System Test Improvements

PROPOSAL

The following is a proposal to update and improve the Kafka system tests.

Motivation

The existing system tests have been around for a few years, but not as much subsequent development effort has gone here as is needed. Consequently, technical debt has accrued. For each new feature, we often have a standalone program to test it, but generally those programs are not integrated into system tests and tend to become obsolete over time.

As Kafka matures, our system tests need to graduate to first-class citizens so that we can be more confident in the stability of the overall system as well as compatibility etc.

Current system tests are:

- Unstable - relatively little maintenance effort has gone in, and many are now redundant or simply too unstable to run successfully.
- Incomplete - there are significant gaps in coverage
- Hard to read - Community members have complained that the existing framework is hard to read/understand and therefore difficult to develop new tests against.

What do we want out of system tests going forward?

From “things we can test” perspective, we need convenient ways to do:

- Sanity checks for normal conditions and operations (e.g. soft bouncing of services)
- Failures (e.g. bounce/hard bounce/chaos tests)
- High load
- Network errors or failures (partitions, loss, or high latencies)
- Run against different combinations of versions to test version compatibility.

From a dev perspective, we need:

- Easy to run ‘in the cloud’ or locally without tweaky config changes,
- Easy-ish to automate (nightly runs)
- Possible to run multiple tests in parallel on different machines.
- Nice reporting, with regression checks against performance statistics.
- More readable/better structured to encourage community members to develop tests.

Plan of Action

The rough plan to move forward is as follows:

- ~~Pick a new framework, or develop one~~
- We chose to use ducktape (https://github.com/confluentinc/ducktape - see notes below)
- Identify “MVP” tests - i.e. a set of high priority tests to fill in major gaps in our existing system test coverage (see further notes below)
- Identify “long tail” of less urgent tests.
- Stabilize and prune existing system tests so they can be run regularly until they are ported(?)

Framework Choice

We evaluated Zopkio (https://github.com/linkedin/Zopkio) which is used internally at LinkedIn for Samza and other projects, and Confluent’s ducktape (https://github.com/confluentinc/ducktape) which is used in system tests for the Confluent Platform (https://github.com/confluentinc/muckrake)

and concluded that ducktape better matched our use case (see comparison table below). Continued development on ducktape is needed, but it is far enough along to be a useful tool for a wide variety of system tests.

	ducttape	zopkio
runnable on cloud or locally (provisioning capability)	yes	no
cluster management	yes	no
run tests in parallel	not implemented, but written with this use case in mind	use of globals makes this harder
easy to read/understand	somewhat	somewhat
good reporting	yes	yes

regression	not yet	yes (at least, naarad claims to support this)
automatable	yes - has flexible test discovery mechanism	test suites run in a self-contained way, but a script to drive multiple test suites would still be needed

Test Coverage

Among other things, we want to include those standalone programs into our system test so that we have reasonable coverage on each important feature.

1. Performance: Need to collect the performance numbers regularly and make sure there is no significant regression.

Setup: 3-node cluster, replication factor 3, 1 topic with 3000 partitions.

1.1 ProducerPerformance:

minimal: measure the throughput of a single producer with a few different configurations (ack =1/all, compression=none/gzip)

nice to have: extend the test to multiple producers

Note: There is system_test/producer_perf, but it's based on the old ProducerPerformance. We need to either replace it or create a new test and delete the old one.

1.2 ConsumerPerformance:

minimal: measure the throughput of a single consumer with a few different configurations (compression=none/gzip)

nice to have: extend the test to multiple consumers

1.3 TestEndToEndLatency: This test measures the end to end latency from the time a message is produced to the time the message is consumed.

Probably need to get KAFKA-1983 fixed first.

2. Feature correctness: Need to make sure there are no errors when testing existing features.

Setup: All tests can be done on a single node cluster.

2.1 TestLogCleaning: This test stress tests the log compaction logic.

2.2 TestOffsetManager: Need to understand a bit more how the test works and its relationship with the offset management suite in system_test.

2.3 KafkaLog4jAppender: Somehow need to make sure that it works. Perhaps we also need to improve the appender itself.

3. Admin Tools testing:

3.1 partition reassignment: load some data; trigger partition assignment, bounce the broker a few times, make sure partition reassignment can complete in the end and there is no data loss.

3.2 leader balancing: bounce broker, make sure leaders balanced after auto-leader balancing time and there is no data loss.

3.3 delete topic

4. System tests: We need to pick a subset of important existing tests. Make sure they are stable and run regularly.

4.1 a few leader failure tests

4.2 a few controller failure tests

4.3 a few mirror maker tests

4.4 offset management suite (need to figure out its relationship with TestOffsetManager)

5. Compatibility tests:

5.1 Wire protocol: This will get better after we move all requests to the new protocol definition. However, in addition to making sure that the existing wire protocol doesn't change, it would be useful to also check the semantics, e.g. whether the client is getting the same response value as before.

5.2 Upgrades: Whether we can upgrade the broker with existing data and configs from an older version.