Apache Pulsar and its enterprise use cases

Yahoo Japan Corporation
Nozomi Kurihara

July 18th, 2018
Who am I?

Nozomi Kurihara

• Software engineer at Yahoo! JAPAN (April 2012 ~)
• Working on internal messaging platform using Apache Pulsar
• Committer of Apache Pulsar
Agenda

1. What is Apache Pulsar?
2. Why is Apache Pulsar useful?
3. How does Yahoo! JAPAN uses Apache Pulsar?
What is Apache Pulsar?
Agenda

1. What is Apache Pulsar?
   - History & Users
   - Pub-Sub messaging
   - Architecture
   - Client libraries
   - Topic
   - Subscription
   - Sample codes

2. Why is Apache Pulsar useful?

3. How does Yahoo! JAPAN uses Apache Pulsar?
Apache Pulsar

Flexible pub-sub system backed by durable log storage

- History:
  - 2014 Development started at Yahoo! Inc.
  - 2015 Available in production in Yahoo! Inc.
  - Sep. 2016 Open-sourced (Apache License 2.0)
  - June 2017 Moved to Apache Incubator Project
  - June 2018 Major version update: 2.0.1

- Competitors:
  - Apache Kafka
  - RabbitMQ
  - Apache ActiveMQ
  - Apache RocketMQ
  - etc.

- Users:
  - Oath Inc. (Yahoo! Inc.)
  - Comcast
  - The Weather Channel
  - Mercado Libre
  - Streamlio
  - Yahoo! JAPAN
  - etc.
Pub-Sub messaging

Message transmission from one system to another via **Topic**

- **Producers** publish messages to Topics
- **Consumers** receive only messages from Topics to which they subscribe
- **Decoupled (no need to know each other)** → asynchronous, scalable, resilient

Pub-Sub system

<table>
<thead>
<tr>
<th>Publish</th>
<th>Subscribe</th>
</tr>
</thead>
<tbody>
<tr>
<td>Producer</td>
<td>Consumer 1</td>
</tr>
<tr>
<td>message (log, notification, etc.)</td>
<td>Consumer 2</td>
</tr>
<tr>
<td>Topic</td>
<td>Consumer 3</td>
</tr>
</tbody>
</table>
Architecture

3 components:
- Broker
- Bookie
- ZooKeeper
Architecture - Broker

Broker
- Serving node for clients’ requests
- No data locality (stateless)

Pulsar Cluster

Configuration Store (Global ZK)

Local ZK

Bookie 1
Bookie 2
Bookie 3
Architecture - Bookie

- **Bookie (Apache BookKeeper)**
  - Storage node for messages
  - Durable, Scalable, Consistent, Fault-tolerant, Low-latency

**Apache BookKeeper**: distributed write-ahead log system
Architecture - ZooKeeper

Apache ZooKeeper

- Store metadata and configuration
- **Local ZK**: within local cluster
- **Configuration Store**: across all clusters

Copyright © 2016 - 2018 The Apache Software Foundation, licensed under the Apache License, version 2.0.
Client libraries

Supported client libraries:

- Java
- C++
- Python
- Go (Coming soon)

Also can use WebSocket API
**Topic URI**

```
persistent://tenant/namespace/topicname
```

**Persistency**

- **Persistent**
  - Producer
  - Consumer
  - Broker
  - Bookie
  - Examples:
    - persistent://shopping/user-log/buy
    - persistent://auction/notifications/bid
    - persistent://news/sports/articles
    - ...

- **Non-persistent**
  - Producer
  - Consumer
  - Broker
  - Bookie
  - Examples:
    - non-persistent://tenant/namespace/topicname
```
Subscription

- **Acknowledgement (ACK)**: confirmation that message is received
- **Cursor**: which messages are acknowledged
- **Subscription**: cursor and which consumers are connecting

Note: Each message is kept unless all subscriptions acknowledge
Sample code(Java) - Producer

// Create client by specifying the broker URI
PulsarClient client = PulsarClient.builder()
    .serviceUrl("pulsar://localhost:6650")
    .build();

// Create producer by specifying the topic URI
Producer<Byte[]> producer = client.newProducer()
    .topic("persistent://my-tenant/my-ns/my-topic")
    .create();

// Send a message
producer.send("My message".getBytes());
Sample code(Java) - Consumer

// Create client by specifying the broker URI
PulsarClient client = PulsarClient.builder()
     .serviceUrl("pulsar://localhost:6650")
     .build();

// Create consumer by specifying the topic URI and subscription name
Consumer consumer = client.newConsumer()
     .topic("persistent://my-tenant/my-ns/my-topic")
     .subscriptionName("my-subscription")
     .subscribe();

// Receive a message
Message msg = consumer.receive();
System.out.printf("Message received: %s", new String(msg.getData()));

// Acknowledge the message so that it can be deleted by the broker
consumer.acknowledge(msg);
Why is Apache Pulsar useful?
Agenda

1. What is Apache Pulsar?

2. Why is Apache Pulsar useful?
   › High performance
   › Scalability
   › Multi-tenant
   › Geo-replication
   › Pulsar Functions
   › Pulsar on Kubernetes
   › Apache Pulsar vs. Apache Kafka

3. How does Yahoo! JAPAN uses Apache Pulsar?
High performance

High throughput & low latency even if

▪ huge number of topics
▪ high reliability is required

ex. At Oath Inc.

Requirements:
• Topics: 2.3 million
• Messages: 100 billion [msg/day]
• Message loss is not acceptable
• Order needs to be guaranteed

Performance (1KB message):
• Throughput: 100 thousand [msg/s]
• Latency: 5 [ms]

Graph is from: [https://yahooeng.tumblr.com/post/150078336821/open-sourcing-pulsar-pub-sub-messaging-at-scale](https://yahooeng.tumblr.com/post/150078336821/open-sourcing-pulsar-pub-sub-messaging-at-scale)
Scalability

Just adding Brokers/Bookies increases serving/storage capacity

- Add Brokers when more producer/consumer’s requests need to be served
- Add Bookies when more messages need to be stored
Multi-tenancy

Multiple services can share one Pulsar system

- Just use Pulsar as a “Tenant” → no need to maintain own messaging system
- Authentication/Authorization mechanism protects messages from interception
Geo-replication

Pulsar can replicate messages to another cluster
1. Producers only have to publish messages to Pulsar in the same data center
2. Pulsar asynchronously replicates messages to another cluster
3. Consumers can receive messages from the same data center
Pulsar Functions

- **Lightweight compute framework** (e.g., AWS lambda, Google Cloud Functions)
- **No need to launch extra systems** (e.g., Apache Heron, Apache Storm, Apache Spark)
- All you have to do is to implement your logic and deploy it to Pulsar cluster
- Java and Python are available and also other languages will be supported in the future

```python
def process(input):
    return input + '!
```
Pulsar on Kubernetes

- Pulsar can be easily deployed in Kubernetes clusters (e.g., AWS, GKE)
- See [https://pulsar.incubator.apache.org/docs/latest/deployment/Kubernetes/](https://pulsar.incubator.apache.org/docs/latest/deployment/Kubernetes/)
## Apache Pulsar vs. Apache Kafka

<table>
<thead>
<tr>
<th></th>
<th>Apache Pulsar</th>
<th>Apache Kafka</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Architecture image:</strong></td>
<td>Broker 1 (replica)</td>
<td>Broker 2 (leader)</td>
</tr>
<tr>
<td></td>
<td>Broker 2 (leader)</td>
<td>Broker 3</td>
</tr>
<tr>
<td></td>
<td>Bookie 1</td>
<td>Bookie 2</td>
</tr>
<tr>
<td></td>
<td>Bookie 2</td>
<td>Bookie 3</td>
</tr>
<tr>
<td></td>
<td>seg-1</td>
<td>seg-2</td>
</tr>
<tr>
<td></td>
<td>seg-1</td>
<td>seg-2</td>
</tr>
<tr>
<td></td>
<td>seg-3</td>
<td>seg-3</td>
</tr>
<tr>
<td></td>
<td>seg-1</td>
<td>seg-3</td>
</tr>
<tr>
<td></td>
<td>seg-2</td>
<td>seg-3</td>
</tr>
<tr>
<td></td>
<td>seg-3</td>
<td>seg-3</td>
</tr>
<tr>
<td><strong>Replication unit</strong></td>
<td>Segment (more granular than Partition)</td>
<td>Partition</td>
</tr>
<tr>
<td><strong>Data distribution</strong></td>
<td>Distributed and balanced across all Bookies</td>
<td>Kept in only leader and replica Brokers</td>
</tr>
<tr>
<td><strong>Max capacity</strong></td>
<td>Not limited by any single node</td>
<td>Limited by the smallest broker’s capacity</td>
</tr>
<tr>
<td><strong>Data Rebalancing when scale</strong></td>
<td>Not required</td>
<td>Required</td>
</tr>
<tr>
<td><strong>Geo-replication</strong></td>
<td>Built-in</td>
<td>Need to start an extra process: MirrorMaker</td>
</tr>
</tbody>
</table>

**Copyright (C) 2018 Yahoo Japan Corporation. All Rights Reserved.**
OpenMessaging Benchmark

http://openmessaging.cloud/docs/benchmarks/

• A suite of tools that can easily compare various messaging systems

https://info.streaml.io/benchmarking-streaming-messaging-platforms

• Pulsar showed better performance than Kafka
How does Yahoo! JAPAN use Apache Pulsar?
Agenda

1. What is Apache Pulsar?
2. Why is Apache Pulsar useful?
3. How does Yahoo! JAPAN uses Apache Pulsar?
   › About Yahoo! JAPAN
   › System architecture
   › Self-service tool
   › Case 1 - Notification
   › Case 2 - Job queuing
   › Case 3 - Log collection
Yahoo! JAPAN – 3 numbers

100+ services

150,000+ servers (bare-metal)

71,300,000,000 PV/month (average in 2017)

image: aflo
Why Yahoo! JAPAN chose Pulsar?

- Large number of users
- Large number of services
- Sensitive/mission-critical messages
- Multiple data centers

Pulsar meets all these requirements!
System architecture in Yahoo! JAPAN

Service A (Node.js)

Service B (Java)

Broker

WebSocket Proxy

Bookie

ZK

Geo-replication

Broker

WebSocket Proxy

Bookie

ZK

Service C (C++)

Prometheus + Grafana

Collect metrics + Visualize

West

East
Monitoring by Prometheus (1/2)
Monitoring by Prometheus (2/2)
Self-service tool

Web UI to manage tenants, namespaces and topics (Yahoo! JAPAN internal)

Create tenant

Create namespace

See topic stats
Case 1 – Notification of contents update

- Various contents files are pushed from partner companies to Yahoo! JAPAN
- Notification is sent to topic when contents are updated
- Once services receive a notification, they then fetch contents from file server

Partner Companies

Weather, map, news etc.

FTP server

Producer

Pulsar

Service A

Service B

Service C

① Send notification

② Receive notification

③ Fetch content files
Case 2 – Job queuing in mail service

- Indexing of mail can be heavy → let’s execute it asynchronously
- Producers register jobs to Pulsar
- Consumers take jobs from Pulsar at their own pace
Case 3 – Log collection (under development)

- Collect logs from all platforms: PaaS, CaaS, …
- Filtering / preprocessing by Pulsar Functions
- Finally send to other DB/platforms: HBase, Prometheus, twilio, …
Conclusion
Conclusion

- Apache Pulsar
  - Fast, Durable, Scalable pub-sub messaging
  - Has useful built-in features (Geo-replication, Pulsar Functions, etc.)

- Feature releases (2.1.0)
  - Go Client
  - Pulsar IO connector framework

- Documents
  - [https://pulsar.incubator.apache.org/](https://pulsar.incubator.apache.org/)

- Contact
  - [https://apache-pulsar.slack.com/](https://apache-pulsar.slack.com/)
  - users@pulsar.incubator.apache.org
References

• “Open-sourcing Pulsar, Pub-sub Messaging at Scale”
  https://yahooeng.tumblr.com/post/150078336821/open-sourcing-pulsar-pub-sub-messaging-at-scale
• “Linked In Stream Processing Meetup – Apache Pulsar”
• “Benchmarking Enterprise Streaming Data and Message Queuing Platforms”
  https://info.streaml.io/benchmarking-streaming-messagingPlatforms
• “Comparing Pulsar and Kafka: how a segment-based architecture delivers better performance, scalability, and resilience”
  https://streaml.io/blog/pulsar-segment-based-architecture/
Apache BookKeeper

**Apache BookKeeper**: Open-source distributed write-ahead log system
- Durable, Scalable, Consistent, Fault-tolerant, Low-latency

The number to replicate is configurable (here 2)

Data is evenly distributed and balanced across bookies

I/O paths for writes/reads are separated
## Existing messaging systems

<table>
<thead>
<tr>
<th>Developed since</th>
<th>Apache Kafka</th>
<th>RabbitMQ</th>
<th>Apache ActiveMQ</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2011 LinkedIn</td>
<td>2007 Rabbit Technologies</td>
<td>2004 LogicBlaze</td>
</tr>
<tr>
<td><strong>Server-side languages</strong></td>
<td>Scala</td>
<td>Erlang</td>
<td>Java</td>
</tr>
<tr>
<td><strong>Client-side languages</strong></td>
<td>Java 3rd party libraries</td>
<td>Erlang .NET/C# 3rd party libraries</td>
<td>Java Multiple languages over various protocols</td>
</tr>
<tr>
<td>Protocol</td>
<td>Original (binary protocol over TCP)</td>
<td>AMQP STOMP MQTT JMS</td>
<td>AMQP STOMP MQTT JMS(1.1) OpenWire</td>
</tr>
</tbody>
</table>
Type 1 - Exclusive

- Only 1 Consumer is allowed to connect
- Default subscription type

Diagram:
- Producer
- Topic
  - sub-A (Exclusive)
  - sub-B (Exclusive)
- Consumer A
- Consumer B
- Consumer C
Type 2 - Shared

- Multiple Consumers are allowed to connect
- Messages are distributed to each consumer in round-robin manner
  → Useful for load balancing
Type 3 - Failover

- Multiple Consumers are allowed to connect
- Only one of them receives messages
- If the primary consumer is down, another consumer starts to receive messages

→ Useful for active / standby
Multi-tenancy

- Multiple services can share one Pulsar system
- Each tenant can manage their namespaces/topics by themselves (e.g., ACL)

```
persistent://tenant/namespace/topicname
```

- Super user (Pulsar maintainer)
- Tenant admin (Pulsar user)
- Create tenant
- Register as admin
- Create and configure namespaces (e.g., ACL)

---

service A

---

service B