Generic Load Driver

User Manual

Ovidiu Feodorov <ovidiu@novaordis.com>

Table of Contents

Overview 3

Installation 4

Usage 5

Load Random Content into the Cache 5

Replay Previously Loaded Random Content 6

Dump Cache Content into a Local Hierarchical File Storage 7

Delete Cache Content 7

Load the Content of a Hierarchical File Storage into the Cache 7

Release Procedure 8

TODO 10

# Overview

gld (Generic Load Driver) is a Java-based command line tool that can be used to stress-test a cache instance. It works by generating random keys or key/value pairs and sending these in the form of read or write operations, on multiple threads, into the cache, while collecting various statistics.

gld can also be configured to save locally the random generated content and use it for further testing.

gld can be used as a high throughput cache loader that reads pre-staged local content and sends it into the cache.

For a description of the most common usage scenarios, see the “Usage” section below.

# Installation

As "cld" user or root:

cd /opt/cld

unzip .../cld-<version>.zip

This will create a /opt/cld/cld-<version> directory.

Link /opt/cld/current to /opt/cld/cld--<version>

Add /opt/cld/current/bin to your PATH.

# Usage

Comprehensive in-line help can be generated at any time by running cld without any argument.

## Load Random Content into the Cache

To Write Random Key/Values into the Cache (and also Store Keys Locally for Replay)

cld load \

--nodes <infinispan-node-ip>:<hotrod-port> \

--output ./statistics.csv \

--threads <thread-count> \

--max-operations <entry-count> \

--key-size 50 \

--value-size <value-size-in-bytes> \

--read-to-write 0 \

--keystore-file ./cld.keys

The utility will send <entry-count> key/value pairs on <thread-count> threads. It will also store the keys into the local ./keys.txt file. The file can then be used for replay.

The default load strategy (not explicitly configured) is WriteThenRead.

Example:

cld load --nodes localhost:11222 --output ./statistics.csv --threads 5 --max-operations 5000 \

--key-size 50 --value-size 512000 --read-to-write 0 --key-store-file ./cld.keys

This command ends up loading 5,000 random key/value pairs into the cache. The keys are 50 characters long, the values are 512,000 characters long. Copy of the keys are stored in the local file ./cld.keys, which also contains 5,000 keys:

wc –l ./cld.keys

5000 cld.keys

## Replay Previously Loaded Random Content

To read from cache keys previously written into the local key file, run:

cld load \

--write-to-read 0 \

--output ./statistics.csv \

--nodes <infinispan-node-ip>:<hotrod-port> \

--threads <thread-count> \

--value-size <value-size-in-bytes> \

--key-store-file ./cld.keys

Example:

cld load --write-to-read 0 --output ./statistics.csv --nodes localhost:11222 \

--threads 20 --value-size 512000 --key-store-file ./cld.keys

If the cache did not evict any of the previously written content, we should get 100% hit ratio. If there were evictions or expirations, we’ll get a hit ratio lower than 100%.

To replay keys previously written into the local key file, using a "Read and if there is a Miss, Write" Strategy, run:

cld load \

--load-strategy ReadThenWriteOnMiss \

--output ./statistics.csv \

--nodes <infinispan-node-ip>:<hotrod-port> \

--threads <thread-count> \

--value-size <value-size-in-bytes> \

--key-store-file ./cld.keys

Example:

cld load --load-strategy ReadThenWriteOnMiss --output ./statistics.csv --nodes localhost:11222 \

--threads 20 --value-size 512000 --key-store-file ./cld.keys

If the cache did not evict any of the previously written content, we should get 100% hit ratio. If there were evictions or expirations, we’ll get a hit ratio lower than 100%, but cld will replace the missing content with random content.

## Dump Cache Content into a Local Hierarchical File Storage

cld content \

--nodes <infinispan-node-ip>:<hotrod-port> \

--storage-strategy hierarchical \

--root ./cache-content

Example:

cld content --nodes localhost:11222 --storage-strategy hierarchical --root ./cache-content

Cache reads will be executed on a single thread, the --threads configuration option does not have any effect on this command.

If the command was used to read the previously written 5,000 entries, we should get:

find ./cache-content -type f | wc –l

5000

## Delete Cache Content

## Load the Content of a Hierarchical File Storage into the Cache

cld load \

<infinispan-node-ip>:<hotrod-port> \

--output ./statistics.csv \

--threads <thread-count> \

--read-to-write 0 \

--storage-strategy hierarchical \

--root ./cache-content

Example:

cld load --nodes localhost:11222 --output ./statistics.csv --threads 5 --read-to-write 0 \

--storage-strategy hierarchical --root ./cache-content

# Release Procedure

Turn Console Logging to INFO

**Build the Local Release for Integration Testing**

./util/make-zip; ./util/install-locally --force

This will run the test suite and won’t be able to build the local release if the tests don’t pass.

**Run Integration Testing**

cd ~tmp

mkdir cld

cd cld

From a different terminal, start the Infinispan instance.

cd ~/runtime/jboss-datagrid-6.3.2/profiles/node01/

./node01.sh

Then:

cld version

Then walk through scenarios described in “Usage” above.

**Save/Restore from Hierarchical Storage (Embedded Cache)**

cld content --nodes embedded[50000] --storage-strategy hierarchical --root ./hierarchical

find ./hierarchical -type f | wc -l

cld load --nodes embedded --threads 10 --read-to-write 0 --storage-strategy hierarchical --root ./hierarchical

cld load --nodes localhost:11222 --threads 10 --read-to-write 0 --storage-strategy hierarchical --root ./hierarchical

**Release**

Run test suite – all tests must pass.

Write down the number of tests that pass.

214

Update pom with the version.

mvn clean

Check in. Specify version and the number of tests. Short summary of the features.

./util/make-zip; ./util/install-locally

Verify it’s the right version

cld version

Save target/cld-x.zip and prepare it for distribution.

Increase the version in pom and append –SNAPHSOT

svn ci -m "starting 1.0.10-SNAPSHOT"

# TODO

* 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.

* Create a SVN repository for it (or a GitHub?). Temporary location NOMBP1:/Users/ovidiu/projects/cache-load-driver
* Comb the code base and documentation and remove all references to Apple. That includes changing package name and "redis"
* Externalize log4j.xml so I can modify it directly, don't have to unjar for it.
* Change the name of Connect to "interactive" or something similar.