Generic Load Driver TODO

# Path To Glory

* Hierarchy of scripts.
* send must stay up indefinitely until shutdown – especially for embedded broker.
* --background
  + Test launching the process in background in EC2.
    - From the ec2 nodes
    - Remotely from the em client
* Come up with the local scenario that would allow me to start load, make sure it finishes, and then collect results:
  + Start load on n instances/machines
  + Make sure the load finished on n instances/machines
  + Collect results from n instances/machines
* Repeat the steps above remotely, with ActiveMQ, get a burst and measure it.
* Repeat the steps above remotely, with the embedded broker, get a burst and measure it.
* Repeat the release process and measurements for the following:
  + Introduce total heap and free heap + tests.
* Apropriately sized producers and consumers, now they’re too small.
* Capture the state of the art of /nfs/load/common, send and receive, check them in somewhere under em (examples, etc) and document them.

# TODO

* **Sampler**
  + Identify the current sampler limits
  + Keep in mind that I want to zip and ship results remotely.
  + Bolt it in and throw away the current implementation.
  + Test locally in real conditions – pump messages into a local ActiveMQ instance and go through the documentation smoke tests.
  + Test in amazon with AMQ.
* **System-wide Metrics.** Aggregate base classes for metrics that have the same source MXBean.
* **From activeMQ load testing:** Count messages based on their ID.
* **Switch to cliff.** 05/03/15.
* **The Hierarchical Repository Key Reading is Slow.** Large variation in key loading time, locally. Try to load it in parallel and also look into how I am reading the keys from the file – the way I am doing it might be inefficient.
* **Compression.** A generic compression service? Apply it to the cache only?
* **Multiple independent agents.** This way I can increase the amount of load.
* **log4j**. Externalize log4j.xml so I can modify it directly, don't have to unjar for it. Ideally, the debugging should be turned on by --debug.
* Make it easy to dump the runtime configuration – implement it as a “dry-run” where I only have to specify “dry-run” somehow and not modify anything else.
* Introduce a CacheLoadStrategy and make all cache load strategies subclasses of that, the same Send and Receive are subclasses of JmsLoadStrategy.
* Implement support for --provider Currently we assume Infinispan for “keyvalue” and ActiveMQ for “message”.
* Move the Content’s StorageStrategy to Configuration.
* Make sure cld load embedded hierarchical fills up the embedded hash with the content of the hierarchical repo (on one and multiple threads).
* Isolate HierarchicalStorageStrategy. toHex(…) into its own class (or Util), test the heck out of it and update https://home.feodorov.com:9443/wiki/Wiki.jsp?page=SHA1
  + 10 different values – calculate hex sha1 with git and write 10 unit tests for them. Use / \* and other characters that are not FS-friendly
* Unify KeyStore and StorageStrategy.
  + Rename KeyStore to LocalStore – it stores keys and values.
  + Verify which of the top-level ConfigurationImpl attributes still make sense.
  + Keep all strategies in the Configuration and not in commands
* Exact distributions of max-operations between runner threads.
* If CLI starts in background, it cannot be controlled with System.in.read – find an alternative solution.
* All “strategies” should end in Strategy.
* Understand the commented out ReadTest tests, and either uncomment or write equivalent ones.
* The ReadThenWriteOnMiss.synthethicValue is created with the default value size of 1024. Make it more user friendly, hint somehow that the values in the cache are larger (or of different size).
* Move Util.getInstance() and associated test to Nova Ordis Utilities. \* Dynamic Logging in interactive mode until I find a way to get the information I need from infinispan and not rely on runtime.
* Reconcile the new style of load manager that preconfigures the operation inside, with the old style SingleThreadedRunner op.initialize(ThreadLocalRandom.current(), lastWrittenKey, keyStore); Old logic in "OldLoad" and "OldSingleThreadedRunner"

saving keys into the file bug: because the collector is doing it asynchronously, some of the keys will stay in memory until "press the key" so we may end up with less keys on disk.

* Change the name of Connect to "interactive" or something similar.