Roadmap - port existing system tests

The existing system tests consist of a small number of heavily parameterized test suites. As part of KIP-25, it is necessary to have a clear understanding of what functionality is currently present, and what of that functionality needs to be ported.

As of this writing, there are five test suites within system_tests. The replacement proposal is summarized as follows.

Tracking JIRA: https://issues.apache.org/jira/browse/KAFKA-2256

Testsuite	Port?	JIRA	Notes
broker_failure	N	None	Written in bash, no longer runs.
producer_pef	Y	None	Writting in bash, no longer runs. kafka_benchmark.py will provide superset
replication_testsuite	Y	https://issues.apache.org/jira/browse/KAFKA-2257	replication_test.py provides a set of representative examples
offset_management_testsuite	Y	https://issues.apache.org/jira/browse/KAFKA-2259	(has 2 test cases) Use new consumer in port
mirror_maker_testsuite	Y	https://issues.apache.org/jira/browse/KAFKA-2258	Fairly straight port

See below for more notes on the current state of each existing system test suite.

Parent JIRA of the existing system tests:

https://issues.apache.org/jira/browse/KAFKA-440

Test cases and associated parametrizations:

https://cwiki.apache.org/confluence/display/KAFKA/Kafka+system+testcases

https://cwiki.apache.org/confluence/display/KAFKA/Kafka+replication+system+tests

broker_failure

Brief description:

Written in bash, these tests validate an end-to-end setup which includes two broker clusters, mirror maker, producer and consumer.

Randomly selected source brokers, target brokers, and mirror makers are shut down cleanly while messages are produced in the background.

This test does not run - the calls to the producer performance and mirror maker command line tools have not been updated to match changes on the java side.

Proposal:

don’t port, this is covered in replication and mirror maker test suites

producer_perf

Brief description:

Written in bash, this suite has two wrappers around ProducerPerformance.java which produce 2e6 messages, but no regression checks on actual performance numbers. Validates size of log agains offset (relevant in kafka 0.7 when log offsets were specified in bytes). Both tests here no longer function.

Proposal:

kafka_benchmark_test.py more than covers this.

replication_testsuite

Concurrently bounce leader, controller, or follower with hard, clean, or soft “kill” while producing with ProducerPerformance in the background. Validates that produced messages are consumed, and compares checksums of per-topicPartition log segments across replicas.

Note: Mixed in here is another type of test called “log retention” (test cases 4001 through 4018). These tests display some information about smallest offsets of log segments but do no validation.

Proposal:

Drop log retention test cases and port specific subset of replication tests.

3 brokers, 3 topics, 3 partitions, 3 replicas on each topic min.insync.replicas == 2 ack -1 failures: [leader, follower, controller] X [clean, hard, soft]
One test cases with with ack 1
One test case with compression toggled on

How does message validation work in old system tests?

Loads message ids from producer log and consumer log, checks for messages produced but not consumed (dropped messages). If producer ack == 1, check whether message loss is within acceptable threshold. Otherwise dropped messages fail this validation step.

How does broker validation work in old system tests?

Validate broker log checksum in source and target brokers.

How does log offset checksum validation work?

foreach topicPartition, for each replica:

compute checksum of broker logs

validate that checksums are identical across replicas

offsest_management_testsuite

in the face of consumer failure (offset_management)

Brief description:

These tests validate tracking of consumer offsets within the __consumer_offsets topic by bouncing console consumers *and* the leaders of (partitions of) the __consumer_offsets topic.

Proposal:

Port something quite similar to this, but use new consumer.

Note that this may break initially while the new consumer is still under heavy development.

What do these tests do?

SETUP

Set up a Zk and Kafka cluster.

Produce messages to a multiple topics - various partition counts.

Start one or more consumer groups to read various subsets of above topics.

Offsets stored in Kafka “__consumer_offsets” topic. 3 replicas, 2 partitions

TEST

stop leaders of __consumer_offsets

bounceConsumers = True/False

(re)start leaders of __consumer_offsets

VALIDATE

Verify that there are no duplicate messages or lost messages on any consumer group.

Producer dimensions : mode:sync, acks:-1, comp:0

how does validation work here?

kafka_system_test_utils.validate_data_matched_in_multi_topics_from_single_consumer_producer(self.systemTestEnv, self.testcaseEnv, replicationUtils)

This is the same as other message validation

7001 uses one producer and one consumer

7002 one producer, 2 consumer groups, first with 1 consumer, second with 4 consumers

mirrormaker_testsuite

Brief description:

Propagate data between two broker clusters with one or more mirror maker processes while periodically bouncing mirror maker. Validate messages and broker checksums.

Original JIRA(s):

https://issues.apache.org/jira/browse/KAFKA-488

What do these tests do?

SETUP

Start up source broker cluster and target broker cluster

Start up producer (using ProducerPerformance.java)

Start up 1 or 2 mirror maker instances, both with the same source and target clusters.

(consumers are in same consumer group)

TEST

Bounce (or not) mirror maker processes.

Consume messages from the target cluster.

VALIDATE

Validate messages

How does data message validation work?

Loads message ids from producer log and consumer log, checks for messages produced but not consumed (dropped messages). If producer ack == 1, check whether message loss is within acceptable threshold. Otherwise dropped messages fail this validation step.

Validate broker log checksum:

Same as replication tests

Port?

Yes, but add more failure modes.

Just run with ack == -1