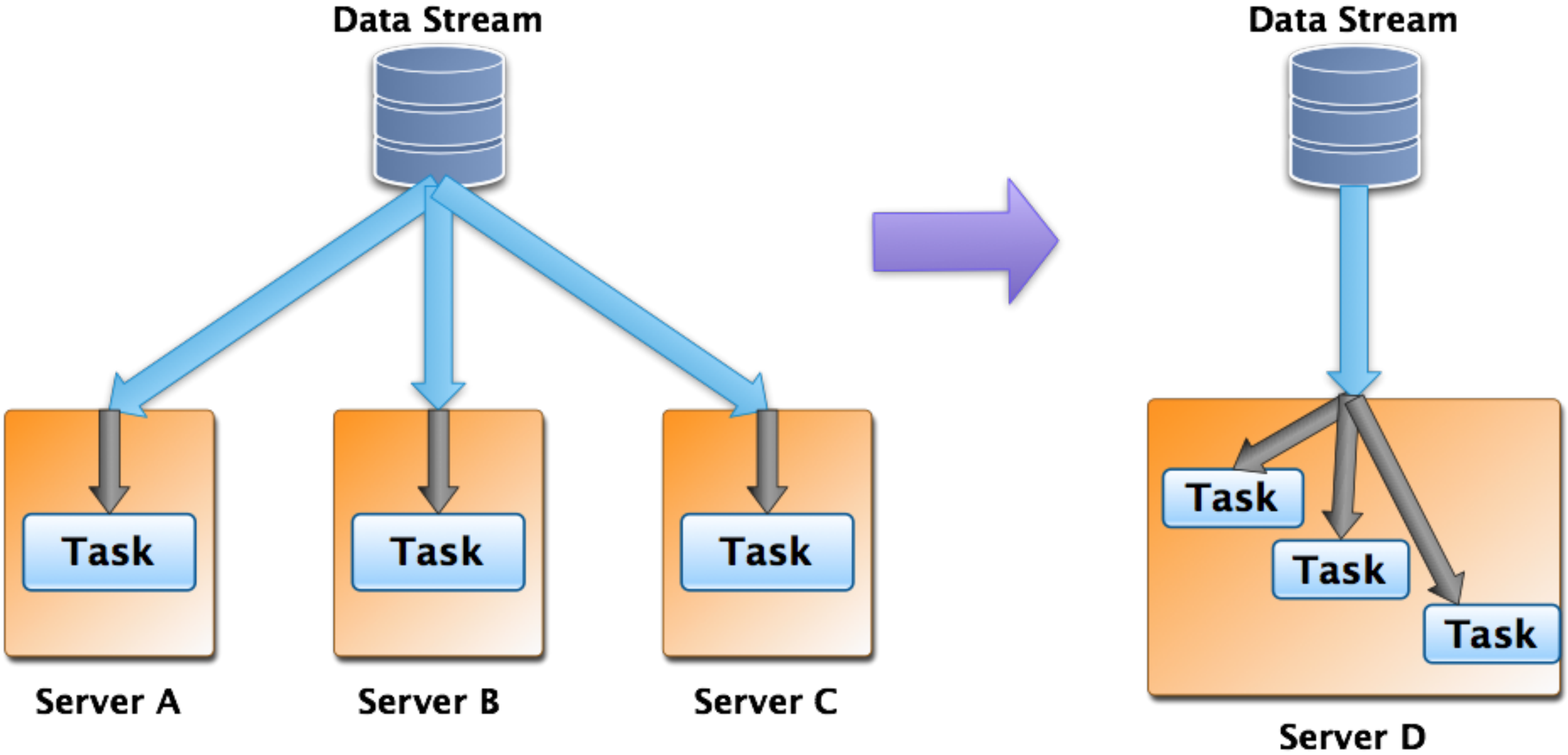

Machine A
Machine C
Task
Task
Task
Sink
Source
Machine B
Machine D
Task
Task
Task

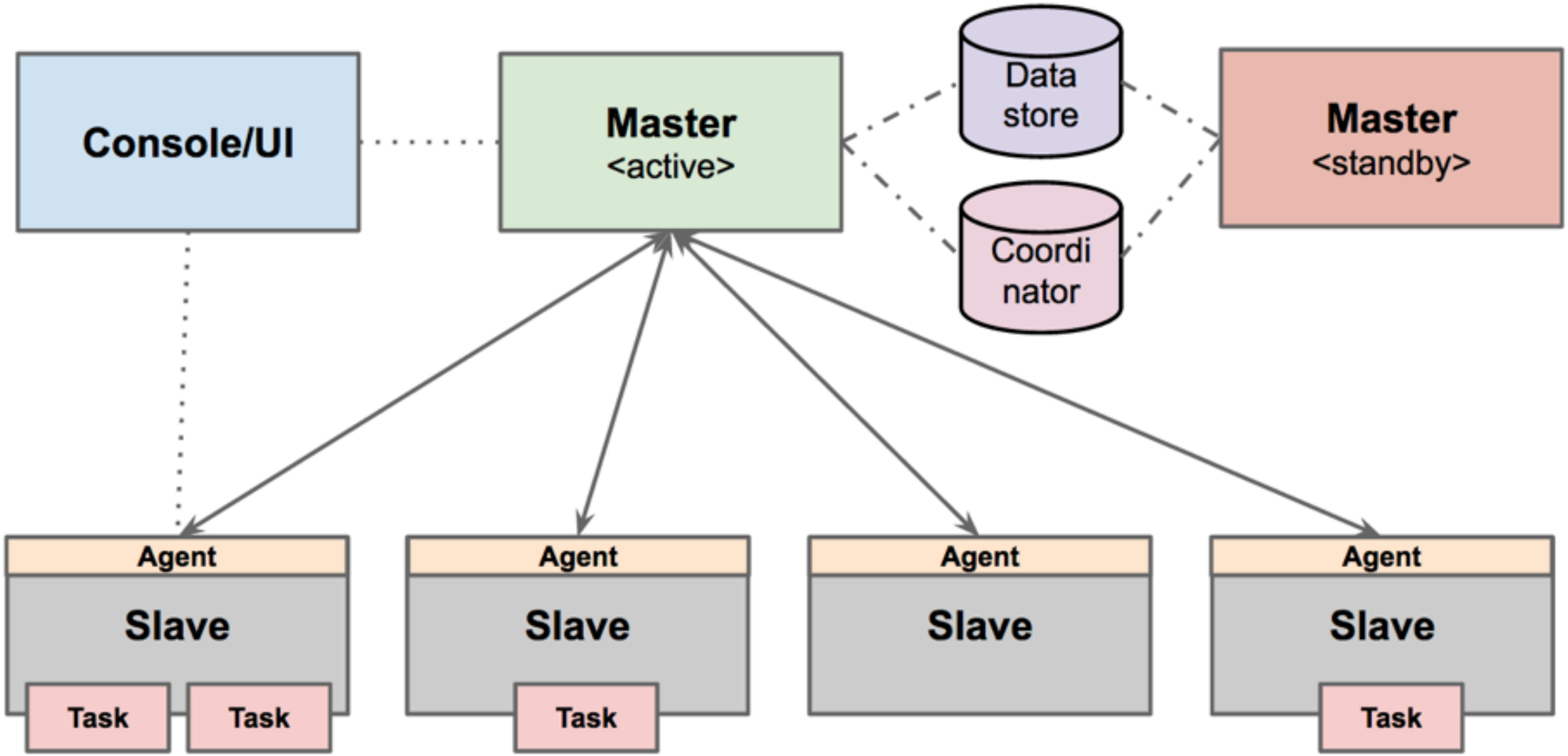
3. Same Source, Multiple Consumers

Solution: Stream Locality

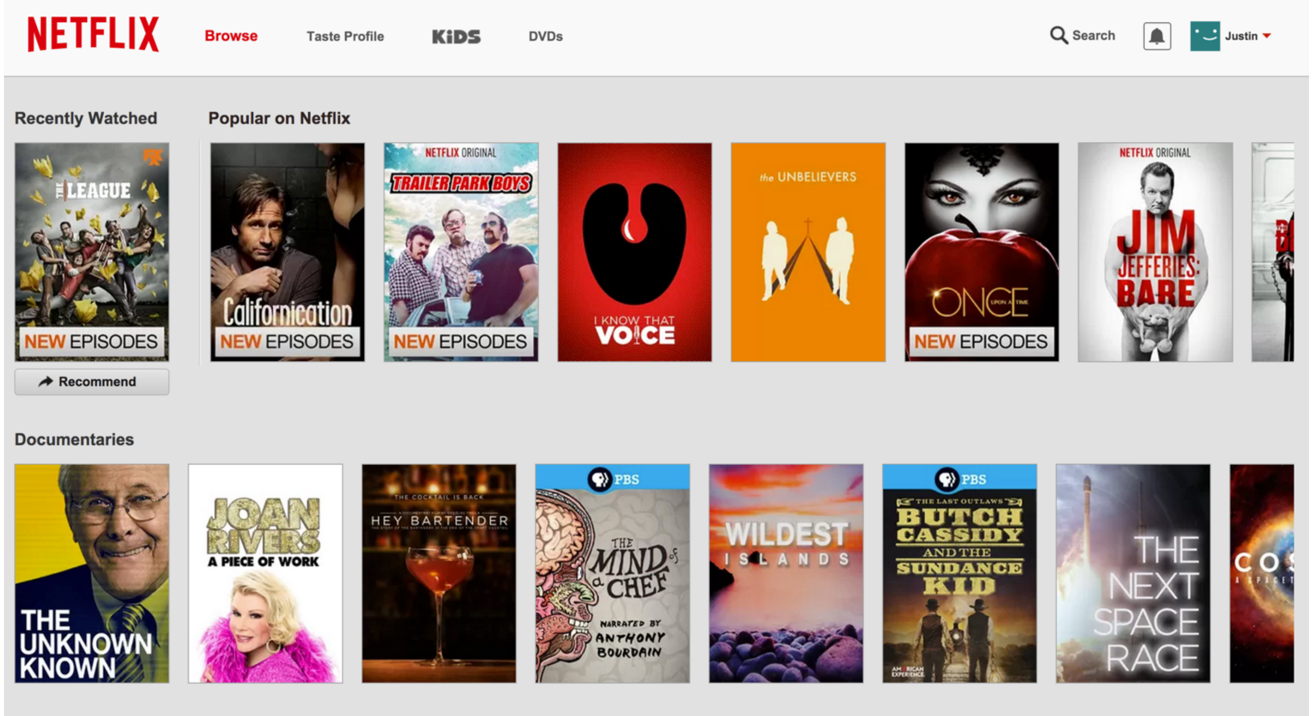


Solution: Stream Locality




Console/UI
Master
<active>
Data
store
Coordi
nator
Master
<standby>
Agent
Slave
Task
Task
Agent
Slave
Task
Agent
Slave
Agent
Slave
Task

We want Internet TV to just work



One problem we need to solve, detect movies that are failing?

Do it fast → limit impact, fix early

Do it at scale → for all permutations

Do it cheap → cost detect <<< serve

Work through the details for how to solve this problem in Mantis

Goal is to highlight unique and interesting design features

… begin with batch approach, the non-Mantis approach

Batch algorithm, runs every N minutes

```
for(play in playAttempts()){
  Stats movieStats = getStats(play.movieId);
  updateStats(movieStats, play);
  if (movieStats.failRatio > THRESHOLD){
    alert(movieId, failRatio, timestamp);
  }
}
```

First problem, each run requires reads + writes to data store per run

Batch algorithm, runs every N minutes

```
for(play in playAttempts()){
  Stats movieStats = getStats(play.movieId);
  updateStats(movieStats, play);
  if (movieStats.failRatio > THRESHOLD){
    alert(movieId, failRatio, timestamp);
  }
}
```

For Mantis don’t want to pay that cost: for latency or storage

Batch algorithm, runs every N minutes

```
for(play in playAttempts()){
  Stats movieStats = getStats(play.movieId);
  updateStats(movieStats, play);
  if (movieStats.failRatio > THRESHOLD){
    alert(movieId, failRatio, timestamp);
  }
}
```

Next problem, “pull” model great for batch processing, bit awkward for stream processing

Batch algorithm, runs every N minutes

```
for(play in playAttempts()){
  Stats movieStats = getStats(play.movieId);
  updateStats(movieStats, play);
  if (movieStats.failRatio > THRESHOLD){
    alert(movieId, failRatio, timestamp);
  }
}
```

By definition, batch processing requires batches. How do I chunk my data? Or, how often do I run?

Batch algorithm, runs every N minutes

```
for(play in playAttempts()){
  Stats movieStats = getStats(play.movieId);
  updateStats(movieStats, play);
  if (movieStats.failRatio > THRESHOLD){
    alert(movieId, failRatio, timestamp);
  }
}
```

For Mantis, prefer “push” model, natural approach to data-in-motion processing

Batch algorithm, runs every N minutes

```
for(play in playAttempts()){
  Stats movieStats = getStats(play.movieId);
  updateStats(movieStats, play);
  if (movieStats.failRatio > THRESHOLD){
    alert(movieId, failRatio, timestamp);
  }
}
```

For our “push” API we decided to use Reactive Extensions (Rx)

Two reasons for choosing Rx: theoretical, practical

1. Observable is a natural abstraction for stream processing, Observable = stream
2. Rx already leveraged throughout the company

NETFLIX ♥

So, what is an Observable?

A sequence of events, aka a stream

```
Observable<String> o =
Observable.just(“hello”,
 “qcon”, “SF”);
o.subscribe(x->{println x;})
```

What can I do with an Observable?

Apply operators → New observable
Subscribe → Observer of data

Operators, familiar lambda functions

```
map(), flatMap(), scan(), ...
```

What is the connection with Mantis?

In Mantis, a job (code-to-run) is the collection of operators applied to a sourced observable where the output is sinked to observers

Think of a “source” observable as the input to a job.

Think of a “sink” observer as the output of the job.

Let’s refactor previous problem using Mantis API terminology

Source:	Play attempts
Operators:	Detection logic
Sink:	Alerting service

Sounds OK, but how will this scale?

For pull model luxury of requesting data at specified rates/time

Analogous to drinking from a straw

In contrast, push is the firehose

No explicit control to limit the flow of data

In Mantis, we solve this problem by scaling horizontally

Horizontal scale is accomplished by arranging operators into logical “stages”, explicitly by job writer or implicitly with fancy tooling (future work)

A stage is a boundary between computations. The boundary may be a network boundary, process boundary, etc.

So, to scale, Mantis job is really,

Source → input observable

Stage(s) → operators

Sink → output observer

Let’s refactor previous problem to follow the Mantis job API structure

```
MantisJob
 .source(Netflix.PlayAttempts())
 .stage({ // detection logic })
 .sink(Alerting.email())
```

We need to provide a computation boundary to scale horizontally

```
MantisJob
 .source(Netflix.PlayAttempts())
 .stage({ // detection logic })
 .sink(Alerting.email())
```

For our problem, scale is a function of the number of movies tracking

```
MantisJob
.source(Netflix.PlayAttempts())
.stage({ // detection logic })
.sink(Alerting.email())
```

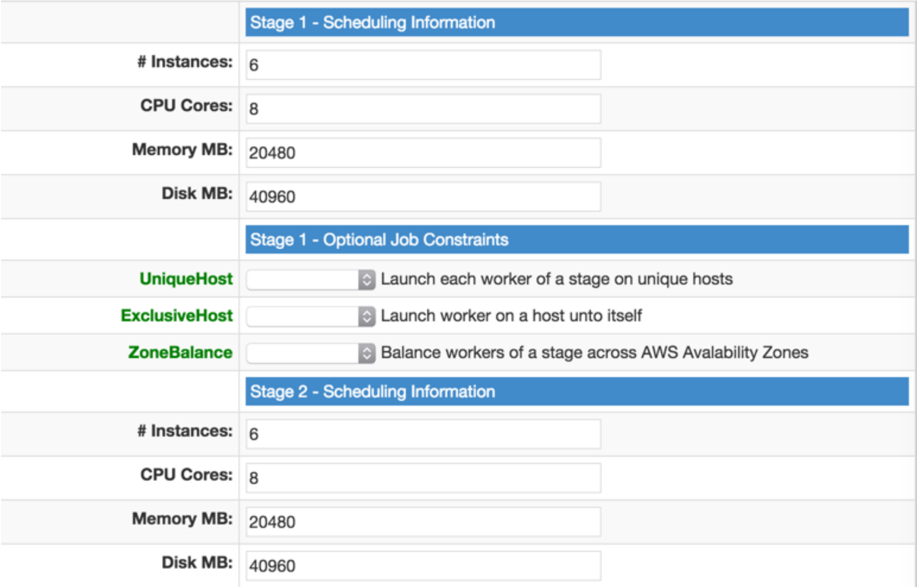
Lets create two stages, one producing groups of movies, other to run detection

```
MantisJob
 .source(Netflix.PlayAttempts())
 .stage({ // groupBy movieId })
 .stage({ // detection logic })
 .sink(Alerting.email())
```

OK, computation logic is split, how is the code scheduled to resources for execution?

```
MantisJob
.source(Netflix.PlayAttempts())
.stage({ // groupBy movieId })
.stage({ // detection logic })
.sink(Alerting.email())
```

One, you tell the Mantis Scheduler explicitly at submit time: number of instances, CPU cores, memory, disk, network, per instance

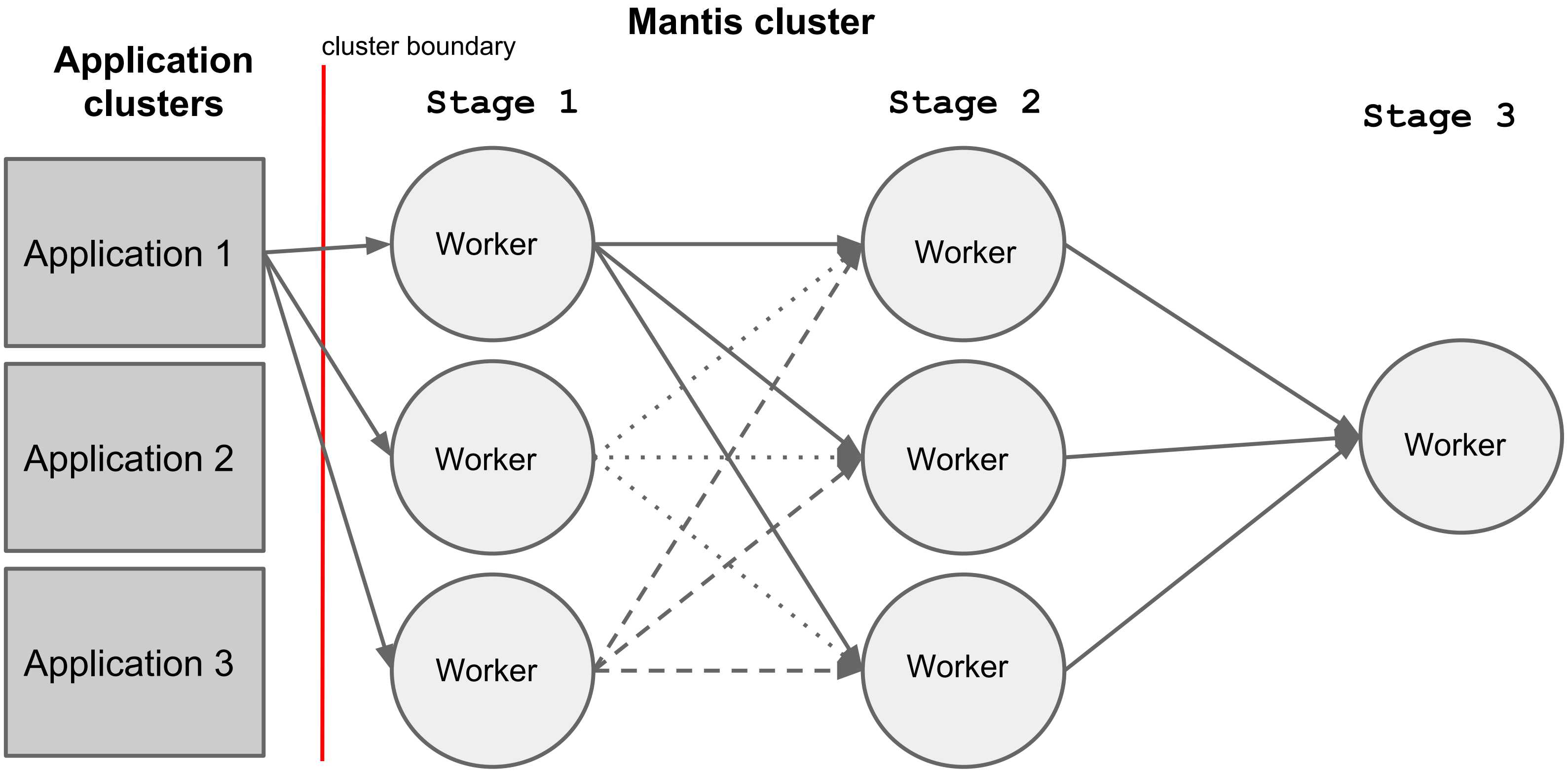


	Stage 1 - Scheduling Information
# Instances:	6
CPU Cores:	8
Memory MB:	20480
Disk MB:	40960
	Stage 1 - Optional Job Constraints
UniqueHost	Launch each worker of a stage on unique hosts
ExclusiveHost	Launch worker on a host unto itself
ZoneBalance	Balance workers of a stage across AWS Avalability Zones
	Stage 2 - Scheduling Information
# Instances:	6
CPU Cores:	8
Memory MB:	20480
Disk MB:	40960

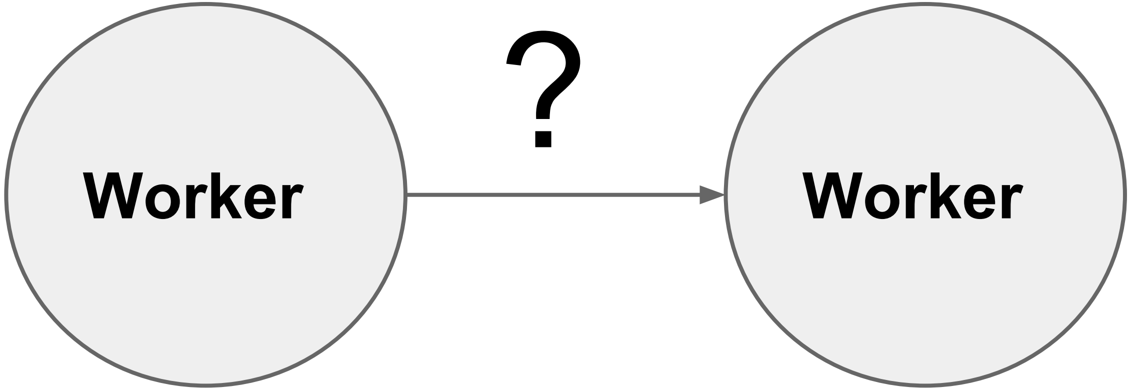
Two, Mantis Scheduler learns how to schedule job (work in progress)

Looks at previous run history
Looks at history for source input
Over/under provision, auto adjust

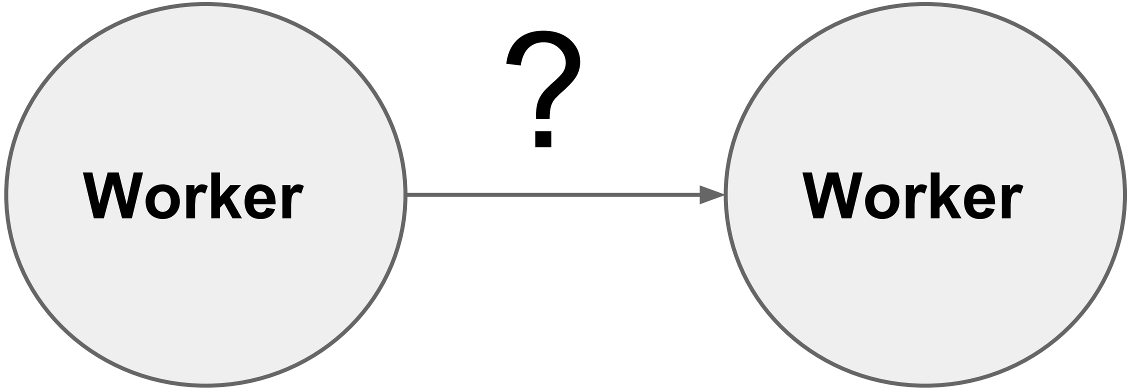
A scheduled job creates a topology



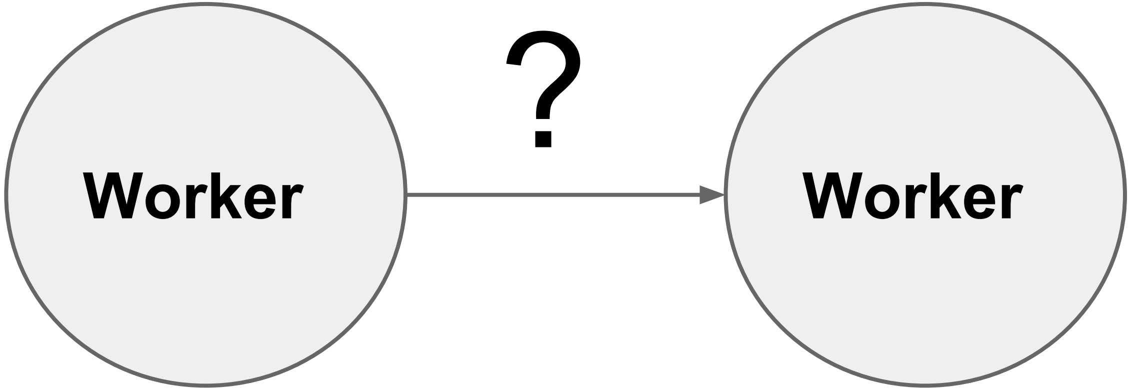
Computation is split, code is scheduled, how is data transmitted over stage boundary?



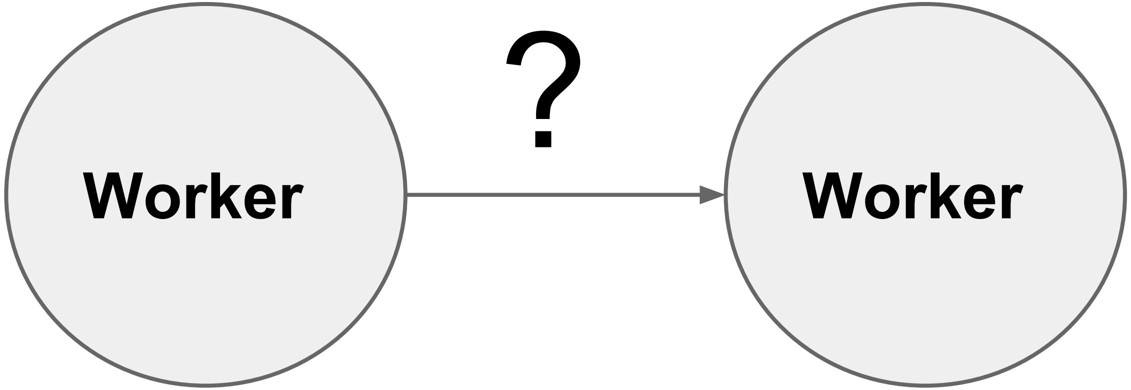
Depends on the service level agreement (SLA) for the Job, transport is pluggable



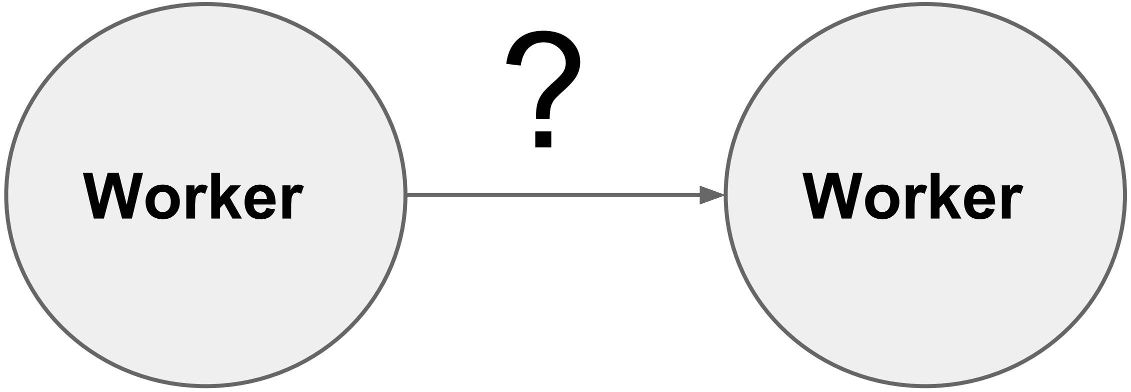
Decision is usually a trade-off between latency and fault tolerance



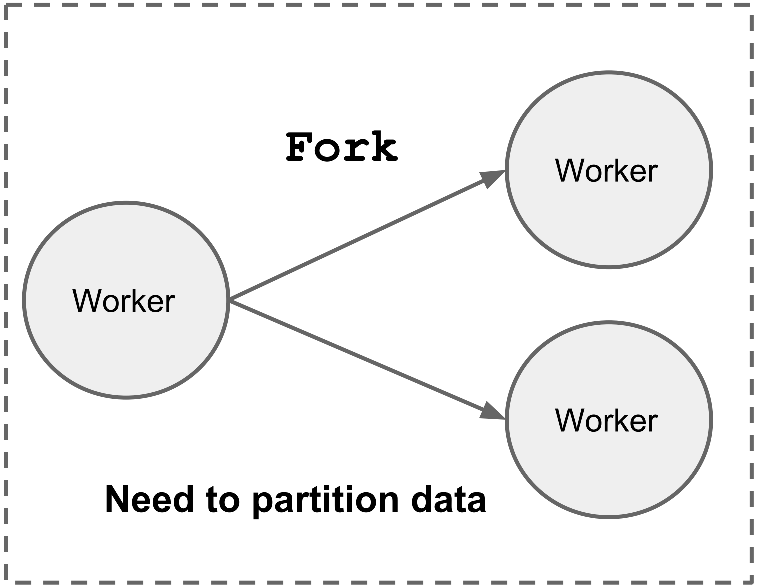
A “weak” SLA job might trade-off fault tolerance for speed, using TCP as the transport



A “strong” SLA job might trade-off speed for fault tolerance, using a queue/broker as a transport



Forks and joins require data partitioning, collecting over boundaries



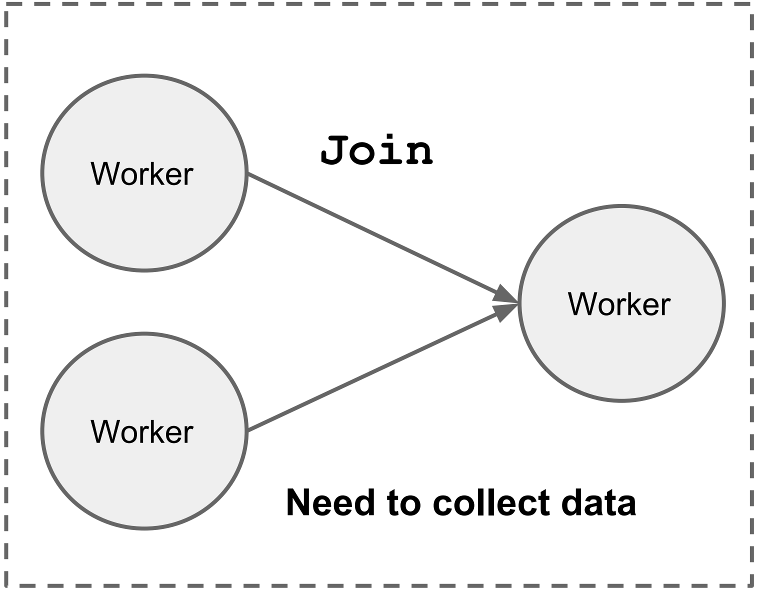
Fork

Worker

Worker

Worker

Need to partition data



Join

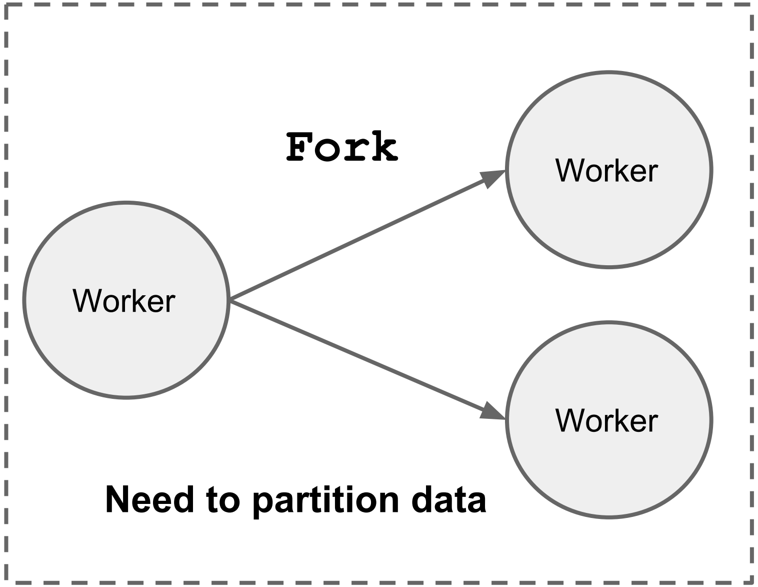
Worker

Worker

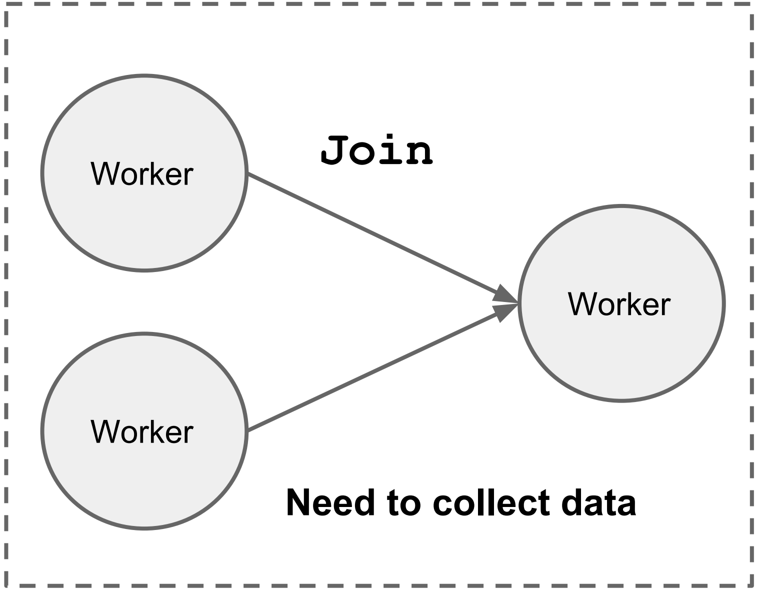
Worker

Need to collect data

Mantis has native support for partitioning, collecting over scalars (T) and groups (K,V)



Need to partition data



Need to collect data

Let’s refactor job to include SLA, for the detection use case we prefer low latency

```
MantisJob
 .source(Netflix.PlayAttempts())
 .stage({ // groupBy movieId })
 .stage({ // detection logic })
 .sink(Alerting.email())
 .config(SLA.weak())
```

The job is scheduled and running what happens when the input-data’s volume changes?

Previous scheduling decision may not hold

Prefer not to over provision, goal is for cost insights <<< product

Good news, Mantis Scheduler has the ability to grow and shrink (autoscale) the cluster and jobs

Google Cloud Platform

amazon web services™

The cluster can scale up/down for two reasons: more/less job (demand) or jobs themselves are growing/shrinking

For cluster we can use submit pending queue depth as a proxy for demand

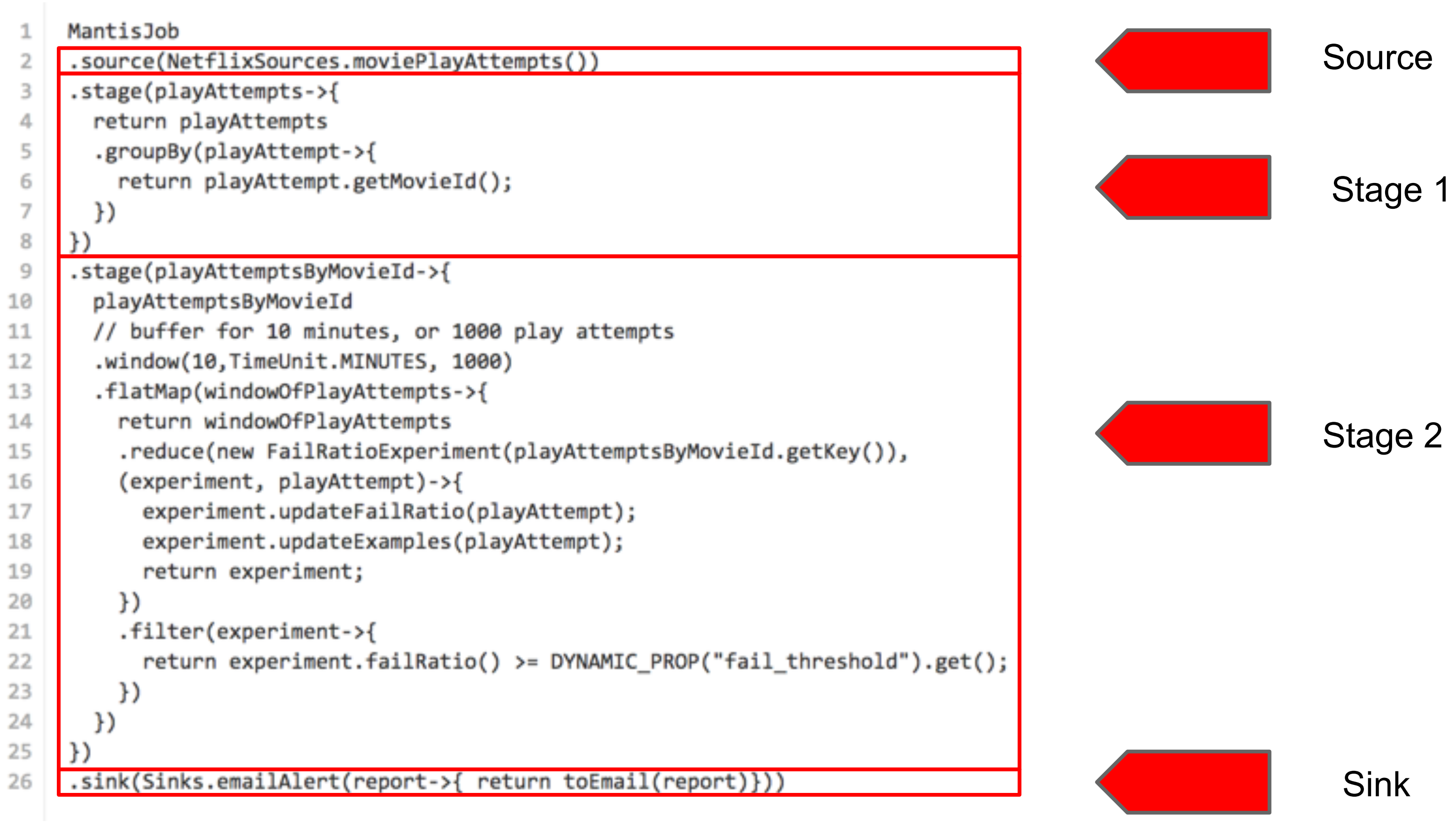
For jobs we use backpressure as a proxy to grow shrink the job

Backpressure is “build up” in a system

Imagine we have a two stage Mantis job,
Second stage is performing a complex calculation, causing data to queue up in the previous stage

We can use this signal to increase nodes at second stage

Having touched on key points in Mantis architecture, want to show a complete job definition



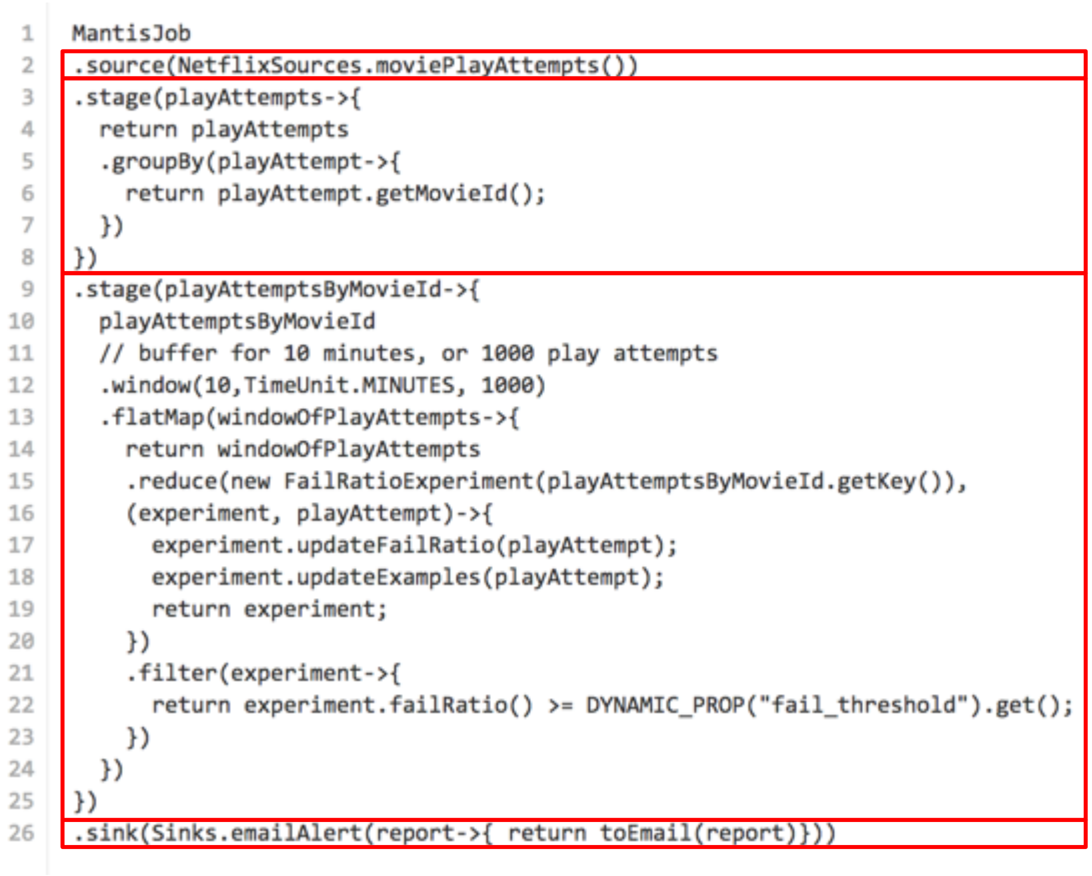
```
MantisJob
.source(NetflixSources.moviePlayAttempts())
.stage(playAttempts->{
  return playAttempts
  .groupBy(playAttempt->{
    return playAttempt.getMovieId();
  })
})
.stage(playAttemptsByMovieId->{
  playAttemptsByMovieId
  // buffer for 10 minutes, or 1000 play attempts
  .window(10,TimeUnit.MINUTES, 1000)
  .flatMap(windowOfPlayAttempts->{
    return windowOfPlayAttempts
    .reduce(new FailRatioExperiment(playAttemptsByMovieId.getKey()),
    (experiment, playAttempt)->{
      experiment.updateFailRatio(playAttempt);
      experiment.updateExamples(playAttempt);
      return experiment;
    })
    .filter(experiment->{
      return experiment.failRatio() >= DYNAMIC_PROP("fail_threshold").get();
    })
  })
})
.sink(Sinks.emailAlert(report->{ return toEmail(report)}))
```

Source

Stage 1

Stage 2

Sink



```
MantisJob
.source(NetflixSources.moviePlayAttempts())
.stage(playAttempts->{
  return playAttempts
  .groupBy(playAttempt->{
    return playAttempt.getMovieId();
  })
})
.stage(playAttemptsByMovieId->{
  playAttemptsByMovieId
  // buffer for 10 minutes, or 1000 play attempts
  .window(10,TimeUnit.MINUTES, 1000)
  .flatMap(windowOfPlayAttempts->{
    return windowOfPlayAttempts
    .reduce(new FailRatioExperiment(playAttemptsByMovieId.getKey()),
    (experiment, playAttempt)->{
      experiment.updateFailRatio(playAttempt);
      experiment.updateExamples(playAttempt);
      return experiment;
    })
    .filter(experiment->{
      return experiment.failRatio() >= DYNAMIC_PROP("fail_threshold").get();
    })
  })
})
.sink(Sinks.emailAlert(report->{ return toEmail(report)}))
```

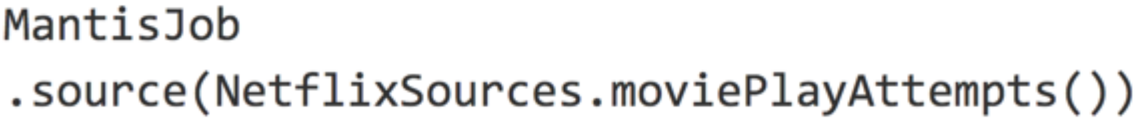
Play Attempts

Grouping by movie Id

Detection algorithm

Email alert

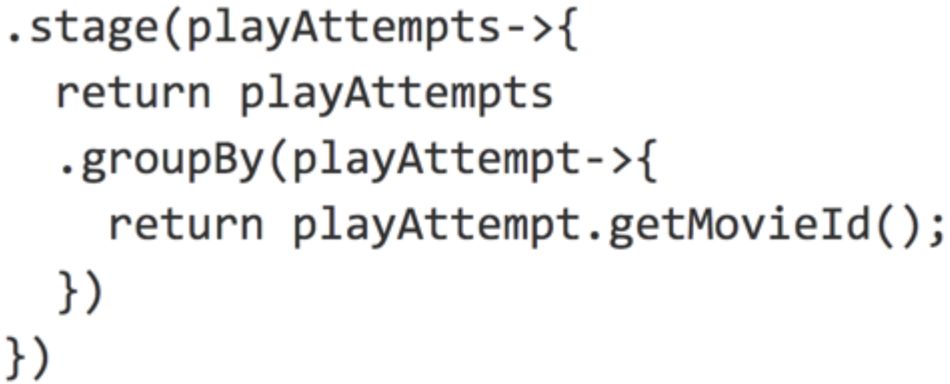
Sourcing play attempts



```
MantisJob
.source(NetflixSources.moviePlayAttempts())
```

Static set of sources

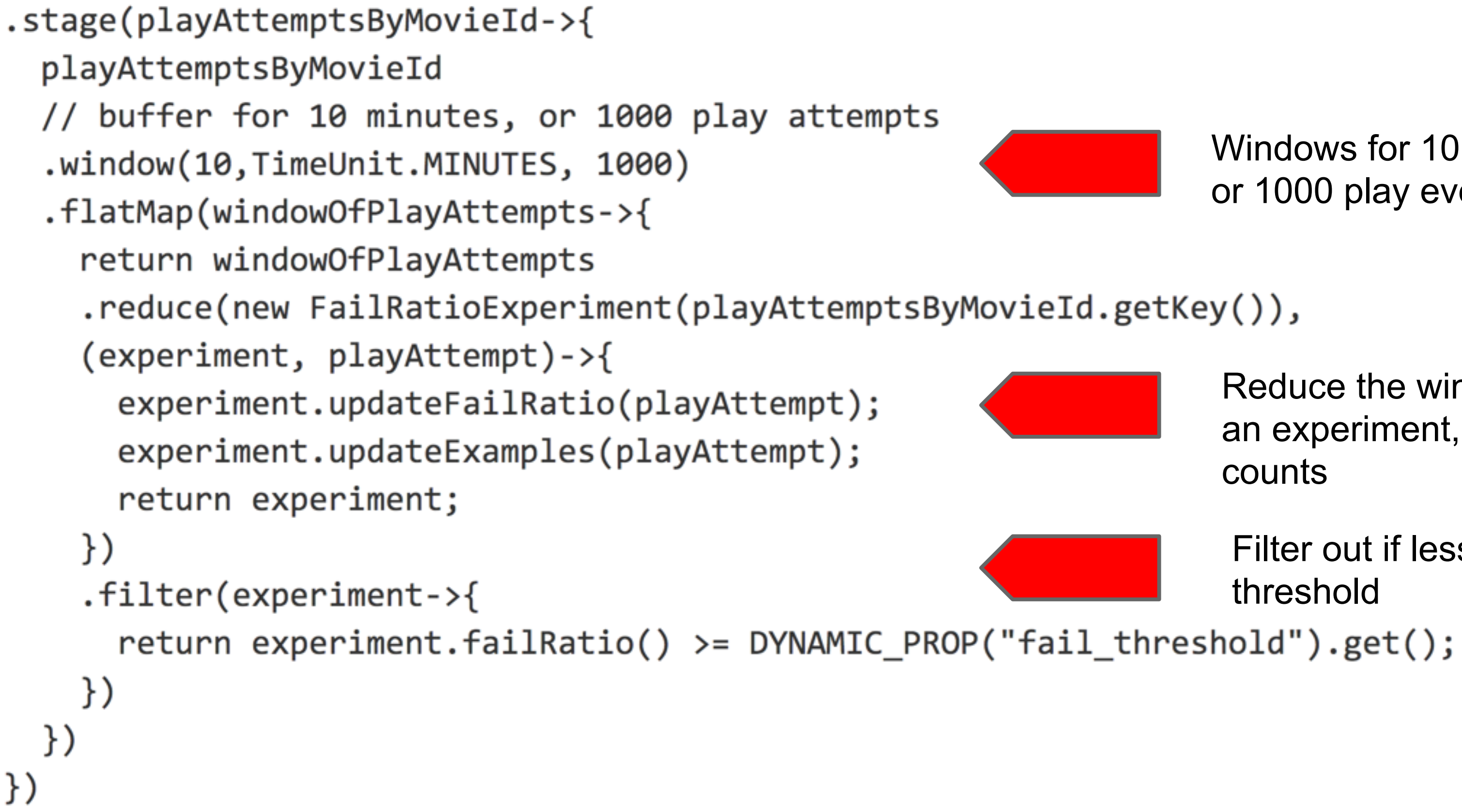
Grouping by movie Id



```
.stage(playAttempts->{
  return playAttempts
  .groupBy(playAttempt->{
    return playAttempt.getMovieId();
  })
})
```

GroupBy operator returns key selector function

Simple detection algorithms



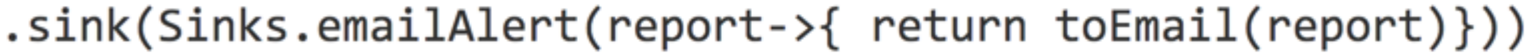
```
.stage(playAttemptsByMovieId->{
  playAttemptsByMovieId
  // buffer for 10 minutes, or 1000 play attempts
  .window(10,TimeUnit.MINUTES, 1000)
  .flatMap(windowOfPlayAttempts->{
    return windowOfPlayAttempts
    .reduce(new FailRatioExperiment(playAttemptsByMovieId.getKey()),
    (experiment, playAttempt)->{
      experiment.updateFailRatio(playAttempt);
      experiment.updateExamples(playAttempt);
      return experiment;
    })
    .filter(experiment->{
      return experiment.failRatio() >= DYNAMIC_PROP("fail_threshold").get();
    })
  })
})
```

Windows for 10 minutes or 1000 play events

Reduce the window to an experiment, update counts

Filter out if less than threshold

Sink alerts



```
.sink(Sinks.emailAlert(report->{ return toEmail(report)}))
```