PIP-XX: Consumer Filters

- Status: Proposed
- Author: Andre Kramer andre.kramer@softwareag.com Github: andrekramer1
- Pull Request
- Mailing List discussion
- Release:

At Software AG, we have prototyped property (key / value) based message filtering on consumer subscriptions in the Pulsar broker. This feature was also requested as Issue 3302 but the mitigation suggestion of using Pulsar Functions, which imply management and runtime overheads and an extra topic, or the alternative of publishing to multiple (sub)topics are not always adequate mitigations to lack of simple consumer subscription filtering.

Motivation

In Pulsar, consumers receive all messages published on a topic, even if only interested in messages with certain properties. Always delivering all messages and having the consumer client application filter means all messages must be sent over the network, processed and acknowledged remotely. Other message brokers and messaging standards (such as Java's JMS message selectors) typically allow a filter to be specified on client subscription, allowing the message broker to filter on behalf of the client subscriber and deliver a subset of the messages over the wire.

In some cases, a separate topic can be used for each subscription filter, but if there are a very large number of possible filters then capacity as well as the management and processing overheads of topics (or any Pulsar Functions) are an issue. It therefore seems sensible to extend consumer subscriptions to perform an additional level of selection that is not directly driven by the publisher and does not require additional Functions and/or topics to be created for consumers.

Feature

Our proposal is to allow the publisher to "tag" messages with one or more additional message properties that can be used to selectively filter by. The consumer (on subscription) should be able to specify which tags to filter on, specified by special consumer “meta data” (using the meta data key value pairs in consumer properties).

The implementation is itself configurable, allowing both for no filtering and for alternative property-based schemes to be implemented and configured in Pulsar brokers, using a new configuration time "ConsumerFilter" dynamic factory mechanism. Our default "tag" based scheme, is explicit about which properties are used for filtering so as to more safely use existing message properties and consumer meta data. Two types of meta data tags are provided to allow both simple AND (all match) as well as OR (any match) filter expressions on message properties.

Design

Our implemented design enables broker side "tag" based filtering of messages on selected message properties by allowing consumers to specify a subset of message properties as being “tags” to filter on, using existing consumer (meta data) property API (and underlying protocol). Message properties which have a key beginning with the string "tag" are matched by value against the consumer subscription properties that are specified on subscription with meta data string keys starting with either "anytag" or "alltag". What follows “tag”, “anytag”, “alltag” textually is immaterial except to allow multiple key / values to be enumerated (e.g. “tag1”, “tag2” etc).

An "anytag" matches successfully if any of the messages "tag" properties has the same property value as the consumer property value. This allows "or" matching.

e.g. Say a consumer subscribes with an "anytag" property with "anytag1" of "urgent" and another with key "anytag2" and value "compressed". Then a message with property "tag0" as key and "urgent" as value will match as would a message with "tagA" / "compressed".

To subscribe the consumer:

```
Consumer<String> consumer = client.newConsumer(Schema.STRING)

                     .topic("sometopic")

                    .property("anytag0", "urgent") // A filtering meta data tag.

                    .property("anytag1", "compressed") // Another filtering meta data tag.

                    .subscribe();
```

To send a message with a tag that matches:

```
Producer<String> producer = pulsarClient.newProducer(Schema.STRING)

                    .topic(topic)

                    .create();

producer.newMessage()

                   .property("tag0", "urgent") // tag with a matching property value.

                   .value("my-matching-message")

                   .sendAsync();
```

An "alltag" will only match if all other alltags in the consumer subscription meta data properties also can be matched. This allows "and" matching of multiple values.

e.g. Say a consumer subscribes with an "alltag" property with "alltag1" of "urgent" and another with key "alltag2" and value "compressed". Then a message with property "tag0" as key and "urgent" and property "tag1" of "compressed" as value will match while a message with "tag0" of "urgent" and "tag1" of "uncompressed" property values will not.

```
Consumer<String> consumer = client.newConsumer(Schema.STRING)

                    .topic("sometopic")

                    .property("alltag0", "urgent") // A filtering meta tag.

                    .property("alltag1", "compressed") // Additional filtering meta tag.
```

```
                   .subscribe();
```

will match on the following message:

```
producer.newMessage()
                   .property("tag0", "urgent") // A tag property value.
                   .property("tag1", "compressed") // A second tag property value.
                   .value("my-matching-message")
                   .sendAsync();
```

But the following will not match:

```
producer.newMessage()
                   .property("tag0", "urgent") // A tag property value.
                   .property("tag1", "uncompressed") // A second tag property value.
                   .value("my-not-matching-message")
                   .sendAsync();
producer.newMessage()
                   .property("tag0", "urgent") // A tag property value.
                   .value("my-other-not-matching-message")
                   .sendAsync();
```

If the consumer subscribes with both anytags and alltags then at least one anytag and all of the alltags must match tags in the message (with overlap allowed) for the message to be passed by the filter.

The default implementation matches only on properties that have keys beginning with "tag" to help avoid unintended matching. A new Java ConsumerFilter interface and factory allows other property-based filtering schemes to be implemented and deployed.

Apart from message tag properties starting with (lowercase) "tag" and consumer tag subscription meta data properties starting with "anytag" and "alltag" there are no other requirements or conventions on tag naming and usage. For example, tags could use more meaningful keys such as "tag_priority" and namespace tag values such as values "prority_urgent" and "priority_low".

Another possibility is to use values such as "priority=urgent" and match by text on such categories or classes (with keys just enumerating "tag0","tag1" etc as in examples above).

These suggestions are a convention only and not checked or enforced in any way. In the examples above we have used "0" and "1" but other unique strings (such as "A","B" etc) could have been used.

Implementation

A new interface "ConsumerFilter" (org/apache/pulsar/broker/service/ConsumerFilter.java) with a factory (ConsumerFilterFactory.java) configured via settings in org/apache/pulsar/broker/ServiceConfiguration.java allows consumer filtering to be configured was added. Our suggested default implementation of this interface (behaviour described in detail above) is defined by the "ConsumerTagFilter" class.

In order to minimise code changes we've limited our modifications mainly to two existing classes. The consumer abstract base class org.apache.pulsar.broker.AbstractBaseDispacher is modified to use the above interface to filter both batches and single messages (conditionally only if consumer filtering is enabled) in method filterEntriesForConsumer(), keeping this method's code as similar as possible to the current implementation.

For filtering entries in a batch, we duplicated an existing method in org.apache.pulsar.client.impl.RawBatchConverted and adapted it for our needs. The new static "filter" method in this class was kept as similar as possible to the "rebatchMessage" it was based on. It's expected that on average most messages will be both batched and filtered out (on average if a filter is in effect) and we've therefore not attempted to optimise other code paths. Non-filtered usage should not be impacted except for a simple check.

A handful of other code files changes in the persistent and nonpersistent sub modules of org.apache.pulsar.broker.service are just single point modifications to pass the consumer object through so that the consumer meta data properties are available for use in filtering.

Testing

The org.apache.pulsar.broker.service.ConsumerFilterTest unit test tests the ConsumerFilter implementations. Both "anytag" and "alltags" combinations are tested by simple combinations of these meta data properties.

A more comprehensive ProducerConsumerBase sub-class test (org.apache.pulsar.client.api.ConsumerFilteringTest) tests sending and receiving messages with variations on the percentage of messages with tag properties that should match the subscription filter. All and no matches as well as 10% and 1% expected matches are tested.

Licenses

Developed by Software AG under the Apache License Version 2.0.

We are looking forward to any feedback.