Processing large files efficiently in Java – part 2

Posted on March 4, 2015 by Arulkumaran Kumaraswamipillai

Processing large files efficiently in Java – part 1 covered different ways to read a large file. This post extends that to include some processing logic in addition to reading a file using **Java 8 stream**.

File details

4MB CSV file with **14,840** lines.

Machine Spec

4 Core CPU DELL Windows machine

Processing time simulated

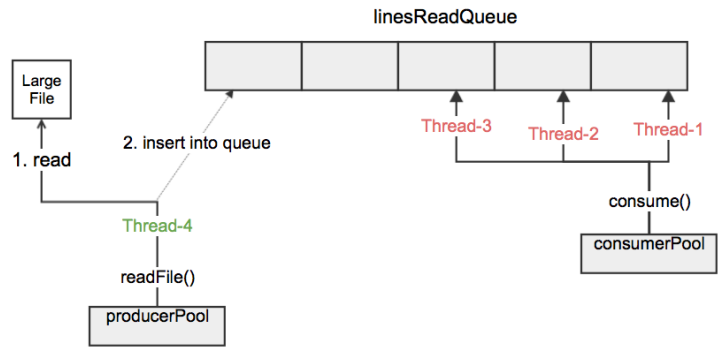
15ms with a dummy "*processCpuDummy()*" method.

So, the estimated processing time:

14,840 lines with each line taking **15ms** CPU processing time in the processCpuDummy() method, total estimated time = 222600 ms (or **222.6** seconds)

Code used

Producer/Consumer paradigm is used via a **BlockingQueue**. The main thread spawns 1 producer thread and x number of consumer threads in a fixed thread pool. The producer is responsible for reading a line at a time from the file, and insert in to a "BlockingQueue". The Consumers are responsible for removing the inserted line from the "BlockingQueue" and processing them. The nature of the blocking queue is such that the producing thread will be blocked if the queue capacity (e.g. 30) is reached, and the consumer thread will be blocked if the queue is empty.



Producer & Consumer with a blocking queue

```
1  package com.large.file;
2
3  import java.nio.charset.StandardCharsets;
4  import java.nio.file.Files;
5  import java.nio.file.Path;
6  import java.nio.file.Paths;
7  import java.util.concurrent.ArrayBlockingQueue;
8  import java.util.concurrent.BlockingQueue;
9  import java.util.concurrent.ExecutorService;
```


As a Java Architect

Java architecture & design concepts interview Q&As with diagrams | What should be a typical Java EE architecture?

```
10  import java.util.concurrent.Executors;
11  import java.util.concurrent.LinkedBlockingQueue
12  import java.util.concurrent.TimeUnit;
13  import java.util.stream.Stream;
14
15  public class Java8StreamRead implements Runnabl
16
17      private static final int CONSUMER_COUNT = 1
18      private final static BlockingQueue<String>
19
20      private boolean isConsumer = false;
21      private static boolean producerIsDone = fal
22
23      public Java8StreamRead(boolean consumer) {
24          this.isConsumer = consumer;
25      }
26
27      public static void main(String[] args) {
28
29          long startTime = System.nanoTime();
30
31          ExecutorService producerPool = Executor
32          producerPool.submit(new Java8StreamRead
33
34
35          // create a pool of consumer threads to
36          ExecutorService consumerPool = Executor
37          for (int i = 0; i < CONSUMER_COUNT; i++
38              consumerPool.submit(new Java8Stream
39
40          }
41
42          producerPool.shutdown();
43          consumerPool.shutdown();
44
45          while (!producerPool.isTerminated() &&
46          }
47
48          long endTime = System.nanoTime();
49          long elapsedTimeInMillis = TimeUnit.MIL
50          System.out.println("Total elapsed time:
51
52      }
53
54      private void readFile() {
55          Path file = Paths.get("c:/temp/my-large
56          try
57          {
58              //Java 8: Stream class
59              Stream<String> lines = Files.lines(
60
61              for( String line : (Iterable<String
62              {
63                  //System.out.println("read=" +
64                  linesReadQueue.put(line); //blo
65                  //System.out.println(Thread.cur
66              }
67
68          } catch (Exception e){
69              e.printStackTrace();
70          }
71
72          producerIsDone = true; // signal consum
73          System.out.println(Thread.currentThread
74      }
75
```

```
76      @Override
77      public void run() {
78          if (isConsumer) {
79              consume();
80          } else {
81              readFile(); //produce data by readi
82          }
83      }
84
85      private void consume() {
86          try {
87              while (!producerIsDone || (producer
88                  String lineToProcess = linesRea
89                  processCpuDummy(); // some CPU
90                  //System.out.println("procesed:
91                  //System.out.println(Thread.cur
92              }
93          } catch (Exception e) {
94              e.printStackTrace();
95          }
96
97          System.out.println(Thread.currentThread
98      }
99
100     public void processCpuDummy() {
101         //takes ~ 15 ms of CPU time
102         //did not use Thread.sleep() as it does
103         for (long i = 0; i < 1000000001; i++) {
104             i = i+1;
105         }
106     }
107
108 }
109
```

Run 1 result: single producer with a single consumer

Output

```
1 pool-1-thread-1 producer is done
2 Total elapsed time: 281154 ms
3 pool-2-thread-1 consumer is done
```

observations

CPU utilization = **50%**

```
1 private static final int CONSUMER_COUNT = 1;
```

Run 2 result: single producer with 4 consumers

Output

```
1 pool-1-thread-1 producer is done
2 Total elapsed time: <strong>138441 ms</strong>
3 pool-2-thread-3 consumer is done
4 pool-2-thread-4 consumer is done
5 pool-2-thread-2 consumer is done
6 pool-2-thread-1 consumer is done
```

observations

CPU utilization = **100%**

```
1 private static final int CONSUMER_COUNT = 4;
```

Run 3 result: single producer with 10 consumers

Output

```
1  pool-1-thread-1 producer is done
2  Total elapsed time: 124907 ms
3  pool-2-thread-2 consumer is done
4  pool-2-thread-8 consumer is done
5  pool-2-thread-4 consumer is done
6  pool-2-thread-3 consumer is done
7  pool-2-thread-1 consumer is done
8  pool-2-thread-10 consumer is done
9  pool-2-thread-5 consumer is done
10 pool-2-thread-9 consumer is done
11 pool-2-thread-7 consumer is done
12 pool-2-thread-6 consumer is done
13
```

observations

CPU utilization = **100%**

```
1 private static final int CONSUMER_COUNT = 10;
```

Now, experiment this with the other file reading methods covered in part-1 like the scanner class. Also, experiment with changing linesReadQueue.**put**(line); to

linesReadQueue.**offer**(line);. You can't compare performance between put & offer as they are used in different scenarios.

offer – offer method is to just offer to the queue and it does not wait for a space to become available or for a specified time. Use this only if you can afford to loose items.

put – waits infinitely for a space to become available. That is what we needed.

Experiment with multi-threaded producers reading every 2nd, 3rd, etc lines.

Popular Posts

♦ 11 Spring boot interview questions & answers

891 views

♦ Q11-Q23: Top 50+ Core on Java OOP Interview Questions & Answers

851 views

18 Java scenarios based interview Questions and Answers

456 views

001A: ♦ 7+ Java integration styles & patterns interview questions & answers

411 views

♦ 7 Java debugging interview questions & answers

315 views

♦ 10 ERD (Entity-Relationship Diagrams) Interview Questions and Answers

313 views

01b: ♦ 13 Spring basics Q8 – Q13 interview questions & answers

305 views

01: ♦ 15 Ice breaker questions asked 90% of the time in Java job interviews with hints

288 views

♦ Q24-Q36: Top 50+ Core on Java classes, interfaces and generics interview questions & answers

267 views

8 Git Source control system interview questions & answers

216 views

Bio | Latest Posts

Arulkumaran Kumaraswamipillai

Mechanical Eng to freelance Java developer in 3 yrs. Contracting since 2003, and attended 150+ Java job interviews, and often got 4 - 7 job offers to choose from. It pays to prepare. So, published Java interview Q&A books via Amazon.com in 2005, and sold 35,000+ copies. Books are outdated and replaced with this subscription based site.

About Arulkumaran Kumaraswamipillai

Mechanical Eng to freelance Java developer in 3 yrs. Contracting since 2003, and attended 150+ Java job interviews, and often got 4 - 7 job offers to choose from. It pays to prepare. So, published Java interview Q&A books via Amazon.com in 2005, and sold 35,000+ copies. Books are outdated and replaced with this subscription based site.

‹ Processing large files efficiently in Java – part 1

5 Swing & AWT interview questions and answers ›

Posted in IO, Performance

Empowers you to open more doors, and fast-track

Technical Know Hows

☀ Java generics in no time ☀ Top 6 tips to transforming your thinking from OOP to FP ☀ How does a HashMap internally work? What is a hashing function?
☀ 10+ Java String class interview Q&As ☀ Java auto un/boxing benefits & caveats ☀ Top 11 slacknesses that can come back and bite you as an experienced Java developer or architect

Non-Technical Know Hows

☀ 6 Aspects that can motivate you to fast-track your career & go places ☀ Are you reinventing yourself as a Java developer? ☀ 8 tips to safeguard your Java career against offshoring ☀ My top 5 career mistakes

Prepare to succeed

☀ Turn readers of your Java CV go from “Blah blah” to “Wow”? ☀ How to prepare for Java job interviews? ☀ 16 Technical Key Areas ☀ How to choose from multiple Java job offers?

Select Category

© Disclaimer