Apache Con 2022
Oct 6, 2022

Heesung Sohn
Sr. Platform Engineer • StreamNative

Tech Deep Dive

Understanding Pulsar Broker Load Balancing

Heesung Sohn
Sr. Platform Engineer
StreamNative

A platform engineer at StreamNative based in the San Francisco Bay Area.

Currently focusing on Pulsar Broker Load Balancing.

Previously worked on scaling Aurora Mysql internals for its Serverless features at AWS.

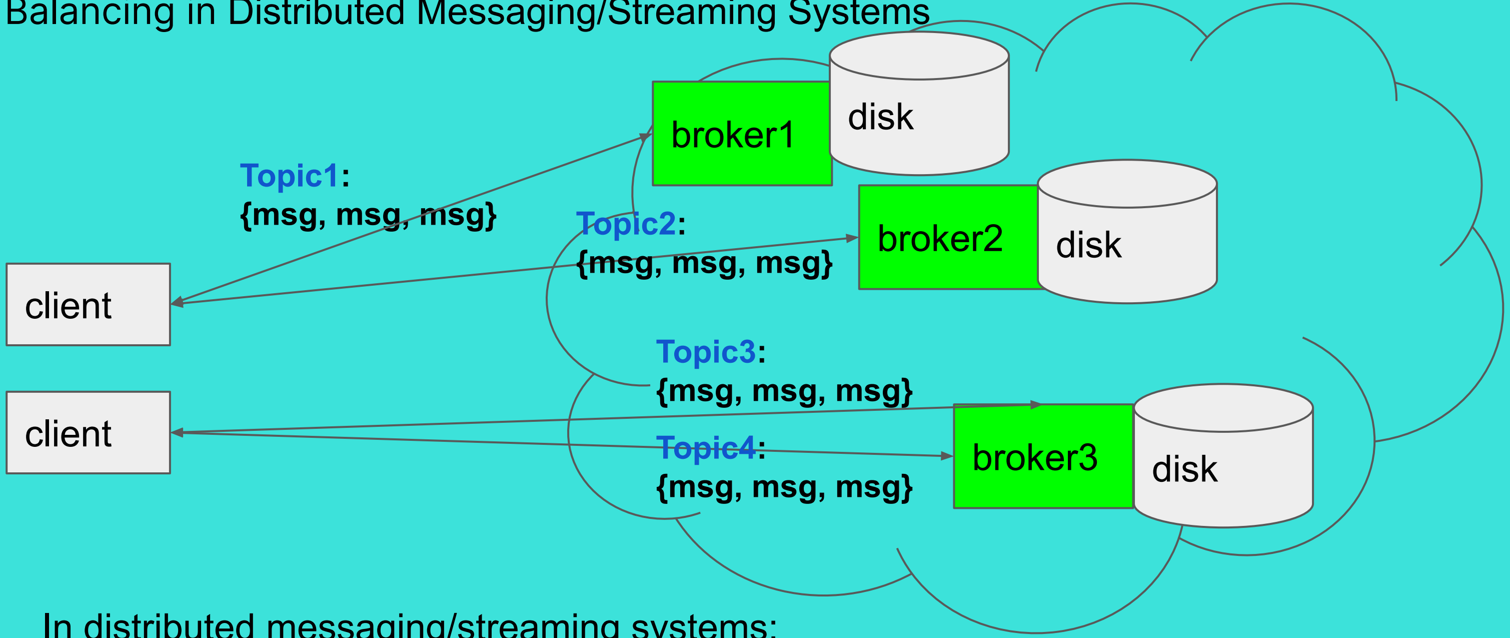
LinkedIn
https://www.linkedin.com/in/heesung-sohn-aa5b9561/

Blog : Achieving Broker Load Balancing with Apache Pulsar
https://streamnative.io/blog/engineering/2022-07-21-achieving-broker-load-balancing-with-apache-pulsar/

Agenda

Understanding Broker Load Balancing

Load Balancing in Distributed Messaging/Streaming Systems



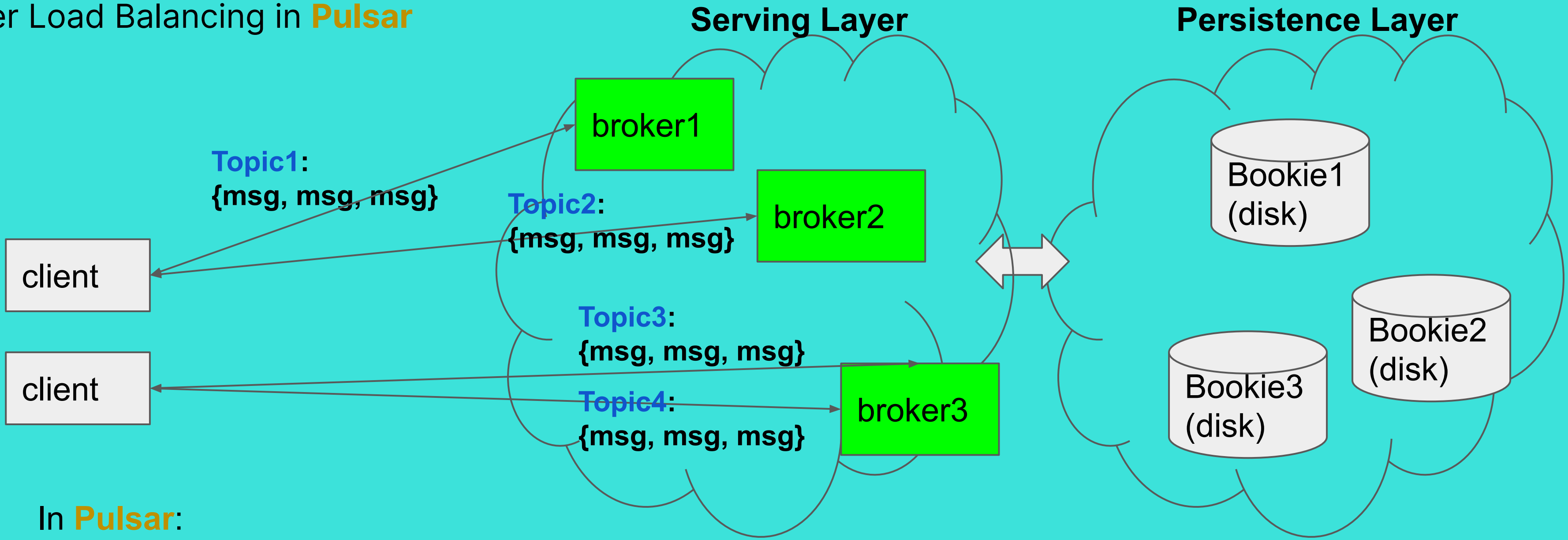
In distributed messaging/streaming systems:

- ➔ Messages are grouped under topics
- ➔ Message pub-sub for a topic is served by a **single** broker
- ➔ Topics(or groups of topics) are considered as a “good” load-balance entity

In our context, Load balancing refers to efficiently distributing **topic messages** across brokers.

Understanding Broker Load Balancing

Broker Load Balancing in **Pulsar**



In **Pulsar**:

Pulsar separates **serving-persistence** layers.

- ➔ Brokers serve **topics'** pub-sub sessions.
- ➔ Brokers read/write messages in Bookies.

Broker Load balancing refers to efficiently distributing **topic serving sessions** across brokers.

Q: How does Pulsar make the Topic Load Balance efficient?

Understanding Broker Load Balancing
Idea: Dynamic Topic Rebalancing

Q: How does Pulsar balance topics(sessions) across brokers?

A: Pulsar uses **dynamic topic rebalancing**.
- ➔ Assign topics to underloaded brokers.
- ➔ Unload topics from overloaded brokers(high cpu, memory, I/O ...)

Q: Is **dynamic unloading** possible without harming the performance?

A: Yes, Pulsar can seamlessly transfer topic sessions **thanks to the serving-persistence separation**.

In Pulsar, unlike monolithic systems,

Topic-broker assignments are “**flexible**” as brokers do not persist messages locally.

- ➔ New owner brokers simply establish new sessions with the clients.
- ➔ Minimal client connection jitters. New owner brokers look up bookie metadata.

Agenda

Understanding Broker Load Balancing

Topics are balanced to brokers at the bundle level

Topics are grouped into **bundles** as Broker Load Balancing Unit

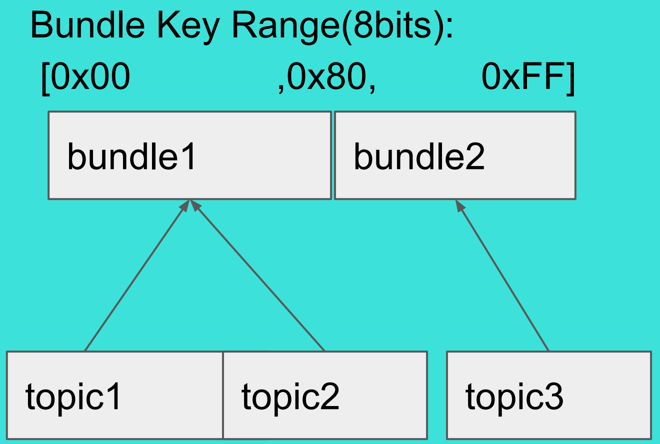
Q: Why need this middle layer group, bundles?

With the multi-tenancy nature,
Pulsar needs to scale for millions of topics.

Problem: too much to track millions of the topic-level metadata

Solution: topic sharding/bundling.

Topic-Bundle LooKup by Hashed Sharding



hash("topic1") => 0x0F(bundle key)

hash("topic2") => 0x4F

hash("topic3") => 0x8F

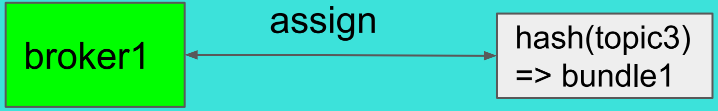
Agenda

Understanding Broker Load Balancing

Auto Bundle Load Balance Logics in Pulsar

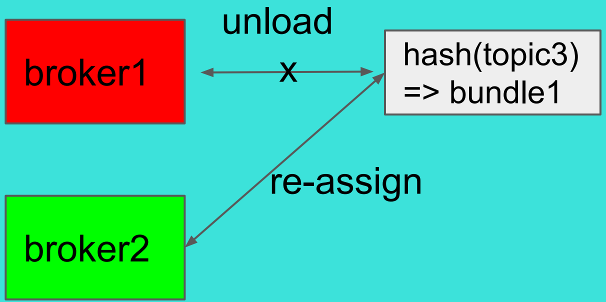
1. Bundle-Broker Assignment

➔ Assign to a new broker when no owner



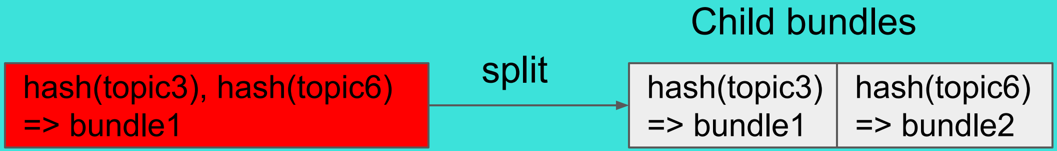
2. Bundle Unload

➔ Unload bundles from overloaded to underloaded brokers



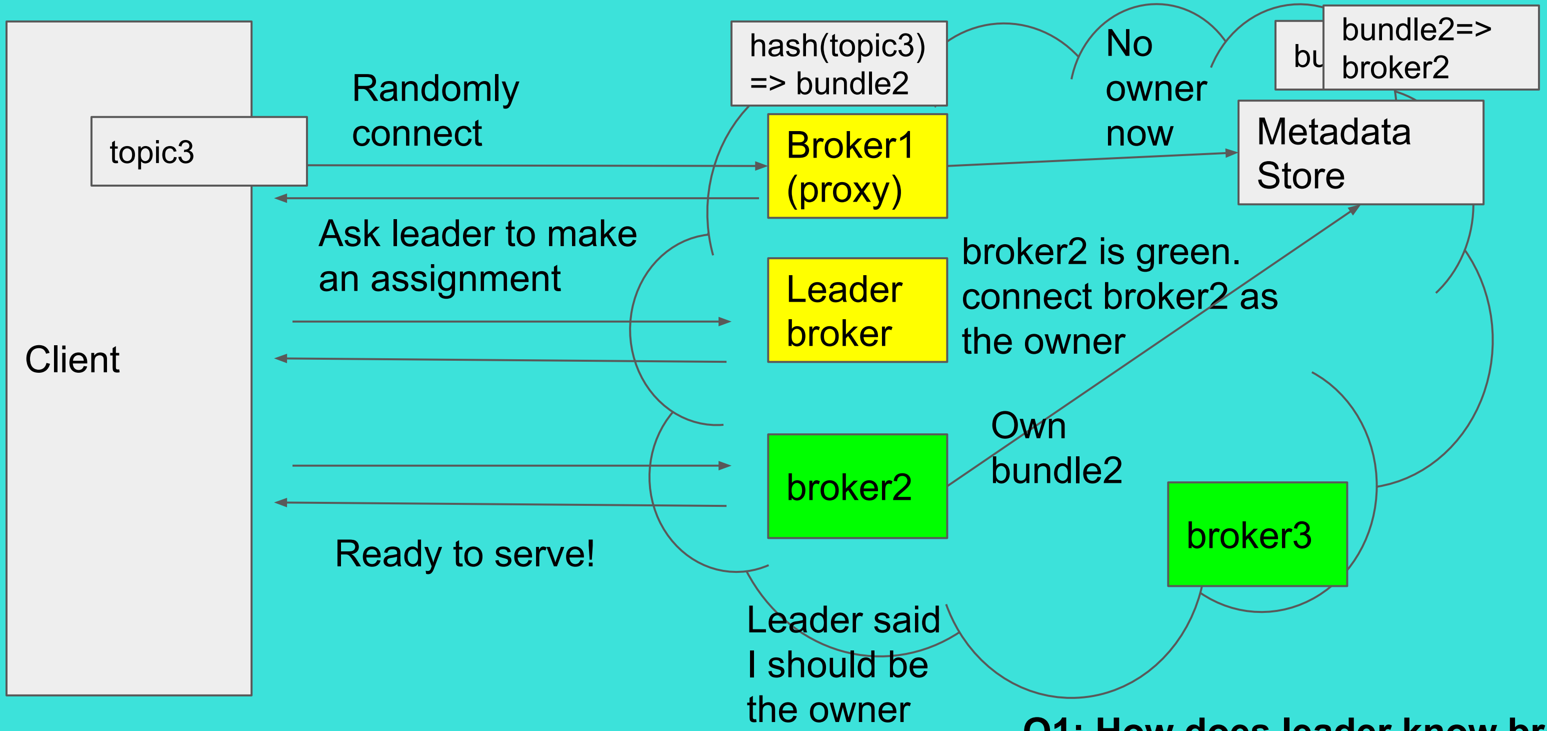
3. Bundle Split

➔ Split overloaded bundles



Understanding Broker Load Balancing

Bundle-Broker Assignment (Assign a bundle to a new broker when no owner)



Q1: How does leader know broker2 is green?

Q2: What's the strategy to select "a broker" among the available brokers?

Understanding Broker Load Balancing

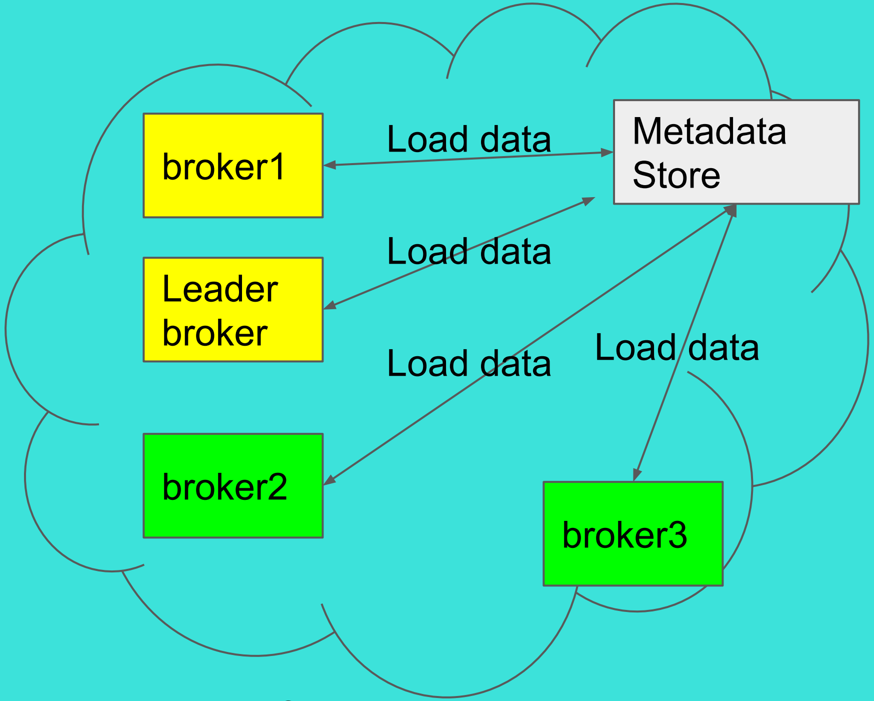
Leader collects global load info

Bundle Load Data :

bundle-level msg in/out rates.

Broker Load Data:

CPU, memory, and network throughput in/out rates.



Q1: How does leader know broker2 is green?

A: The leader broker **collects global load info via Metadata Store.**

Currently, the leader makes all load balance decisions.

Understanding Broker Load Balancing

Bundle-Broker Assignment Strategy

Q2: What's the strategy to select "a broker" among the available brokers?

A: Pulsar can configure the following strategies(configurable by ModularLoadManagerStrategy.)

LeastLongTermMessageRate(default)

load=

If max(cpu, mem, network) <= threshold(85%)

=> *f*(longTermMsgIn, longTermMsgOut)

Select a random broker among least long-term msg rate.

LeastResourceUsageWithWeight(new)

cur_load = max(cpu, mem, network)

load = w * load + (1 - w) * cur_load ← Exponential Moving Average(EMA) w=0.9

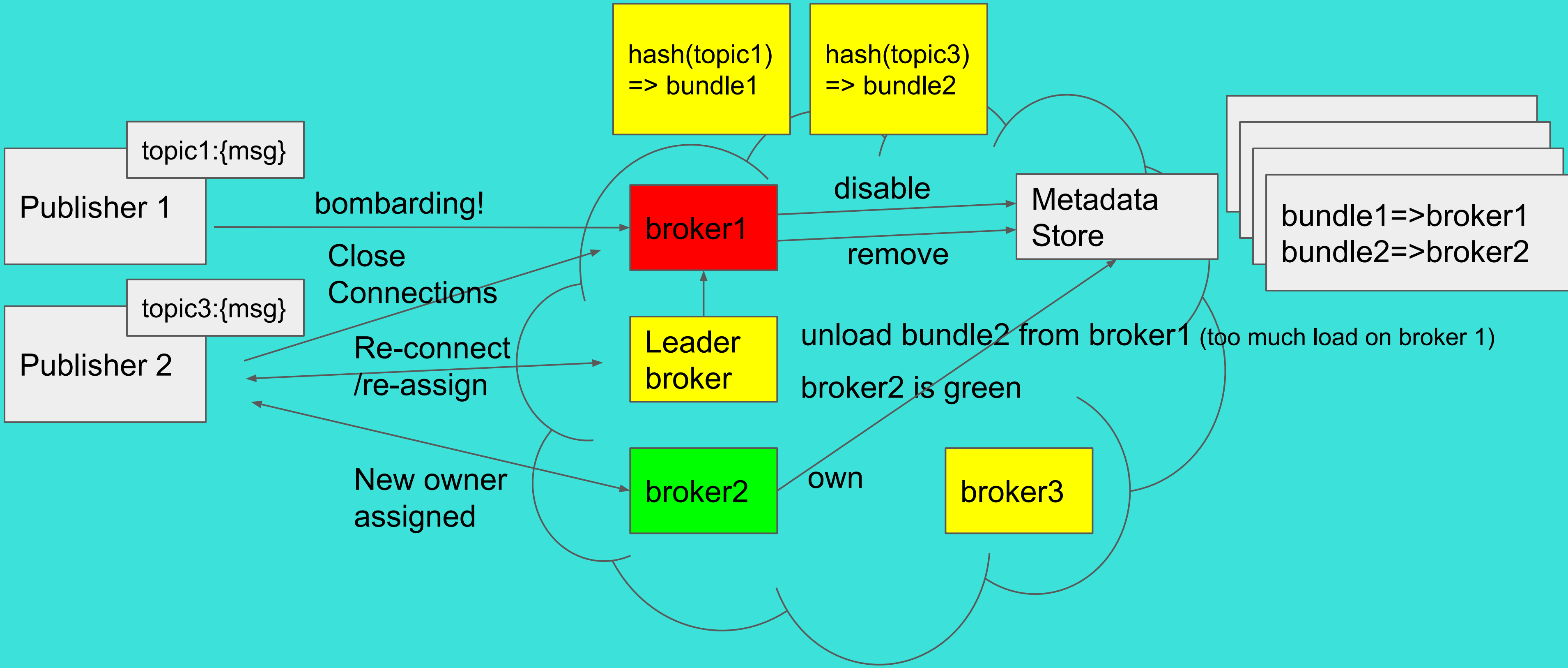
avg_load = avg(load) from all brokers' load

candidates = brokers, load < avg_load - α(default 10%)

Select a random broker among the candidates

Understanding Broker Load Balancing

Bundle Unload (Unload bundles from overloaded to underloaded brokers)



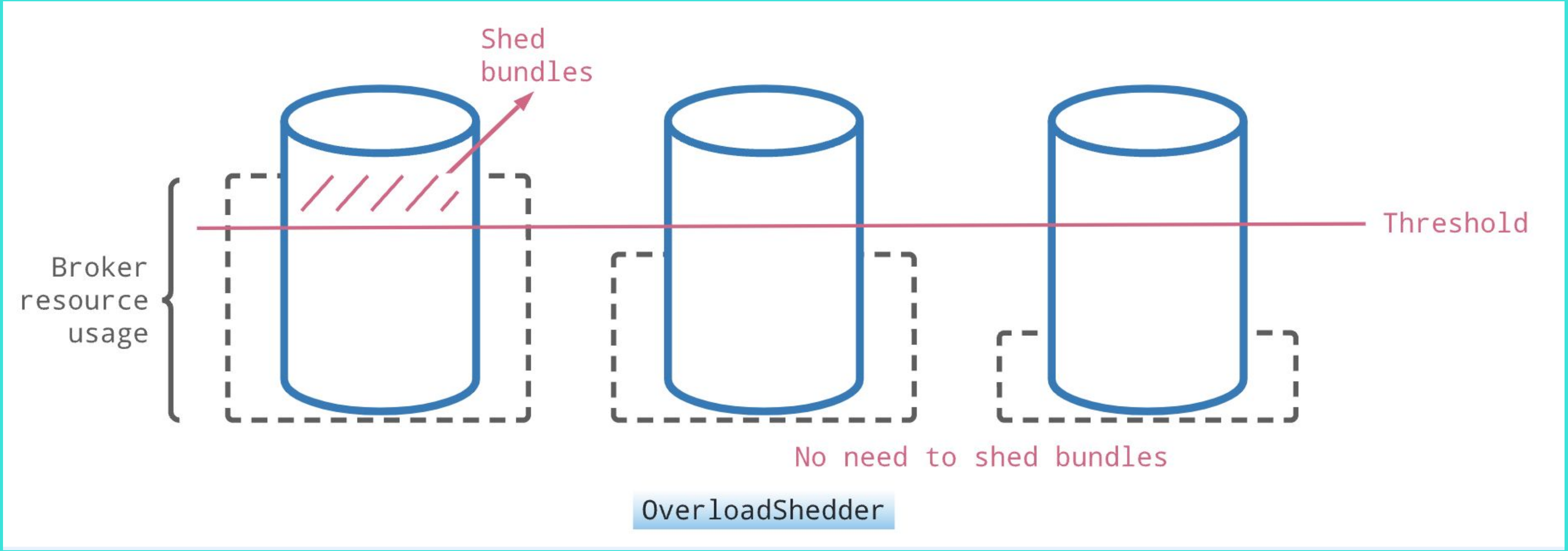
Q: What's the strategy to select overloaded brokers and bundles?

Understanding Broker Load Balancing

Bundle Unloading(Shedding) Overload Shedder Strategy

Q: What's the strategy to select overloaded brokers and bundles?

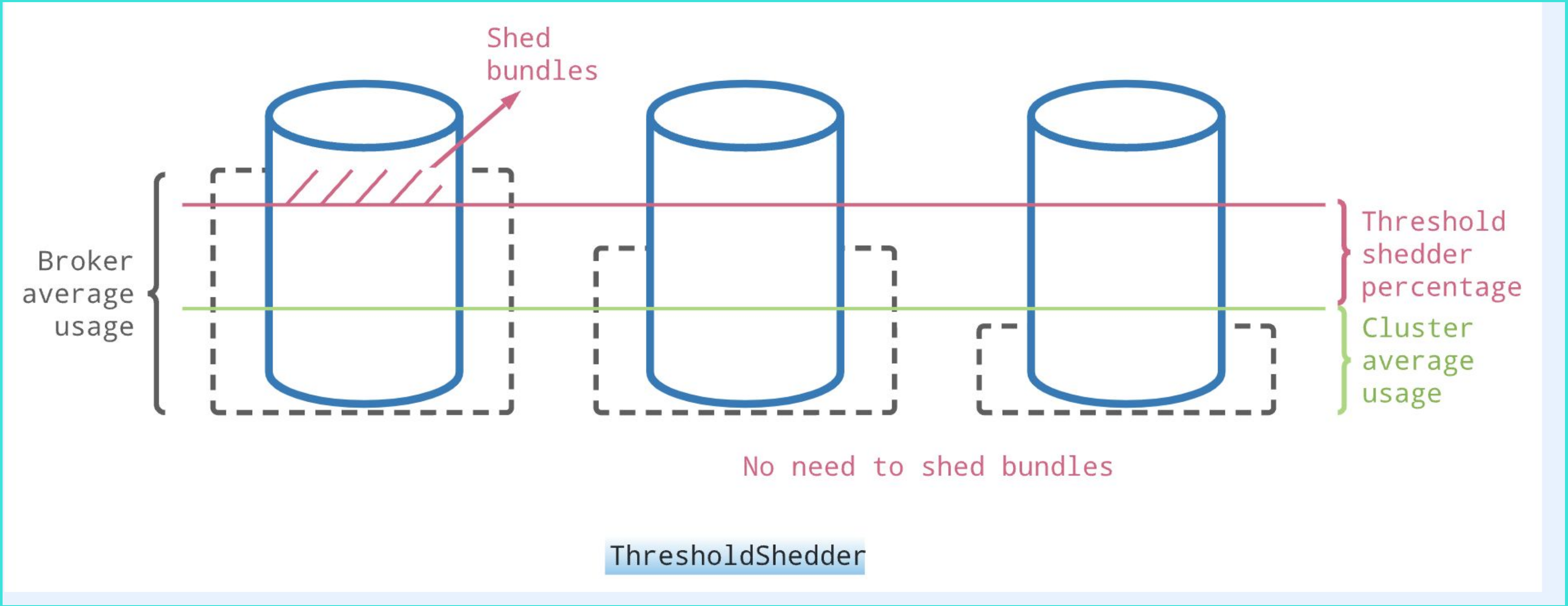
A: Pulsar can configure **ThresholdShedder(default), OverloadedShedder, UniformLoadShedder**



```
loadBalancerBrokerOverloadedThresholdPercentage= default 85%
```

Understanding Broker Load Balancing

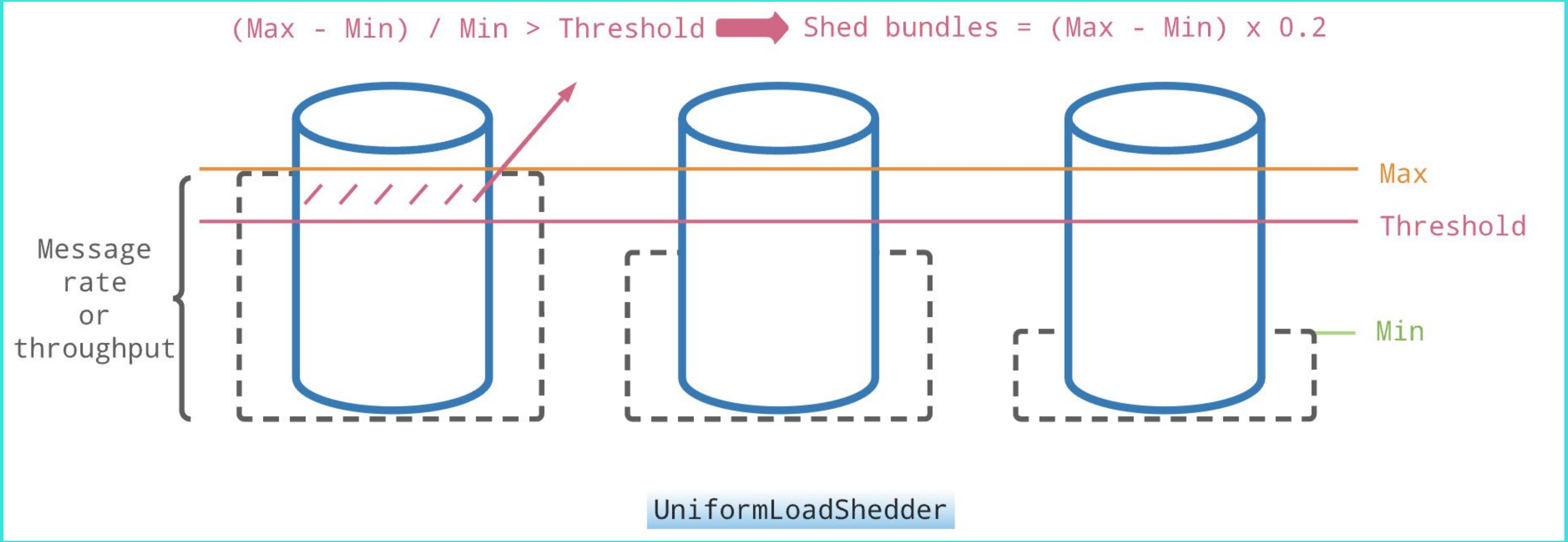
Bundle Unloading(Shedding) ThresholdShedder Strategy



```
loadBalancerBrokerThresholdShedderPercentage = default 10%
```

Understanding Broker Load Balancing

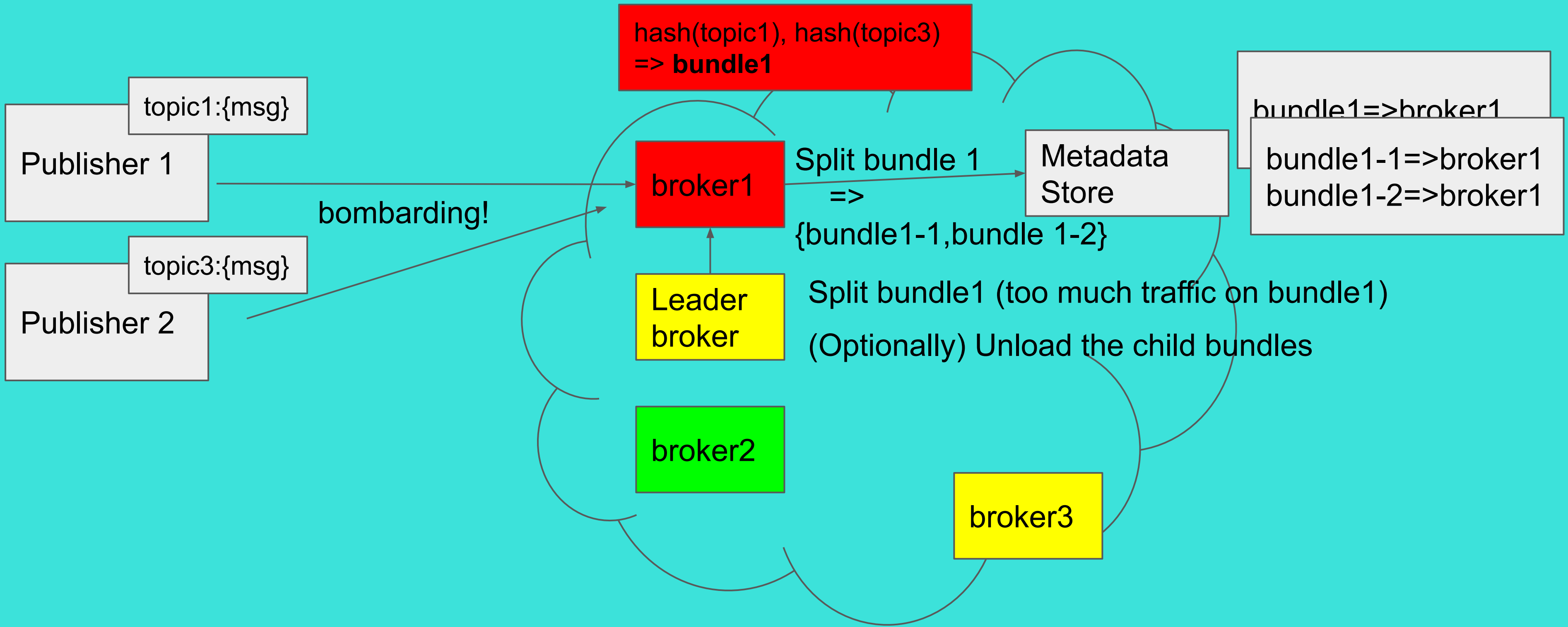
Bundle Unloading(Shedding) UniformLoadShedder Strategy


(Max - Min) / Min > Threshold
Shed bundles = (Max - Min) x 0.2
Max
Threshold
Min
Message rate or throughput
UniformLoadShedder

```
loadBalancerMsgThroughputMultiplierDifferenceShedderThreshold
loadBalancerMsgRateDifferenceShedderThreshold
```

Understanding Broker Load Balancing

Bundle split (Split overloaded bundles)



Q: What’s the strategy to split overloaded bundles?

Understanding Broker Load Balancing

Bundle Split Strategies

Q: What’s the strategy to split overloaded bundles?

A: Pulsar can configure when and how to splits bundles.

Threshold-based Bundle Split Strategy (when to split)

Split bundles if any resource(OR gate) is beyond LoadBalancerNamespaceBundle* thresholds.

Defaults

LoadBalancerNamespaceBundleMaxTopics = 1000
LoadBalancerNamespaceBundleMaxSessions = 1000
LoadBalancerNamespaceBundleMaxMsgRate = 30000
LoadBalancerNamespaceBundleMaxBandwidthMbytes = 100

Bundle Split Boundary Compute Strategy (how to split)

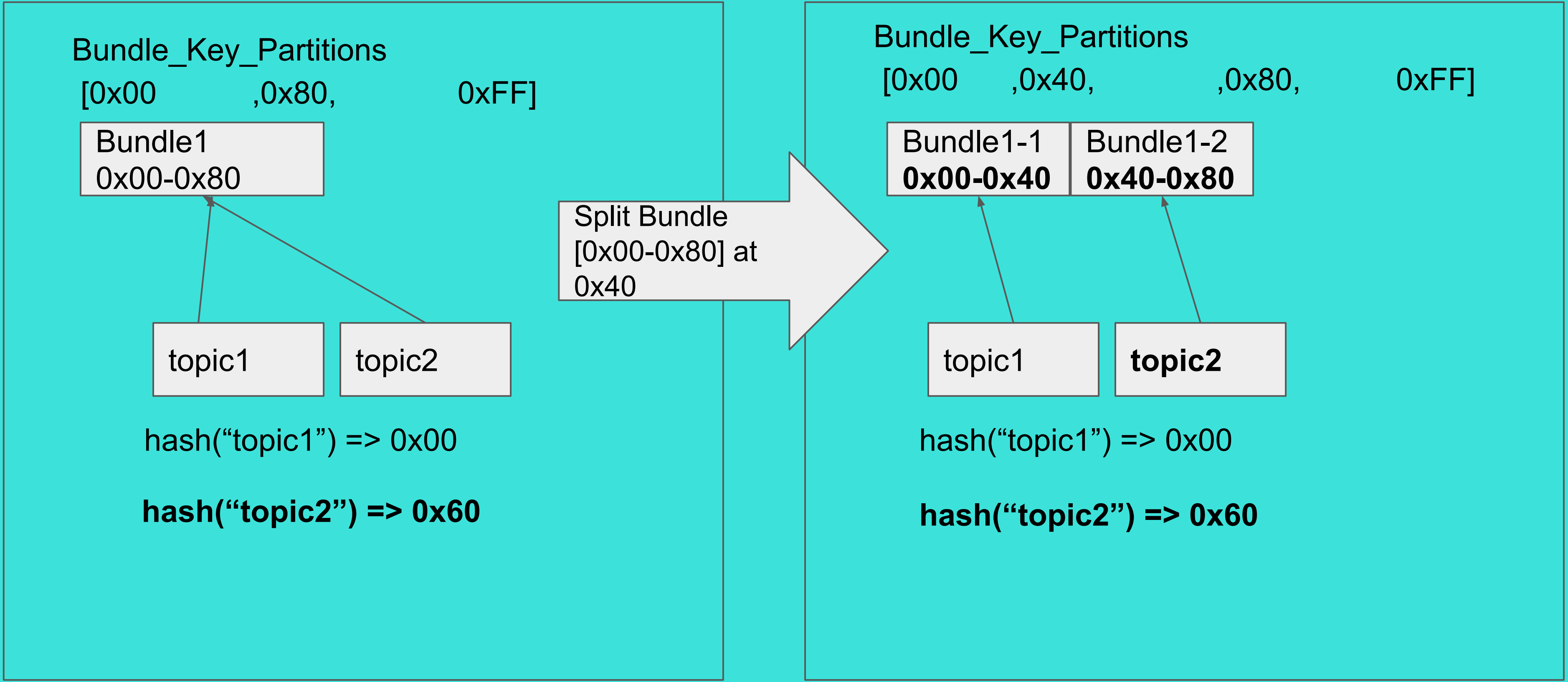
RANGE_EQUALLY_DIVIDE_NAME (default):
split to parts with **the same hash range size**

TOPIC_COUNT_EQUALLY_DIVIDE:
split to parts with **the same topic count.**

configurable by DefaultNamespaceBundleSplitAlgorithm.

Understanding Broker Load Balancing

RANGE_EQUALLY_DIVIDE_NAME Split Example



Agenda

Intro: Pulsar Broker Load Balancing

Bundles

Auto Bundle Load Balance Logic

Operation Tips

On-going Work

Q&A

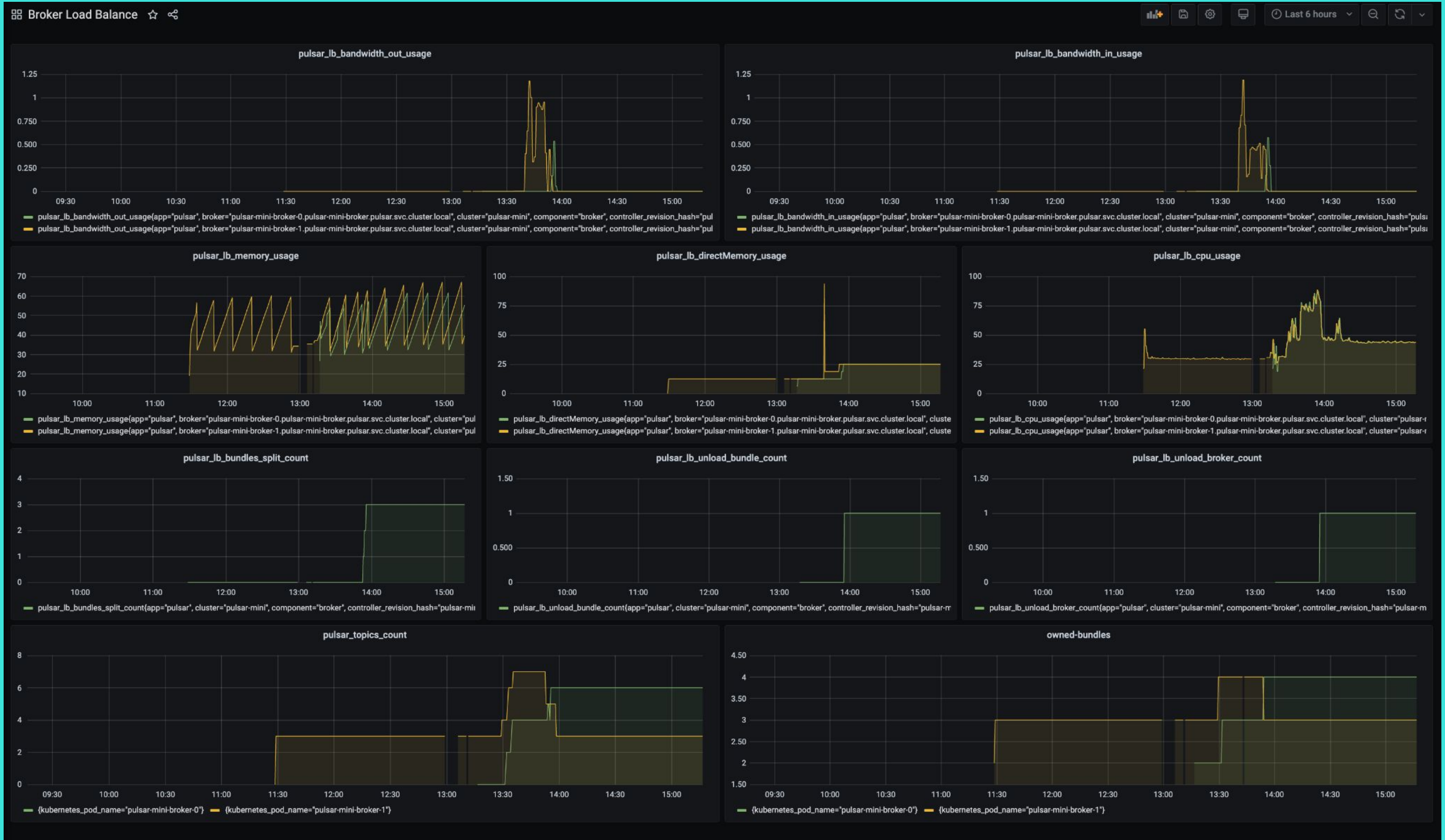
Understanding Broker Load Balancing

Load Balance Metrics: Useful to monitor Load Balance Input / Output

Type	Name	Description
Load Balance Input signal	pulsar_lb_bandwidth_in_usage	Broker bandwidth in usage % out of 100%
Load Balance Input signal	pulsar_lb_bandwidth_out_usage	Broker bandwidth out usage % out of 100%
Load Balance Input signal	pulsar_lb_memory_usage	Broker heap usage % out of 100%
Load Balance Input signal	pulsar_lb_directMemory_usage	Broker dict_memory usage % out of 100%
Load Balance Input signal	pulsar_lb_cpu_usage	Broker cpu usage % out of 100%
Load Balance Split Output	pulsar_lb_bundles_split_count	Bundle split counts
Load Balance Unload Output	pulsar_lb_unload_bundle_count	Bundle unload counts
Load Balance Unload Output	pulsar_lb_unload_broker_count	Bundle unload broker counts
Load Balance Assignment Output	pulsar_topics_count	Serving topic counts
Load Balance Assignment Output	owned-bundles	Bundle ownership cache size by caffeine_cache_estimated_size

Understanding Broker Load Balancing

Load Balance Dashboard



pulsar_lb_bandwidth_out_usage

```
pulsar_lb_bandwidth_out_usage{app="pulsar", broker="pulsar-mini-broker-0.pulsar-mini-broker.pulsar.svc.cluster.local", cluster="pulsar-mini", component="broker", controller_revision_hash="pul
pulsar_lb_bandwidth_out_usage{app="pulsar", broker="pulsar-mini-broker-1.pulsar-mini-broker.pulsar.svc.cluster.local", cluster="pulsar-mini", component="broker", controller_revision_hash="pul
```

pulsar_lb_bandwidth_in_usage

```
pulsar_lb_bandwidth_in_usage{app="pulsar", broker="pulsar-mini-broker-0.pulsar-mini-broker.pulsar.svc.cluster.local", cluster="pulsar-mini", component="broker", controller_revision_hash="pulsa
pulsar_lb_bandwidth_in_usage{app="pulsar", broker="pulsar-mini-broker-1.pulsar-mini-broker.pulsar.svc.cluster.local", cluster="pulsar-mini", component="broker", controller_revision_hash="pulsa
```

pulsar_lb_memory_usage

```
pulsar_lb_memory_usage{app="pulsar", broker="pulsar-mini-broker-0.pulsar-mini-broker.pulsar.svc.cluster.local", cluster="pul
pulsar_lb_memory_usage{app="pulsar", broker="pulsar-mini-broker-1.pulsar-mini-broker.pulsar.svc.cluster.local", cluster="pul
```

pulsar_lb_directMemory_usage

```
pulsar_lb_directMemory_usage{app="pulsar", broker="pulsar-mini-broker-0.pulsar-mini-broker.pulsar.svc.cluster.local", cluste
pulsar_lb_directMemory_usage{app="pulsar", broker="pulsar-mini-broker-1.pulsar-mini-broker.pulsar.svc.cluster.local", cluste
```

pulsar_lb_cpu_usage

```
pulsar_lb_cpu_usage{app="pulsar", broker="pulsar-mini-broker-0.pulsar-mini-broker.pulsar.svc.cluster.local", cluster="pulsar-
pulsar_lb_cpu_usage{app="pulsar", broker="pulsar-mini-broker-1.pulsar-mini-broker.pulsar.svc.cluster.local", cluster="pulsar-
```

pulsar_lb_bundles_split_count

```
pulsar_lb_bundles_split_count{app="pulsar", cluster="pulsar-mini", component="broker", controller_revision_hash="pulsar-mi
```

pulsar_lb_unload_bundle_count

```
pulsar_lb_unload_bundle_count{app="pulsar", cluster="pulsar-mini", component="broker", controller_revision_hash="pulsar-m
```

pulsar_lb_unload_broker_count

```
pulsar_lb_unload_broker_count{app="pulsar", cluster="pulsar-mini", component="broker", controller_revision_hash="pulsar-m
```

pulsar_topics_count

```
{kubernetes_pod_name="pulsar-mini-broker-0"}   {kubernetes_pod_name="pulsar-mini-broker-1"}
```

owned-bundles

```
{kubernetes_pod_name="pulsar-mini-broker-0"}   {kubernetes_pod_name="pulsar-mini-broker-1"}
```

Understanding Broker Load Balancing

Useful Admin CLIs to Check Bundle States

1. How to list the bundles in the namespace

```
# bin/pulsar-admin namespaces bundles my-tenant/my-namespace
⇒ "boundaries" : [ "0×00000000", "0×10000000", ..., "0xffffffff" ], "numBundles" : 16
```

2. How to list the topics in the bundle

```
# bin/pulsar-admin topics list my-tenant/my-namespace --bundle 0x00000000_0x10000000
⇒ persistent://my-tenant/my-namespace/my-topic
```

3. How to look up the bundle by topic

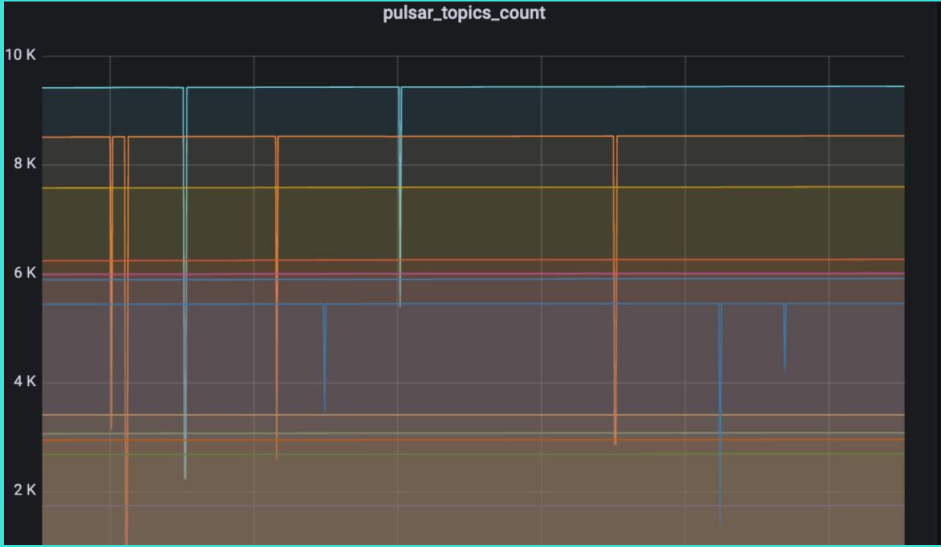
```
# bin/pulsar-admin topics bundle-range persistent://my-tenant/my-namespace/my-topic
⇒ 0×00000000_0×10000000
```

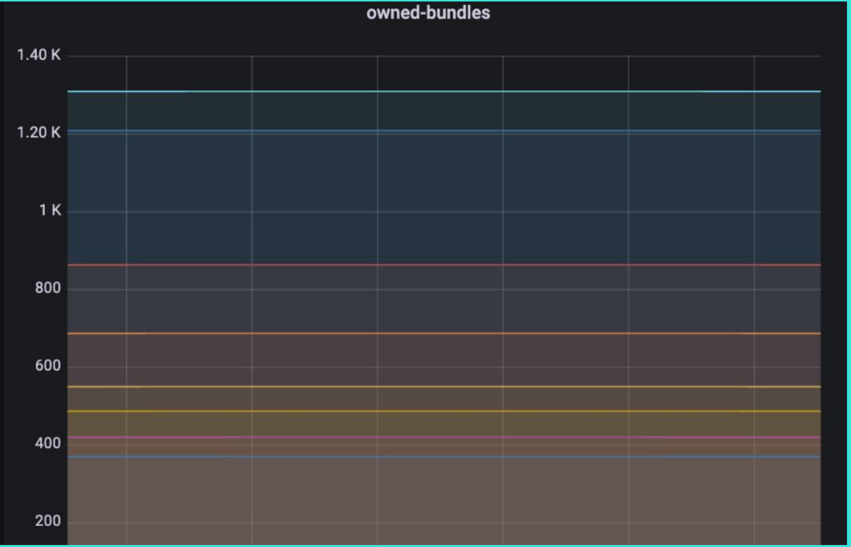
4. How to look up the owner broker by topic

```
# bin/pulsar-admin topics lookup persistent://my-tenant/my-namespace/topic
⇒ pulsar://my-broker-1:6650
```

Understanding Broker Load Balancing

Manual Split and Unload





Q: Can Admin manually split and unload bundles?

A: Yes. The unloaded bundles will be reloaded to the next available brokers soon.

Let’s check the CLIs for this operations.

Understanding Broker Load Balancing

How to check bundle load stats

How to check bundle load stats

```
# pulsar-admin --admin-url http://my-broker-x-url:8080 broker-stats load-report
⇒
...
 "bundleStats" : {
   "my-tenant/my-namespace/0×80000000_0xc0000000" : {
     "msgRateIn" : 2100.99
     "msgThroughputIn" : 2367100.92
     "msgRateOut" : 2100.99
     "msgThroughputOut" : 2367100.92
     "consumerCount" : 2200,
     "producerCount" : 2200,
     "topics" : 2300,
     "cacheSize" : 107600
   }
```

Understanding Broker Load Balancing

Manual Split and Unload

1. How to manually split and unload the bundles in the namespace

```
# pulsar-admin namespaces split-bundle --bundle 0×80000000_0xc0000000 -san range_equally_divide -u tenant/namespace
```

2. How to manually split and unload the largest bundles in the namespace

```
# pulsar-admin namespaces split-bundle -b LARGEST -san topic_count_equally_divide -u tenant/namespace
```

3. How to unload a bundle

```
# pulsar-admin namespaces unload tenant/namespace -b 0×80000000_0xc0000000
```

4. How to unload every bundle in the namespace.

```
# pulsar-admin namespaces unload tenant/namespace
```

Agenda

Understanding Broker Load Balancing
Recent Community Work

More randomized assignment strategy with additional signals

- **LearResourceUsageWithWeight**, new bundle assignment 16281

Improve input accuracy and output visibility

- Resource usage limit validation and better unload logging(sample non-unload decisions) 16937
- Disregard fluctuating memory when computing load report frequency (less zk overhead) 17598
- Better cgroup cpu usage collection (more accurate cgroup cpu usage) 17820

[PIP-192] Broker Load Balancer Improvement Project (architectural change) 16691

Understanding Broker Load Balancing

PIP-192: Goals and Proposals

Goal 1: Make auto load balance fault-tolerant, consistent, distributed.

Currently	Proposal
Leader globally makes **all** load balance (assignment/unload/split) decisions and commands each owner broker via RPC with retries.	Each owner broker decides and runs **assignment** and **split** logic. Leader broker still globally decides unload logic. RPC → Event-Sourcing : brokers reliability react on load balance commands from a persistent topic.

Understanding Broker Load Balancing

PIP-192: Goals and Proposals

Goal 2: Efficiently replicate ownership and load data across brokers for high-performance

Currently	Proposal
The metadata are persisted in ZK and replicated to all broker's in-memory cache(via watcher)	For the ownership data, use a persistent topic and replicate to all broker's in-memory cache(via topic table-view) For the load data, use non-persistent topics and replicate only necessary metrics. (light-weight)

Understanding Broker Load Balancing
PIP-192: Goals and Proposals

Goal 3: Minimize the topic unavailability during unloading

Currently	Proposal
When clients tries to look up the new owner broker, they need to go through the bundle assignment logic via the leader broker.	Minimize the gap by “**transfer**”, where the new owner is pre-assigned.

Understanding Broker Load Balancing

PIP-192: Goals and Proposals

Goal 4: Provide ways to manually override unload decisions to particular brokers.

Currently	Proposal
Not supported.	Introduce “--dest” option in the unload admin command, using the new unload “transfer” behavior.

Understanding Broker Load Balancing

PIP-192: Topics and TableView for load data and bundle ownership store

Long-term Goal:

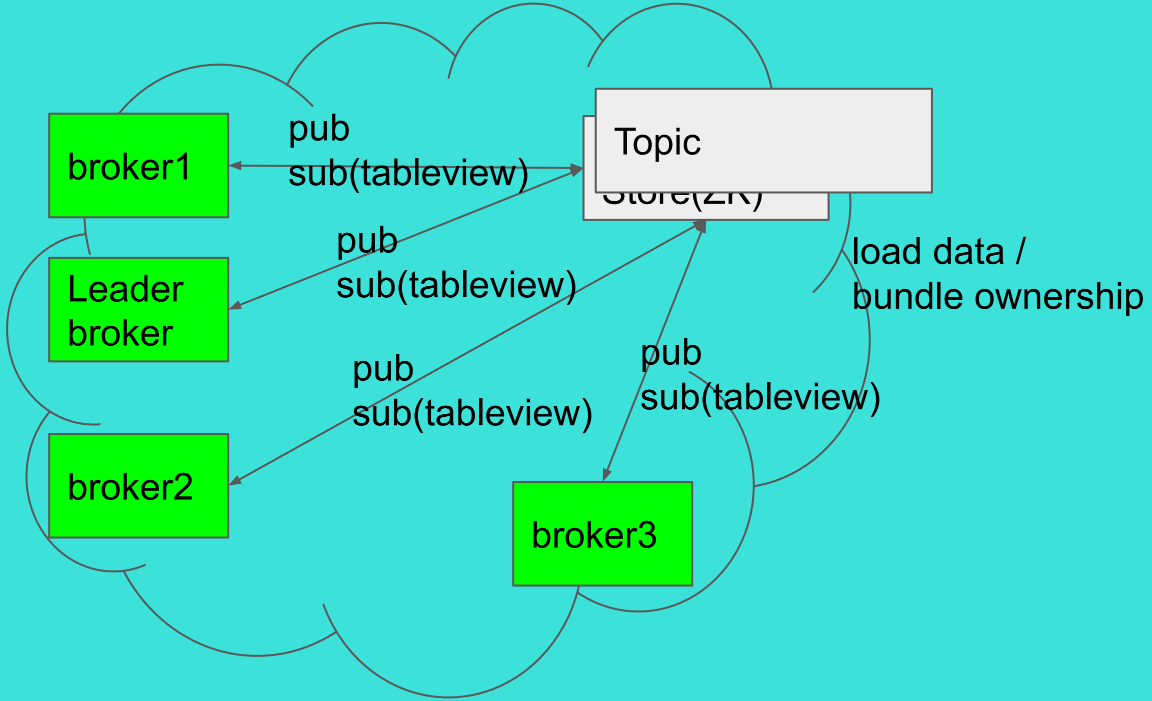
- Minimize ZK(Metadata store) dependency

[ZK dependency discussions]

Proposal:

Pulsar already has solutions to replicate such light-weight KV stores.

Use **Topic** and **TableView** to store and replicate load and bundle ownership data



In PIP-192, the bundle ownership topic is called as **"Bundle State Channel".**

Understanding Broker Load Balancing
PIP-192: Bundle State Channel

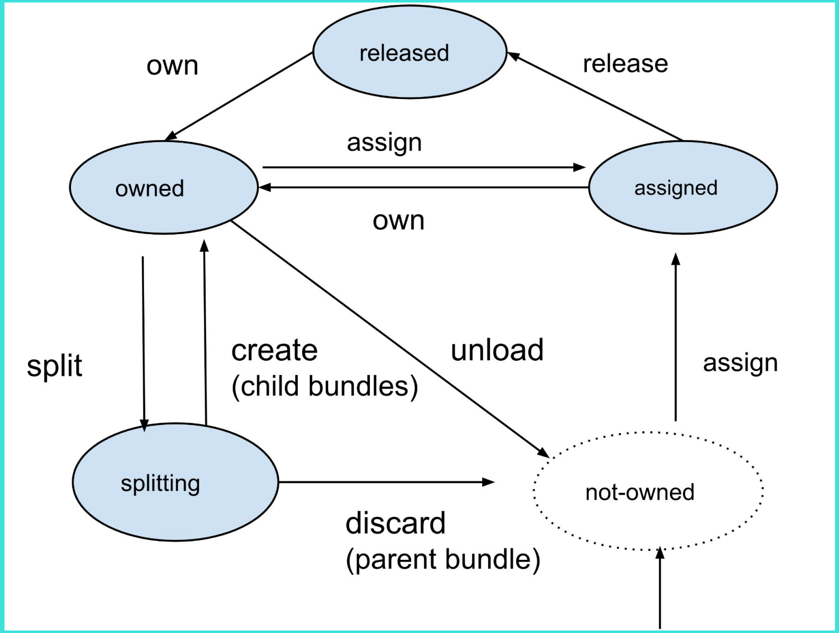
BSC is a bundle ownership store for owner broker discovery(lookup).

- A persistent topic with table-view
- All brokers publish and consume the ownership state(table-view).
- Materialize the global ownership state, by the topic auto compaction.
- Ownership look-ups can be deferred if the bundle states are in-flight(not "owned")

BSC broadcasts bundle state changes.

- Broadcast the *total order* of all bundle state changes (*sequential consistency*)
- All brokers react(plays their role) on the bundle state changes.

Bundle State Life Cycles



PIP-192: Bundle State Channel PoC Demo

- Bundle State Channel
- Auto Bundle Assignment
- Auto Bundle Unload
- Manual Bundle Transfer
- Compaction
- Recovery

Questions?

Apache Con 2022
Oct 6, 2022

Heesung Sohn
Sr. Platform Engineer • StreamNative

Tech Deep Dive

Understanding Broker Load Balancing

Thank you!