Synthetic Load Generator Guide

Table of contents

1 Overview

The synthetic load generator (SLG) is a tool for testing NameNode behavior under different client loads. The user can generate different mixes of read, write, and list requests by specifying the probabilities of read and write. The user controls the intensity of the load by adjusting parameters for the number of worker threads and the delay between operations. While load generators are running, the user can profile and monitor the running of the NameNode. When a load generator exits, it prints some NameNode statistics like the average execution time of each kind of operation and the NameNode throughput.

2 Synopsis

The synopsis of the command is:

```
java LoadGenerator [options]
```

Options include:

- `-readProbability <read probability>`
  The probability of the read operation; default is 0.3333.
- `-writeProbability <write probability>`
  The probability of the write operations; default is 0.3333.
- `-root <test space root>`
  The root of the test space; default is /testLoadSpace.
- `-maxDelayBetweenOps <maxDelayBetweenOpsInMillis>`
  The maximum delay between two consecutive operations in a thread; default is 0 indicating no delay.
- `-numOfThreads <numOfThreads>`
  The number of threads to spawn; default is 200.
- `-elapsedTime <elapsedTimeInSecs>`
  The number of seconds that the program will run; A value of zero indicates that the program runs forever. The default value is 0.
- `-startTime <startTimeInMillis>`
  The time that all worker threads start to run. By default it is 10 seconds after the main program starts running.This creates a barrier if more than one load generator is running.
- `-seed <seed>`
  The random generator seed for repeating requests to NameNode when running with a single thread; default is the current time.

After command line argument parsing, the load generator traverses the test space and builds a table of all directories and another table of all files in the test space. It then waits until the start time to spawn the number of worker threads as specified by the user. Each thread sends a stream of requests to NameNode. At each iteration, it first decides if it is going to read a

file, create a file, or list a directory following the read and write probabilities specified by the user. The listing probability is equal to *1-read probability-write probability*. When reading, it randomly picks a file in the test space and reads the entire file. When writing, it randomly picks a directory in the test space and creates a file there.

To avoid two threads with the same load generator or from two different load generators creating the same file, the file name consists of the current machine's host name and the thread id. The length of the file follows Gaussian distribution with an average size of 2 blocks and the standard deviation of 1. The new file is filled with byte 'a'. To avoid the test space growing indefinitely, the file is deleted immediately after the file creation completes. While listing, it randomly picks a directory in the test space and lists its content.

After an operation completes, the thread pauses for a random amount of time in the range of [0, maxDelayBetweenOps] if the specified maximum delay is not zero. All threads are stopped when the specified elapsed time is passed. Before exiting, the program prints the average execution for each kind of NameNode operations, and the number of requests served by the NameNode per second.

3 Test Space Population

The user needs to populate a test space before running a load generator. The structure generator generates a random test space structure and the data generator creates the files and directories of the test space in Hadoop distributed file system.

3.1 Structure Generator

This tool generates a random namespace structure with the following constraints:

1. The number of subdirectories that a directory can have is a random number in [minWidth, maxWidth].
2. The maximum depth of each subdirectory is a random number [2*maxDepth/3, maxDepth].
3. Files are randomly placed in leaf directories. The size of each file follows Gaussian distribution with an average size of 1 block and a standard deviation of 1.

The generated namespace structure is described by two files in the output directory. Each line of the first file contains the full name of a leaf directory. Each line of the second file contains the full name of a file and its size, separated by a blank.

The synopsis of the command is:

```
java StructureGenerator [options]
```

Options include:

- `-maxDepth <maxDepth>`

Maximum depth of the directory tree; default is 5.

- `-minWidth <minWidth>`
  Minimum number of subdirectories per directories; default is 1.
- `-maxWidth <maxWidth>`
  Maximum number of subdirectories per directories; default is 5.
- `-numOfFiles <#OfFiles>`
  The total number of files in the test space; default is 10.
- `-avgFileSize <avgFileSizeInBlocks>`
  Average size of blocks; default is 1.
- `-outDir <outDir>`
  Output directory; default is the current directory.
- `-seed <seed>`
  Random number generator seed; default is the current time.

3.2 Data Generator

This tool reads the directory structure and file structure from the input directory and creates the namespace in Hadoop distributed file system. All files are filled with byte 'a'.

The synopsis of the command is:

```
java DataGenerator [options]
```

Options include:

- `-inDir <inDir>`
  Input directory name where directory/file structures are stored; default is the current directory.
- `-root <test space root>`
  The name of the root directory which the new namespace is going to be placed under; default is "/testLoadSpace".