Consumer Client Re-Design

Motivation

We've received quite a lot of feedback on the consumer side features over the past few months. Some of them are improvements to the current consumer design and some are simply new feature/API requests. I have attempted to write up the requirements that I've heard on this wiki - Kafka 0.9 Consumer Rewrite Design
This would involve some significant changes to the consumer APIs, so we would like to collect feedback on the proposal from our community. Since the list of changes is not small, we would like to understand if some features are preferred over others, and more importantly, if some features are not required at all.

Thin consumer client:

1. We have a lot of users who have expressed interest in using and writing non-java clients. Currently, this is pretty straightfoward for the SimpleConsumer but not for the high level consumer. The high level consumer does some complex failure detection and rebalancing, which is non-trivial to re-implement correctly.
2. The goal is to have a very thin consumer client, with minimum dependencies to make this easy for the users.

Central co-ordination :

1. The current version of the high level consumer suffers from herd and split brain problems, where multiple consumers in a group run a distributed algorithm to agree on the same partition ownership decision. Due to different view of the zookeeper data, they run into conflicts that makes the rebalancing attempt fail. But there is no way for a consumer to verify if a rebalancing operation completed successfully on the entire group. This also leads to some potential bugs in the rebalancing logic, for example, https://issues.apache.org/jira/browse/KAFKA-242
2. This can be mediated by moving the failure detection and rebalancing logic to a centralized highly-available co-ordinator - Kafka 0.9 Consumer Rewrite Design

We think the first two requirements are the prerequisite of the rest. So we have proceeded by trying to design a centralized coordinator for consumer rebalancing without Zookeeper, for details please read here.

Allow manual partition assignment

1. There are a number of stateful data systems would like to manually assign partitions to consumers. The main motive is to enable them to keep some local per-partition state since the mapping from their consumer to partition never changes; also there are some use cases where it makes sense to co-locate brokers and consumer processes, hence would be nice to optimize the automatic partition assignment algorithm to consider co-location. Examples of such systems are databases, search indexers etc
2. A side effect of this requirement is wanting to turn off automatic rebalancing in the high level consumer.
3. This feature depends on the central co-ordination feature since it is cannot be correctly and easily implemented with the current distributed co-ordination model.

Allow manual offset management

1. Some systems require offset management in a custom database, at specific intervals. Overall, the requirement is to have access to the message metadata like topic, partition, offset of the message, and to be able to provide per-partition offsets on consumer startup.
2. This would require designing new consumer APIs that allow providing offsets on startup and return message metadata with the consumer iterator.
3. One thing that needs to be thought through is if the consumer client can be allowed to pick manual offset management for some, but not all topics. One option is to allow the consumer to pick one offset management only. This could potentially make the API a bit simpler
4. This feature depends on the central co-ordination feature since it is cannot be correctly and easily implemented with the current distributed co-ordination model.

Invocation of user specified callback on rebalance

1. Some applications maintain transient per-partition state in-memory. On rebalance operation, they would need to “flush” the transient state to some persistent storage.
2. The requirement is to let the user plugin some sort of callback that the high level consumer invokes when a rebalance operation is triggered.
3. This requirement has some overlap with the manual partition assignment requirement. Probably, if we allow manual partition assignment, such applications might be able to leverage that to flush transient state. But, the issue is that these applications do want automatic rebalancing and might not want to use the manual partition assignment feature.

Non blocking consumer APIs

1. This requirement is coming from stream processing applications that implement high-level stream processing primitives like filter by, group by, join operations on kafka streams.
2. To facilitate stream join operations, it is desirable that Kafka provides non-blocking consumer APIs. Today, since the consumer streams are essentially blocking, these sort of stream join operations are not possible.

3. This requirement seems to involve some significant redesign of the consumer APIs and the consumer stream logic. So it will be good to give this some more thought.

Comments (Guozhang)

- Currently Kafka have two types of consumers: high-level consumer and simple consumer. In simple consumer user can specify broker-partition and offset, but there is no failover/re-balance support. So users with requirements 3 and 4 but no requirement for group/re-balance would more prefer to use the simple consumer. Basically the high-level consumer provides the following main functionalities against simple consumer:
  - Auto/Hidden Offset Management
  - Auto(Simple) Partition Assignment
  - Broker Failover => Auto Rebalance
  - Consumer Failover => Auto Rebalance
  - If user do not want any of these, then simple consumer is sufficient
  - If user want to control over offset management with others unchanged, one option is to expose the current ZK implementation of the high-level consumer to users and allow them to override; another option is to change the high-level consumer API to return the offset vector associated with messages
  - If user want to control partition assignment, one option is to change the high-level consumer API to allow such config info be passed in while creating the stream; another option is ad-hoc: just make a single-partition topic and assign it to the consumer.
  - If user just want the automatic partition assignment be more "smart" with co-location consideration, etc, one option is to store the host /rack info in ZK and let the rebalance algorithm read them while doing the computation.

* Bottom line: complicating the simple consumer will risk its compatibility with the current applications, hence might not be a good option compared with patching the high-level consumer

- User specified callback upon rebalancing is good to have, while users are responsible for decreasing the re-balancing performance with a heavy callback function.
- For non-blocking primitive support, again it depends on what user "really" wants:
  - Currently one stream only contains messages from one or more partitions of the same topic, and has the option to return an exception when the next() call has timed out. If what user want is simply want a stream from multiple topics, one option is to implement a special fetcher that put messages from different topics in a shared queue for the end iterator.
  - If what user want is to support some stream join operations, which are usually windowed (i.e. only two messages arrive within a relatively small window size can be correlated), then the current timeout mechanism of KafkaMessageStream is nearly sufficient.