Back to Tech Blog >

Jul 27, 2018

Unicorn - Rheos remediation center

By:
Lubin Liu

Comments:
0

Social:

TOPICS: Cloud, Data Infrastructure and Services

Rheos is the near-line data platform of eBay and it owns thousands of stateful machines on cloud. This article explains how Unicorn helps Rheos to remediate the system in an automatic way. Unicorn guarantees a better SLA and saves human support efforts.

Motivation

Rheos is the near-line data platform of eBay and it owns thousands of machines on cloud. A group of streaming related applications are running in Rheos, like Kafka and Storm.

Managing thousands of stateful machines on cloud is very challenging. The operation tasks come from two categories:

- Hardware failures on cloud happen everyday
- Stateful application specific issues when scale grows

Automation is the key to save efforts and provide a better SLA. Rheos team kept to build and enhance the automation system in the past two years. However, with new problems come out frequently and monitoring system upgraded, it's time to unify the past work and build a modern remediation system. To be more specific, the remediation system wanna to cover the following three points:

A centralized place to manage operation jobs

Previously, Rheos has tools to remediate clusters. Many tools are scripts and run in separate places. Such kind of dispersive tool based remediation has several limitations:

- Hard to develop and maintain
- Conflicts between tools
- Learning curve for new support candidates
- Not truly automation, cost human efforts

Handle alert more efficiently

Rheos already delivered a bunch of metrics and alerts. If we can collect these input in a centralized place and associate with other external information, a rule based remediation service could heal the clusters automatically.

Handle the alert with automation flow will:

- Reduce human efforts
- Reduce the SLA to handle alerts

Store alert and remediation history for further analysis

Automation can't cover all the issue at the very beginning and may also introduce new issues if the algorithm is not good enough. Storing the history in a centralized place could help team to improve the system.

On the other hand, by analysing the alert history, team can try to find new automation scope.

Challenge

Building such a centralized remediation center is not easy. Two points need to be highlighted.

A generic model to cover known experiences and easy to extend

Build a specific tool to solve a specific problem is a very straightforward solution. On the contrary, identify a common pattern and model to apply all the existing experience is much harder. To avoid build another group of tools in the future, the pattern proposed also need the ability to extend to solve new issues in the future.

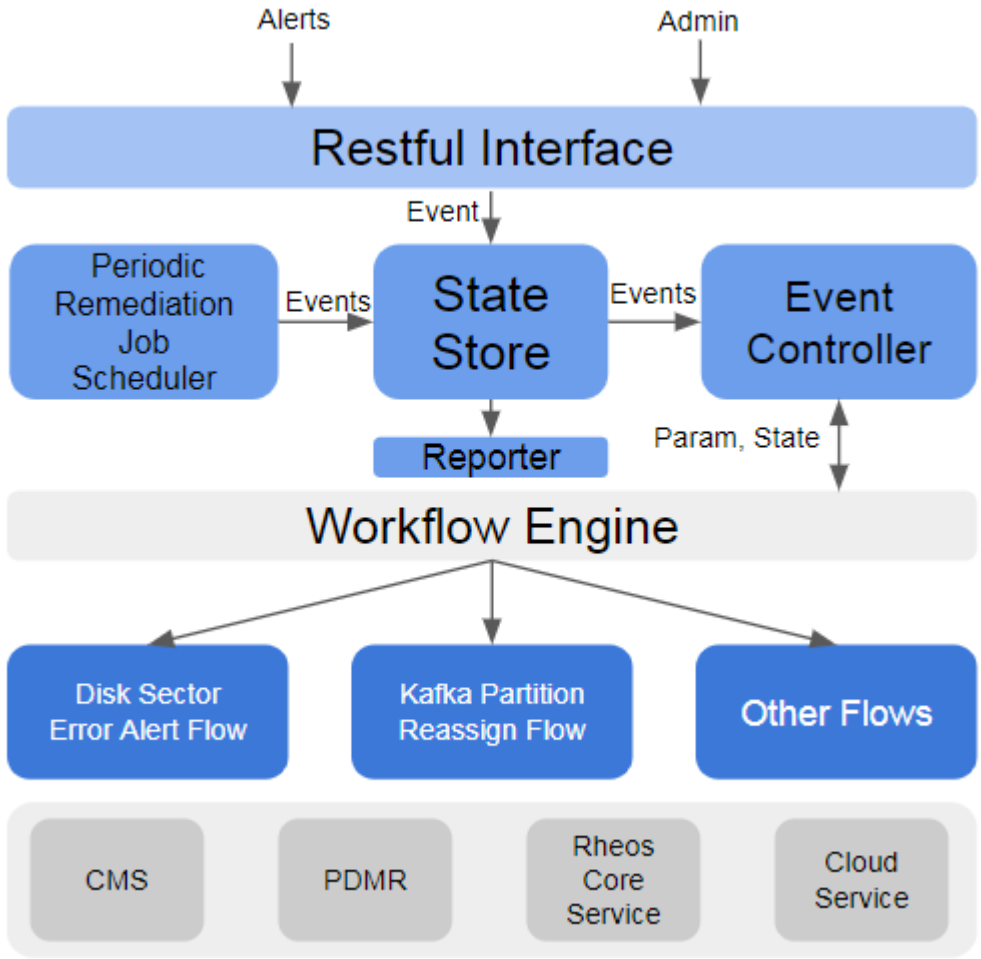
Avoid over automation with good efficiency

Automation sometimes is dangerous from two points of view:

- Do something not allowed: the actions that done by an automation system must be limited and under control. Do anything that unexpected may make things worse.
- Do something too quickly: the concurrence of automation system must be limited, especially for stateful cluster. It needs time to sync state back after operation and easier to rollback when realize bugs in automation system.

Serialize all the tasks may avoid part of the over automation issues. But it's better to define a model to guarantee efficiency. In a big complicated system, handling tasks in a more efficient way offers a better SLA.

Design



Overview

The above picture shows the overview design of Unicorn.

Unicorn defines a key resource called “Event” to abstract all the issues it needs to solve. Event has three sources:

- Alerts send out from upstream: this is the most common source
- Manually dropped by admin: a way to manually operate in urgent case
- Periodic remediation job: some routine check tasks

Event controller reads the events periodically and trigger workflow to handle events. This module needs to consider how to avoid over automation and guarantee efficiency.

Workflow engine handled the workflow life cycle and send back the event status to event controller.

In each workflow, it may invoke some external dependency services, like CMS and underlying PDMR system to do the remediation logics.

Reporter module sends out summary/detail report to subscribed targets.

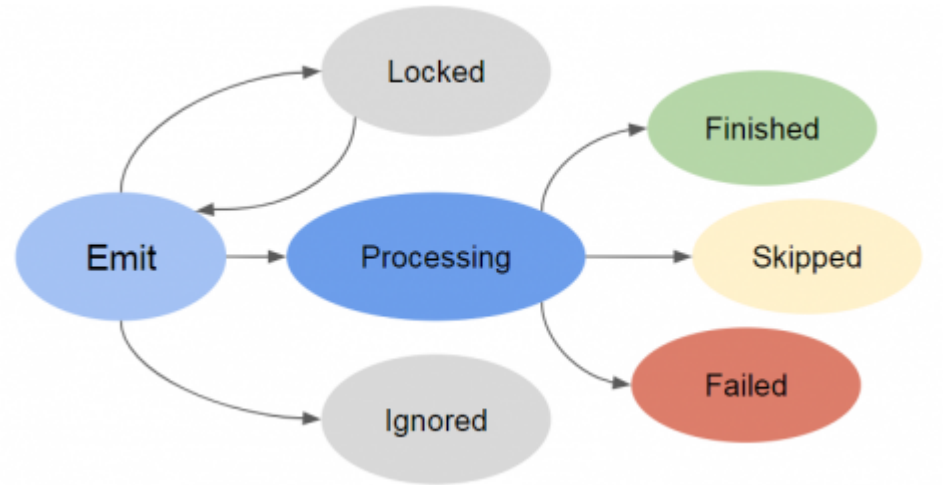
Event

Model

Name	Type	Description
ID	Long	Unique id to index each event
TYPE	String	The issue type that this event need to handle
GROUP_ID	String	The event in the same group need to be handled sequentially
STATUS	Enum	Status life cycle of the event

LABELS	Map	Information to describe the issue
PAYLOAD	String	Information output during the processing
PRIORITY	Int	How urgent of this event, bigger value means more urgent
FLOW_ID	Long	The id of the workflow to handle this event
TIMESTAMP	Long	The timestamp of this event to become valid. Maybe a future time for a delay event.
TIME_TO_LIVE_MS	Long	The lifetime of this event after become valid
OWNER	String	The source of this event
RETRY_COUNT	Int	How many times has been tried to handle this event
PROCESS_TIMESTAMP	Long	The timestamp that workflow begin to handle the event
REFERENCE_ID	String	The id that defined by user for querying
LOG	String	The processing log

State Machine



The state of each event defines the file cycle of it.

- Emit: Event first dropped in the state store
- Processing: Event is picked by the event controller and handling by the workflow engine
- Locked: Another event in the same group is processing and marked the remaining as locked
- Finished: Event has been processed successfully
- Skipped: Event has been processed, but for some reason, it has reached the final state. Usually means doesn't need to take any further action
- Failed: Event has been processed, but failed. Need to take a look APSP
- Ignored: Event is not considered in current scope

Event Controller

GroupId

Unicorn introduces the concept of "GroupID" to achieve the isolation between events. The semantic of GroupId is as follows:

- The events in the same group must be handled sequentially
- The events in different group could be handled concurrently

Even this field is totally open to self-define, the common case is to use physical cluster id. From the past operation experience, it is safer to avoid handle multiple nodes in one stateful cluster.

Based on GroupId field, event controller works as follows to pick events in each round:

- List all the group
- Iterate all the groups, check whether there is already one event in this group in "Processing" state

- If yes, skip the current group. If no, pick one event and kick off processing
- Wait for next round

Priority

The next question to answer is how to determine which group/event should be picked first. That's the reason Unicorn introduces the concept of "Priority". The semantic of priority is as follows:

- The group with higher priority event will always be scanned first
- The event with higher priority in one group will always be picked first

The priority is related to event type, i.e., the same type of issue has the same priority. The priority of each event type is not defined in Unicorn. When introduces a new event type, the user must define this field.

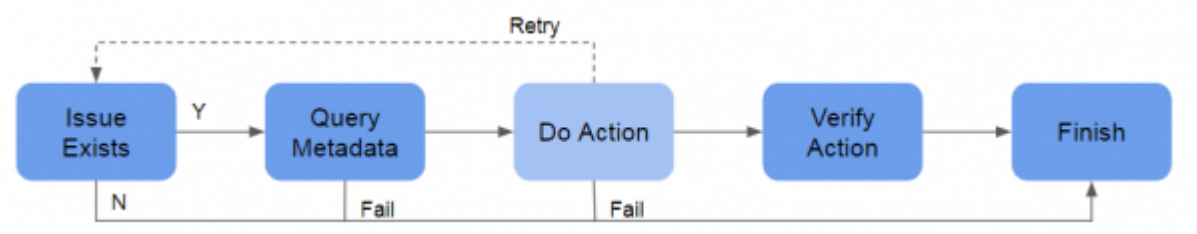
Rate Control

To avoid over automation, Unicorn introduces several concepts to control the rate:

- Max processors: The max concurrent events in "Processing" state. When event controller find already reached this threshold, even there still are groups need to scan, just skip. "Vip priority threshold" could break this rule.

- Rate control window: The max event count could be handled within some sliding window for this specific event type.
- Vip priority threshold: The group whose highest priority event exceed this threshold could break the "Max Processors" limitation, i.e., even the concurrent processing events already reach the max processors, the high priority event could also be processed. Note that this rule never break the "GroupId" limitation.

Workflow



The detail workflow defers between event types. However, most of the workflow in Unicorn follows the above common pattern:

- Issue exists stage: check whether this issue still exists. This stage could help to avoid do some action duplicated.
- Query metadata stage: query metadata based on input. Most of the case, input only contains the index info, like cluster id, node id ect and for detail information, need to be queried in the workflow.
- Do action stage: this stage is event type specific and when need to retry, should return to "Issue exists stage".
- Verify action stage: the implementation of this stage may also event type specific, but it is a good pattern to verify what has been done in this flow.
- Finish stage: update the status and sync state to event.

Implementation

NAP(NuData automation platform)

NAP is the restful framework contributed by NuData team. Unicorn highly relies on the features provided by NAP.

Workflow in Unicorn

Unicorn workflows leverage NAP workflow engine. Basically, Unicorn implements separate tasks and build workflows through configuration files.

Each task may fail with many reasons, which will create multiple branches in a workflow diagram, all the tasks in Unicorn have an output variable called “flowStatus”. “flowStatus” describes the result of the current task and Unicorn leverage the “SwitchBy” semantic in NAP workflow engine to determine what should be the next task. For the detail info of the current task, Unicorn output to the “log” field of the event.

Besides basic workflow, Unicorn has a lot of “Scheduled workflow”. The scheduler in NAP is much powerful than the ScheduledExecutorService in Java and even support crontab style. This feature is very useful for Unicorn to control daily and weekly jobs.

Restful service in Unicorn

Unicorn restful services all leverage NAP micro-service restful framework. The services in Unicorn could be divided into two categories:

Resource CRUD

Two main resources in Unicorn: event and alert. Event is the key resource in Unicorn and Unicorn manages the whole life cycle of it, including create, get, update and delete. Alert interface is opened for alerting system to inject through webhook, and the main logic in AlertHandler is to transfer alert to standard event.

Note that in current implementation, the GroupId of event, who is transformed from alert, is also injected in the AlertHandler. The GroupId Unicorn choosed is the clusterId.

As an automation system, Unicorn also expose an API to query the clusters that recently handled. This is useful for support guy as well as generate reports.

Workflow resources also fall into this type, but NAP framework already covered this part.

Admin API

This group of API helps admin to debug issues. NAP has two amazing interface:

- Thread dump: return the thread dump info of the current service. Useful for deadlock analysis and thread hang issue debugging.
- Logs: streaming the log file content. Useful for detail debug.

Portal in Unicorn

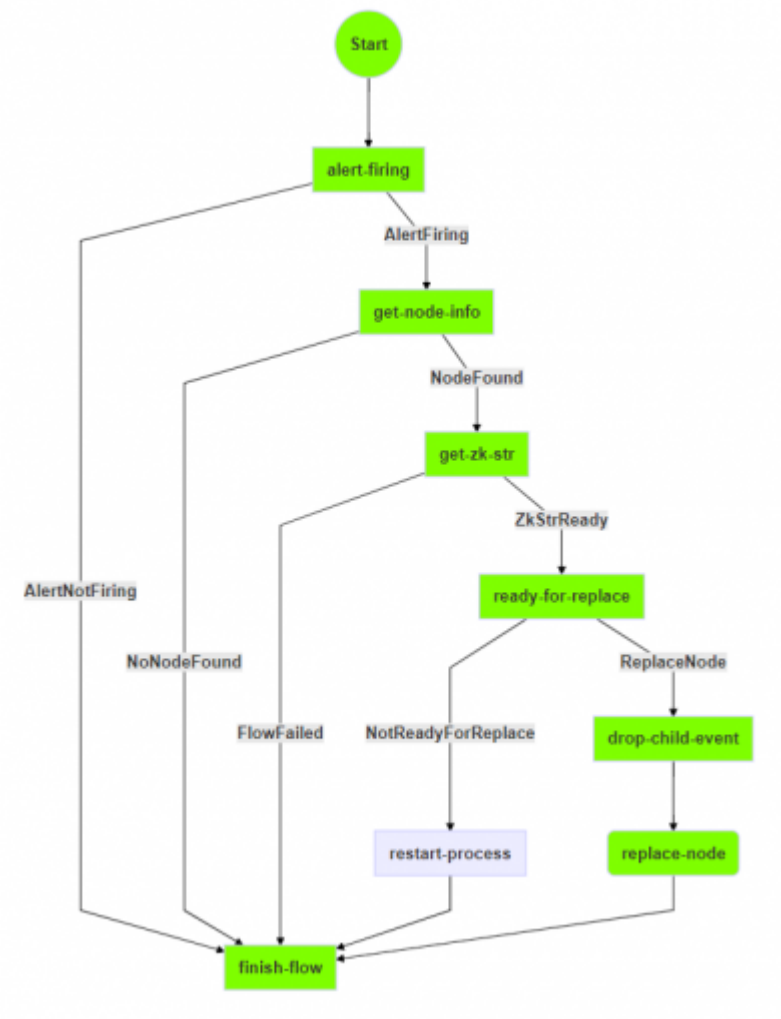
Unicorn portal is totally leverage NAP config driven UI framework. The portal of Unicorn is very standard operation usage which focus on resource query and status update.

To show the overall status of the system, on the main page of Unicorn shows some statistical information. For example:

- The total event type supported currently
- The count of clusters handled recently
- How many events are handled successfully recently
- How many events are handled failed recently

Sample workflows

Disk sector error flow



Kafka depends on the stability of disk a lot. An HDD with high medie sector error will impact the Kafka application and cause the following two main issues:

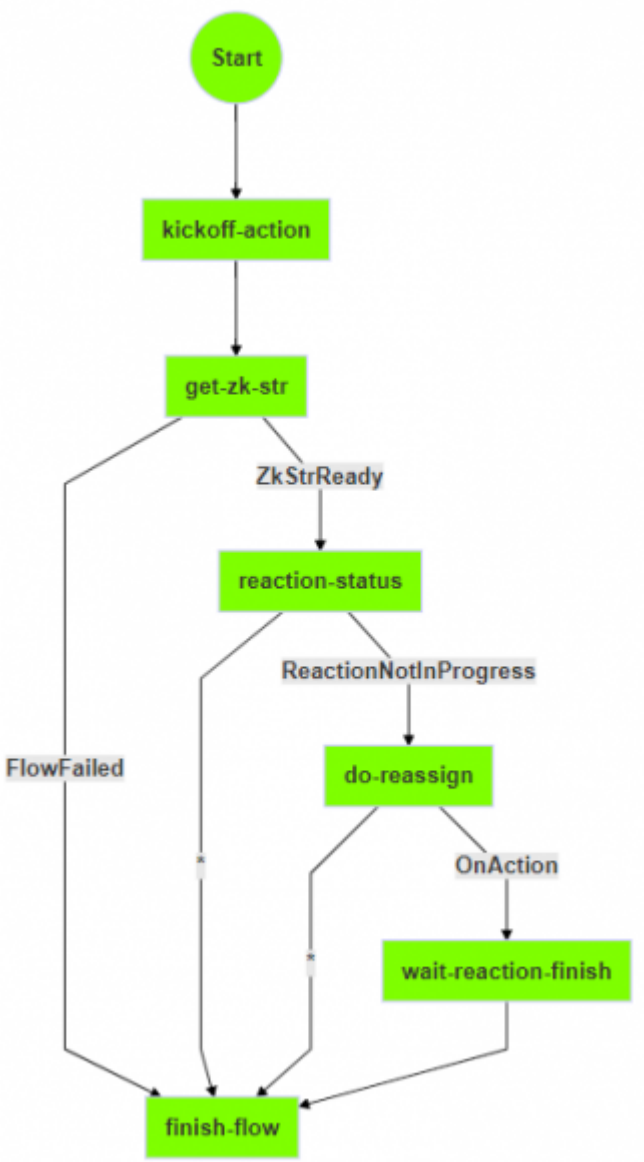
- Traffic drop dramatically
- High end 2 end latency

In Unicorn, when receive a medie sector error alert, the flow is shown on the above image. Two steps should be highlighted:

- Alert firing: check back the monitoring system to see whether this issue still exists.
- Ready for replace: remove the old node and bring back a new node with the same broker id is the best solution to solve this issue. However, when the ISR is not in health state, it is dangerous to replace node directly. In this case, Unicorn will only restart the kafka process to temporary slow down the impact.

This flow is a standard issue caused by the underlying hardware failure.

Kafka partition reassign flow



Each Kafka broker holds some number of topic-partition. When the traffic in one cluster increase dramatically, Rheos will do the flexup. However, Kafka will not move the existing topic-partition to the new brokers automatically. Guarantee the balance of traffic keeps the cluster stable and gets better throughput.

The “PartitionReassignEvent” is dropped periodically. Unicorn remediate the traffic imbalance issue with a greedy algorithm. The main process of this event workflow is as follows:

- Pick one broker with the most partitions
- Pick one broker with the least partitions
- Generate a partition reassign plan and kick off the process with partition reassign tool provided by kafka
- Wait and check until the reassign finishes

This flow is to solve a standard issue caused by application specific design.

Reporter

It’s important to know what operations have been done by an automation system. Send out an operation report periodically is the most common cases in Rheos.

In Unicorn, the reporter is implemented with the scheduled workflow provided by NAP. Most of the the reports it used today is related to event statistical information. To serve difference target receiver, Unicorn make the reporter pluggable with the following configurations:

- Handler: the class that implement the reporter logic
- SchedulerPattern: the pattern to send out the report, in a crontab style
- ReportWindowHour: the time window of this report
- EventTypeSubscribed: the event type list that the report will include
- Receiver: the receiver list that will receive this report
- Type: the protocol of this report to send out, currently, only Email is supported

The developer and boss may need different granularity and different dimension reports. Making this module pluggable provides such kind of capability.

On Production

Unicorn has run on production for several months. This section introduces the statistical information based on production data.

Event types

Currently, 10 event types are supported on production. A brief description and the percentage of each event type in the total event count are as follows:

Event Type	Percentage	Description
MedieErrorDisk	0.06%	Remediate the node with high medie sector error
RollingRestart	0.02%	Restart the process one by one and guarantee state synced
NodeDown	66.89%	Reboot node or replace to bring down virtual machine back
Replace	0.37%	Delete the legacy node and set up a new one with same id and configurations
BehaviorMmLagHighFor5Min	20.00%	Remediate the high mirrormaker lag caused by network shake
PartitionReassign	4.79%	Reassign the partition distribution within one cluster
LeaderReelection	4.78%	Reelect a proper leader for some specific topic partition
ReadonlyDisk	0.31%	Remediate the node when the disk is in read-only state
HighDiskUsageFor5Min	0.09%	Clean up the disk usage when the occupation is high
StraasAgentHeal	1.76%	Auto heal the PRMR system agent

Several observations based on the above table:

- Virtual machine down and network shake contribute the biggest part of alerts
- Application specific flows take an important part in Unicorn
- Disk failure causes a lot of issues in Rheos

Event status

The distribution of the three final event status(Finished, Failed, Skipped) for each event type is as follows:

Event Type	Finished	Failed	Skipped
MedieErrorDisk	50%	9%	41%
RollingRestart	36%	1%	63%
NodeDown	2%	3%	95%
Replace	20%	79%	1%

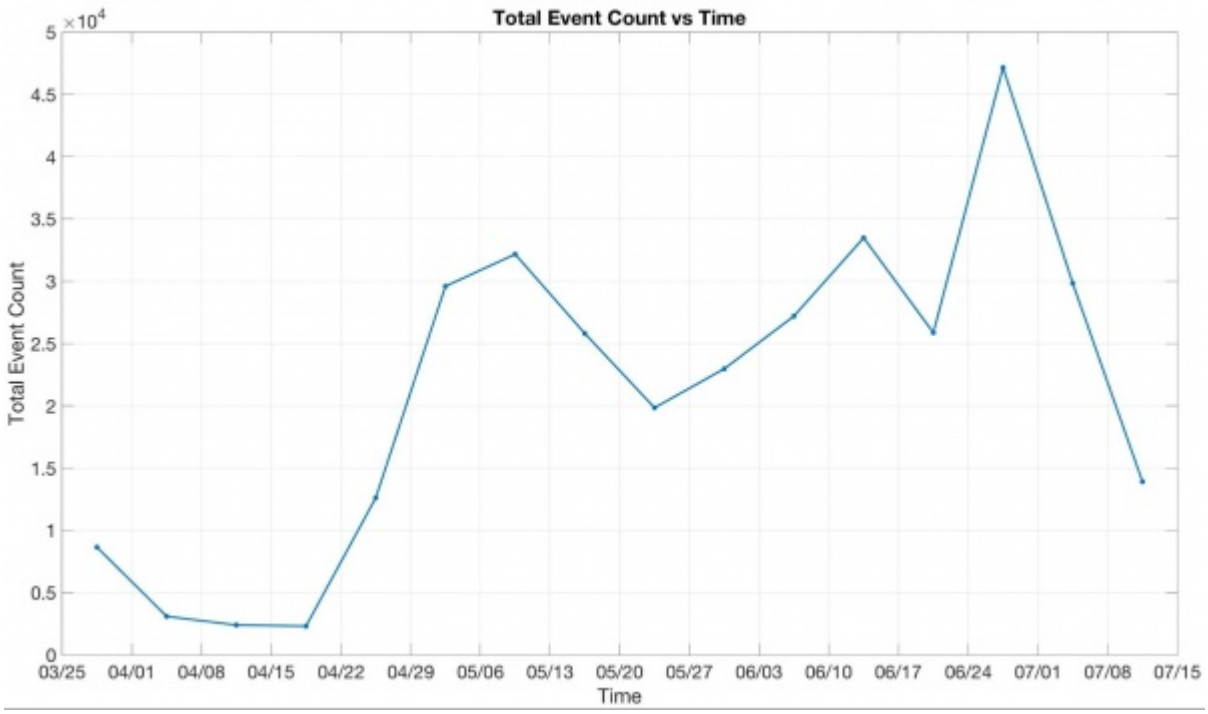
BehaviorMmLagHighFor5Min	0.04%	0.06%	99.9%
PartitionReassign	11%	1%	78%
LeaderReelection	70%	1%	29%
ReadonlyDisk	81%	6%	13%
HighDiskUsageFor5Min	18%	3%	79%
StraasAgentHeal	76%	8%	16%
Total	6%	2%	92%

Several observations based on the above table:

- Most of the events are marked as “Skipped”. When the problem solved once, the other related events waiting in the queue will not need to take any action. Handle false alert is an important advantage of Unicorn.
- Disk related issue could be remediated efficiently.
- Replace flow has the highest failed rate. As the finally step of many flows, replace flow suffers a lot of underlying system failures.

Event Count

A simple count of handled events after deploying to production is shown in below image:



Several observations from this diagram:

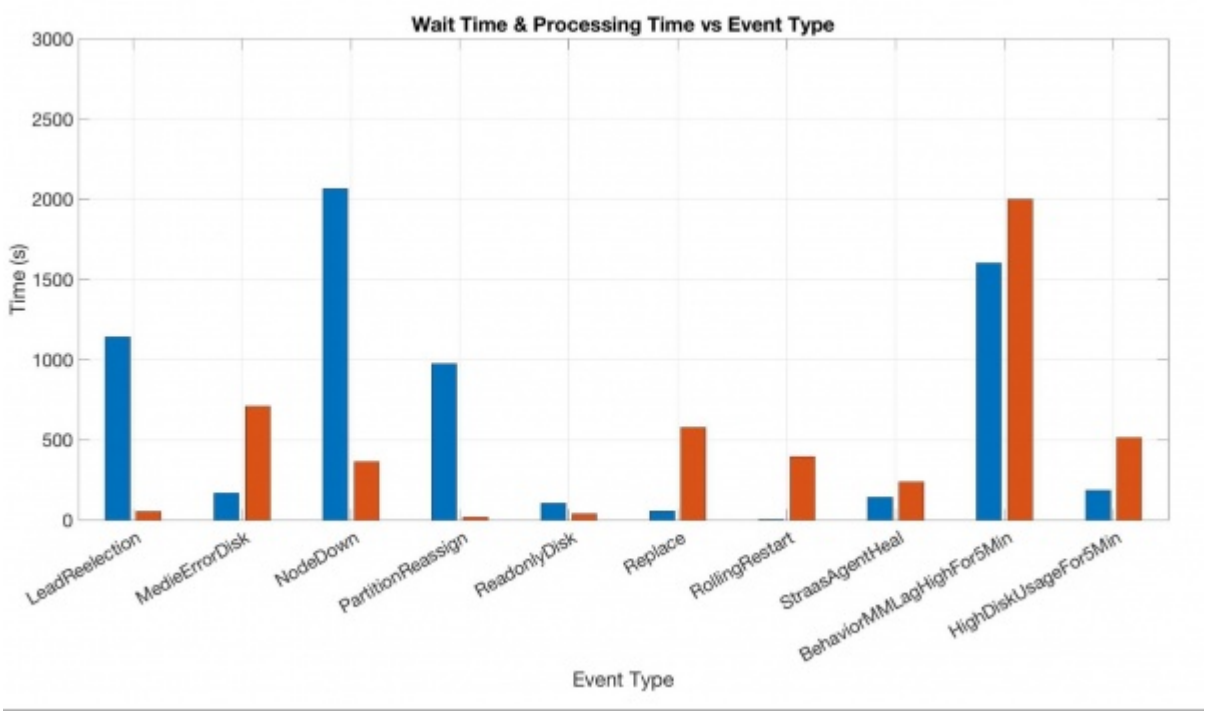
- Each week, Unicorn will scan thousands of events
- After enable new features, the handled events count increased obviously
- When some issues were solved, the events that need to be handled decreased

Event Timestamp

Two concepts to show the efficiency of Unicorn are as below:

- Time to response(Blue color on the chart): processing_timestamp - creation_timestamp, the time between Unicorn received the event and start processing timestamp

- Time to resolve(Red color on the chart): last_modified_timestamp - processing_timestamp, the time that Unicorn to mark an event as “Finished” or “Failed”.



Several observations from this chart:

- Event with rate control(NodeDown and BehaviorLagHighFor5Min) and low priority(LeaderReelection and PartitionReassign) have a bigger response time
- Urgent events, including disk related and manually dropped events, have a small response time, under 5 minutes
- Most of the resolve time is around 10 minutes
- BehaviorMMLagHighFor5Min needs some sampling time, may need to enhance the algorithm

Summary

Unicorn saves the support efforts, shorten the SLA to remediate urgent issues and guarantee the stability of stateful clusters in Rheos. From technical perspective, the contributions of Unicorn are as follows: an event driven solution to modeling the past Rheos support experiences and easy to extend for new issues; define the concept of “GroupId” and “Priority” to isolate conflicting operations and make sure acceptable efficiency; some best practices based on NAP which may provide an example to other tools to manage clusters running on C3.

Unicorn needs to be continuously enhanced in Rheos daily work. New issues and new flows keep to be found when go through the alert list and generated from daily support work. It is also very important to add intelligence to Unicorn in the future to handle grand new issues automatically.

TOPICS: Cloud, Data Infrastructure and Services

Previous Post:
The Web Push Checklist

eBay Tech Blog Social

View More

Related Blog Posts

Large-Scale Product Image Recognition with Cloud TPUs
Jul 24, 2018

Managing HTTP Header Size on NetScaler Load Balancers
Jul 17, 2018

Optimization Study on Processing Order of NetScaler Load Balancer Layer 7 Policies
Jul 3, 2018 | 3 comments

Beats @ eBay - Collectbeat - A Journey where Company and Community Come Together
Oct 4, 2017 | 8 comments

Cube Planner - Build an Apache Kylin OLAP Cube Efficiently and Intelligently
Aug 2, 2017 | 6 comments

Most Commented Posts

Caching HTTP POST Requests and Responses
Aug 20, 2012 | 9 comments

Event Sourcing in Action with eBay's Continuous Delivery Team (Part 2)
Jul 12, 2018 | 3 comments

Optimization Study on Processing Order of NetScaler Load Balancer Layer 7 Policies
Jul 3, 2018 | 3 comments

Stepping Towards a Password-Free World
May 15, 2018 | 2 comments

Announcing the Accelerator
Apr 26, 2018 | 2 comments

Press Room

Follow Us

Contact Us
