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

Perform JMS Load Testing in an em-managed Amazon EC2 environment 8

Create Producers and Consumer Instances 8

Start All Instances Required by the Load Test 8

Make Sure All Instances are On-line 9

Optionally Remove the Collocated Broker Storage Directory on Producers 9

Make Sure Brokers Started Fine 9

Log into the NFS Server and Configure Load Run Parameters 9

Remove All Previous Tests’ Artifacts 9

Start the Consumers 10

Start the Producers 10

Run the Test 10

Stop All gld Processes 10

Collect Statistics 10

Stop All Instances Involved 11

# 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

## Installation Pre-Requisites

### Java SDK

gld requires Java SDK version 1.7.

Download and install the latest JDK 1.7 following instructions specific to your system from Oracle web site: http://www.oracle.com/technetwork/java/javase/downloads/index.html

Once JDK has been installed on your system, export JAVA\_HOME in your environment and add ${JAVA\_HOME}/bin to your PATH.

## gld Installation

### Download the Installation Archive

Download the latest release from https://github.com/NovaOrdis/gld/releases

### Extract the Installation Archive

Extract the content of the gld installation zip into the directory conventionally used for external utilities. /opt or /usr/local are common choices. For the remainder of this document we will assume that the external utility directory is /opt.

The extraction process will create an “gld-<version>” top-level directory.

Assuming that the top level directory is gld-1.0.1, link to it with a generic link “gld”, as follows:

cd /opt

ln –s ./gld-1.0.1 ./gld

### Setup the Environment Variables

Set the GLD\_HOME environment variable to point to the generic “gld” link. This way, you will be able to upgrade by simply unzipping a new version and re-linking, without any environment modification.

Also adjust the PATH environment variable as follows:

export GLD\_HOME=/opt/gld

export PATH=${PATH}:${GLD\_HOME}/bin

# 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

# Perform JMS Load Testing in an em-managed Amazon EC2 environment

Build an Amazon EC2 environment containing at least an AMQ broker, as described in the em User Manual.

## Create Producers and Consumer Instances

Create as many producer and consumer instances as your load test requires, using the gld overlay:

em create --instance-type c4.xlarge --group sg-bb3222de \

--storage-size 10 p001-p010 c01-c10

# update the gld overlay configuration 1) the potential server list

em overlay gld p001-p010 c01-c10

## Start All Instances Required by the Load Test

Start the NFS server, brokers, producers and consumers

em status

em start nfs01

em start b01 b02 c01-c10 p001-p010

# wait several tens of seconds until the instances had a chance to start

em sync

## Make Sure All Instances are On-line

em status

em run b01 b02 c01-c10 p001-p010 -- uptime

## Optionally Remove the Collocated Broker Storage Directory on Producers

em run p001-p010 -- rm –r /tmp/gld

## Make Sure Brokers Started Fine

b01

amql

tail –f amq.log

## Log into the NFS Server and Configure Load Run Parameters

f01

nfsl

# update ‘produce’ and ‘consumer’, configure the current test

Optionally, start an associated Excel spreadsheet.

## Remove All Previous Tests’ Artifacts

em run c01-c10 p001-p010 -- clean

## Start the Consumers

em run c01-c10 -- consume

Log into one of the consumers and tail the consume.csv to make sure the process started and it collects statistics.

## Start the Producers

em run p001-p010 -- produce

## Run the Test

Wait until the message batch was generated.

## Stop All gld Processes

em run c01-c10 p001-p010 -- stop --force

## Collect Statistics

cd /Users/ovidiu/c082-Fitbit/em-scripts

./collect-stats

## Stop All Instances Involved

em stop p001-p010 c01-c10 b01 b02 nfs01