Scalyr Coding Challenge: S3 file fetcher

Goal

For this challenge, you will implement a parallelized file-search utility, using a simulated client to pull files from “Amazon S3” and then search them with a grep-like function we provide. In other words, the “network fetch” and “search these bytes” parts are implemented already, and your job is to use them in a clever fashion to execute the search as quickly as possible.

We provide a naive implementation which downloads and searches each file serially. Your solution should focus on parallelizing these operations, with thought given to your threading model, supporting data structures, and potential bottlenecks. You are encouraged to use modern Java constructs like lambdas, etc.

Although we only benchmark with files totalling 50MB, your solution should support much larger data sets that do not fit in main memory nor on local disk. You can assume that individual files are small (<=1MB) as in our benchmark.

You will be implementing this project during a roughly two hour pair programming exercise with members from Scalyr’s engineering team. They will provide feedback on your design and implementation, while also providing additional structure to the exercise.

Pre-interview Preparation

While most of the work for this exercise will be done during your onsite interview, we ask that you do the following before you come in:

- Read over problem description
- Download the provided source to a laptop that you will bring onsite
- Set up the implementation in any development environment of your choosing.
- Verify that you can run the naive implementation in your development environment.

Additionally, please feel free to look over the naive implementation and begin thinking of ways you would improve it.

If you do not have a laptop that you can use for the interview, please contact your interview coordinator to make sure a desktop machine is available for your use.

Onsite Deliverables

During your onsite interview, you will be asked to provide a working solution for the problem. We ask that you treat this as you would any professional piece of programming. At the end of the exercise, you will be asked to package up your work (in a compilable+runnable form, including any 3rd party libraries you introduce) and given to your interviewer.

Your solution should:

- Allow the user to specify a search term as a commandline argument
- Perform the search as quickly as possible
- Print the total number of matches found and the elapsed search time
    - note the sample implementation doesn't quite meet this requirement

You are also encouraged to add any features or options that you feel are appropriate.

Milestones

Candidates should move through each of these stages, in order:

- Working parallel implementation
- Benchmarking your solution & exploring its parameter space
- "Pull request ready" code

Simulation Environment

See the *Setup, building, running* section below for the project URL and for an overview of the directory layout.

The classes

Your searcher will rely on two classes we provide:

```
com.scalyr.s3search....
  s3simulation.SimulatedS3Client  # Implements a simulated version of Amazon S3
  textsearch.TextSearcher         # Fast boyer-moore searcher for UTF-8 text
```

To get a sense for how they are used, see the sample implementation provided in `src/com/scalyr/s3search/Main.java`. Your implementation should use the same SimulatedS3Client constructor and, for testing, you can specify the same search term (`"pewter"`).

Both classes are threadsafe; a single instance of each should be shared amongst all your parallel work units.

Additional details relevant to your task

Exceptions

By default, 0.25% of calls to `SimulatedS3Client.readFileFromS3` will throw a `FlakyNetworkException`. This error is transient; you can (and should) retry in a smart way.

Bandwidth

The network simulation has a maximum bandwidth for the entire network connection, as well as a per-stream bandwidth limit. Note that request performance includes a random element, meant to represent factors such as network noise and disk queuing. If you request the same file twice, the resulting requests may complete at different speeds.

Underlying storage mechanism

The `SimulatedS3Client` reads from a local directory of 100 test files, using a `NetworkSimulator` which slows these reads as discussed above.

The files themselves are generated using a utility program, `FileCreator` (see below); you need run this only once ever.

Additional details not relevant to your task

`TextSearcher.java` actually searches for all strings that are close permutations of the search string (techically, all strings that are a single transposition or character replacement away).

Setup, building, running

You may download the source for this project at the following URL:

https://s3.amazonaws.com/scalyr-static/BackendMiniProject.tar.gz.

This package contains bare-bones bash scripts for building and running, but you should use whatever development environment/IDE you prefer.

Brief overview of the directory structure:

```
./src  - the "production" java source files
./test - junit test files for the above
./lib  - 3rd party jars (junit).  If you add 3rd party libs, please put them here
```

To build and run the source, there are two bash scripts (aka “poor man’s make”):

```
./build        - builds all classes in 'src' and 'test' to target/{,test-}classes
./run <CLASS>  - runs the given main class, defaulting to com.scalyr.s3search.Main
```

To setup the simulation environment, you’ll need to run (once only) the `FileCreator` class:

```
./build
./run com.scalyr.s3search.FileCreator
```

FileCreator creates a `./s3SimulationFiles` directory and writes 100 x 500KB files to it. The `SimulatedS3Client` then reads files from that same location.

Once `./s3SimulationFiles` is populated, you can run the sample (naive) implementation like so:

```
./run
./run com.scalyr.s3search.Main  # (same as above)
```