Laboratory Assignment 8 Write Up
Computer Science 441

Andreas Bach Landgrebe

March 30, 2016

Date Submitted: March 30, 2016
Partners: Andreas Bach Landgrebe
Instructor: Dr. Gregory M. Kapfhammer

1 The well-commented source code and output for your own file system benchmark.

readData.java

```
import java.io.*;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.text.DecimalFormat;

public class readData {

static int bufferSize = 1024; //buffered size
static String fileName = "/tmp/benchmark.dat"; //file name and path
static String logFileName = "logfile.txt"; //txt file that is being
    written to
static int noOfRepetitions = 10; //number of times the experiements are
    being repeated
static byte[] emptyBuffer; //btye array
static long fileSize = 100*1000*bufferSize;; //file size
static PrintWriter logFile; //printer writer being used to print out the
    output

public static void main(String args[]) {

// program call: readData buffersize filesize filename
// buffersize: size of buffer in bytes (integer), default:
// filesize: size of file in bytes (long)
// filename: name of temporary file including path
```

```
long timeSpend;

try { //attetmpt the benchmarks
logFile = new PrintWriter(logFileName);
} catch (IOException e) {
System.err.println(e);
}


if (args.length>=1) { //if one set one argument, then this is going to
    be an integer or a buffered size
try {
bufferSize = Integer.parseInt(args[0]);
} catch (NumberFormatException e) {
System.err.println("Argument BufferSize" + args[0] + " must be an
    integer.");
System.exit(1);
}
}

if (args.length>=2) {
//if two arguments are set, then the first will be the buffered size and
    the second will be the size of the file in bytes
try {
fileSize = Long.parseLong(args[1]);
} catch (NumberFormatException e) {
System.err.println("Argument fileSize" + args[1] + " must be a long.");
System.exit(1);
}
}

if (args.length>=3) fileName = args[2];
//if three arguments are set,
//the first one will be the buffered size
//the second one will be the size of the file in bytes
//the three one will be the file path and name

emptyBuffer = new byte[bufferSize];

createFile(fileName, fileSize);


//for loop to do a certain number of repetitions to the experiment
for (int i =0 ; i<noOfRepetitions; i++){
timeSpend = readFileBuffered(fileName);
logFile.println(csvLine(fileName,"read",timeSpend,((fileSize/timeSpend)
    * 1000),fileSize,bufferSize));
}
//for loop to do a certain number of reptitions to the experiment
for (int i =0 ; i<noOfRepetitions; i++){
```

```
timeSpend = writeFileBuffered(fileName, fileSize);
logFile.println(csvLine(fileName,"write",timeSpend,((fileSize/timeSpend)
    * 1000),fileSize,bufferSize));
}

deleteFile(fileName);

logFile.close();
}
//custom method to be formatted into a csv file
public static String csvLine(String fileName, String operation, long
    timespend, long bytesPersecond, long fileSize, int bufferSize) {
return String.format("\"%s\",\"%s\",%d,%d,%d,%d",
    fileName,operation,timespend,bytesPersecond,fileSize,bufferSize);
}

public static void createFile(String fileName, long fileSize) {
//this method creates the file
{
try {
RandomAccessFile file = new RandomAccessFile(fileName, "rw");

for (long i = 0; i < fileSize; i += bufferSize)
{
file.write(emptyBuffer, 0, bufferSize);
}
file.close();
} catch (IOException e) {
System.err.println(e);
}

}

}

public static void deleteFile(String fileName) {
//this method delete the file once it has been completed
try {
Files.deleteIfExists(Paths.get(fileName));
} catch (IOException e) {
// File permission problems are caught here.
System.err.println(e);
}
}

public static long readFileBuffered(String fileName) {
//this method is being used to read the file in

long startTime = System.currentTimeMillis();
try {
```

```java
FileInputStream fileInputStream = new FileInputStream(fileName);
BufferedInputStream bufferedInputStream = new
    BufferedInputStream(fileInputStream,bufferSize);
int count = 0;
int x;
while ((x = bufferedInputStream.read()) != -1) {
count++;
}
bufferedInputStream.close();


} catch (IOException e) {
System.err.println(e);
}
long endTime = System.currentTimeMillis();
long elapsedTime = endTime - startTime;

return elapsedTime;
}

public static long writeFileBuffered(String fileName,long fileSize) {
//this method is being used to write to the file
long startTime = System.currentTimeMillis();
try {
FileOutputStream fileOutputStream = new FileOutputStream(fileName);

for (long i = 0; i < fileSize; i += bufferSize) {
fileOutputStream.write(emptyBuffer);

}
fileOutputStream.flush();
fileOutputStream.close();

} catch (IOException e) {
System.err.println(e);
}

long endTime = System.currentTimeMillis();
long elapsedTime = endTime - startTime;


return elapsedTime;
}

}
```

Output
logfile.txt

SSD 100MB

```
"C:/tmp/benchmark.dat","read",2153,46446000,100000000,1024
"C:/tmp/benchmark.dat","read",2121,47147000,100000000,1024
"C:/tmp/benchmark.dat","read",326,306748000,100000000,1024
"C:/tmp/benchmark.dat","read",323,309597000,100000000,1024
"C:/tmp/benchmark.dat","read",325,307692000,100000000,1024
"C:/tmp/benchmark.dat","read",324,308641000,100000000,1024
"C:/tmp/benchmark.dat","read",323,309597000,100000000,1024
"C:/tmp/benchmark.dat","read",328,304878000,100000000,1024
"C:/tmp/benchmark.dat","read",323,309597000,100000000,1024
"C:/tmp/benchmark.dat","read",320,312500000,100000000,1024
"C:/tmp/benchmark.dat","write",316,316455000,100000000,1024
"C:/tmp/benchmark.dat","write",303,330033000,100000000,1024
"C:/tmp/benchmark.dat","write",305,327868000,100000000,1024
"C:/tmp/benchmark.dat","write",304,328947000,100000000,1024
"C:/tmp/benchmark.dat","write",306,326797000,100000000,1024
"C:/tmp/benchmark.dat","write",300,333333000,100000000,1024
"C:/tmp/benchmark.dat","write",305,327868000,100000000,1024
"C:/tmp/benchmark.dat","write",300,333333000,100000000,1024
"C:/tmp/benchmark.dat","write",304,328947000,100000000,1024
"C:/tmp/benchmark.dat","write",303,330033000,100000000,1024
```

SSD 2MB

```
"C:/tmp/benchmark.dat","read",47,42553000,2000000,1024
"C:/tmp/benchmark.dat","read",44,45454000,2000000,1024
"C:/tmp/benchmark.dat","read",44,45454000,2000000,1024
"C:/tmp/benchmark.dat","read",45,44444000,2000000,1024
"C:/tmp/benchmark.dat","read",42,47619000,2000000,1024
"C:/tmp/benchmark.dat","read",43,46511000,2000000,1024
"C:/tmp/benchmark.dat","read",44,45454000,2000000,1024
"C:/tmp/benchmark.dat","read",43,46511000,2000000,1024
"C:/tmp/benchmark.dat","read",44,45454000,2000000,1024
"C:/tmp/benchmark.dat","read",43,46511000,2000000,1024
"C:/tmp/benchmark.dat","write",9,222222000,2000000,1024
"C:/tmp/benchmark.dat","write",9,222222000,2000000,1024
"C:/tmp/benchmark.dat","write",8,250000000,2000000,1024
"C:/tmp/benchmark.dat","write",8,250000000,2000000,1024
"C:/tmp/benchmark.dat","write",7,285714000,2000000,1024
"C:/tmp/benchmark.dat","write",8,250000000,2000000,1024
"C:/tmp/benchmark.dat","write",10,200000000,2000000,1024
"C:/tmp/benchmark.dat","write",7,285714000,2000000,1024
"C:/tmp/benchmark.dat","write",8,250000000,2000000,1024
"C:/tmp/benchmark.dat","write",10,200000000,2000000,1024
```

HDD 2MB

```
"E:/tmp/benchmark.dat","read",47,42553000,2000000,1024
```

```
"E:/tmp/benchmark.dat","read",54,37037000,2000000,1024
"E:/tmp/benchmark.dat","read",45,44444000,2000000,1024
"E:/tmp/benchmark.dat","read",45,44444000,2000000,1024
"E:/tmp/benchmark.dat","read",43,46511000,2000000,1024
"E:/tmp/benchmark.dat","read",42,47619000,2000000,1024
"E:/tmp/benchmark.dat","read",43,46511000,2000000,1024
"E:/tmp/benchmark.dat","read",43,46511000,2000000,1024
"E:/tmp/benchmark.dat","read",42,47619000,2000000,1024
"E:/tmp/benchmark.dat","read",46,43478000,2000000,1024
"E:/tmp/benchmark.dat","write",13,153846000,2000000,1024
"E:/tmp/benchmark.dat","write",8,250000000,2000000,1024
"E:/tmp/benchmark.dat","write",6,333333000,2000000,1024
"E:/tmp/benchmark.dat","write",7,285714000,2000000,1024
"E:/tmp/benchmark.dat","write",7,285714000,2000000,1024
"E:/tmp/benchmark.dat","write",7,285714000,2000000,1024
"E:/tmp/benchmark.dat","write",7,285714000,2000000,1024
"E:/tmp/benchmark.dat","write",8,250000000,2000000,1024
"E:/tmp/benchmark.dat","write",7,285714000,2000000,1024
"E:/tmp/benchmark.dat","write",7,285714000,2000000,1024

HDD 100MB
"E:/tmp/benchmark.dat","read",2247,44503000,100000000,1024
"E:/tmp/benchmark.dat","read",2109,47415000,100000000,1024
"E:/tmp/benchmark.dat","read",317,315457000,100000000,1024
"E:/tmp/benchmark.dat","read",316,316455000,100000000,1024
"E:/tmp/benchmark.dat","read",317,315457000,100000000,1024
"E:/tmp/benchmark.dat","read",314,318471000,100000000,1024
"E:/tmp/benchmark.dat","read",315,317460000,100000000,1024
"E:/tmp/benchmark.dat","read",317,315457000,100000000,1024
"E:/tmp/benchmark.dat","read",315,317460000,100000000,1024
"E:/tmp/benchmark.dat","read",314,318471000,100000000,1024
"E:/tmp/benchmark.dat","write",309,323624000,100000000,1024
"E:/tmp/benchmark.dat","write",294,340136000,100000000,1024
"E:/tmp/benchmark.dat","write",297,336700000,100000000,1024
"E:/tmp/benchmark.dat","write",296,337837000,100000000,1024
"E:/tmp/benchmark.dat","write",294,340136000,100000000,1024
"E:/tmp/benchmark.dat","write",294,340136000,100000000,1024
"E:/tmp/benchmark.dat","write",376,265957000,100000000,1024
"E:/tmp/benchmark.dat","write",310,322580000,100000000,1024
"E:/tmp/benchmark.dat","write",290,344827000,100000000,1024
"E:/tmp/benchmark.dat","write",292,342465000,100000000,1024
```

2 Using both text and diagrams, a description of all benchmarks used in your experiments.

The benchmarks used in my experiments include testing the solid state drive (SSD) and Hard Drive (HDD) on the computer that I own and testing the speed of these specific kinds of drives. I decided to test the performance of my Solid State Drive and Hard Drive in regards to being able to do read and write permissions. In the above section, I displayed the output being generated each time between the SSD and the HDD. The first part is the path being used. The second part is declaring whether it is reading or writing to the drive. The next number illustrates the amount of time it took in milliseconds. The next number is the amount of bytes being processed per second. The next number is the size of the file. The last number is the buffered size which stayed the same throughout the experiments for this laboratory assignment.

3 A document that summarizes the published paper that explains store system benchmarking.

After reading the published paper "A Nine Year Study of File System and Storage Benchmarking", it is important to be able to understand store system benchmarking. In the paper, it addressed how benchmarking is crucial when looking into the performance for file and storage systems. This paper surveys 415 file system and storage benchmarks from 106 recent papers [3]. This paper categories three different types of benchmarks. These categories are Macrobenchmarks, Trace Replays, and Microbenchmarks. Macrobenchmarks is the performance that is tested against a particular workload. The Trace Replays is a program that replays operations which were recorded in a real scenario. Microbenchmarks are a few operations that are tested to isolate their specific overheads within the system. This article also addressing two underlying themes. These themes are explains what was done in as much detail as possible and in addition to saying what was done, say why it was done that way. In order to understand the results of my experiments, it is crucial to be able to share my hard specification of my computer. This way, it is easier to understand my results and why they came out to be this particular way. In addition to say what was done, it is also important to explain why it was done a particular way. This is due to the fact that it is important for readers to best understand the results and why certain decisions were made to best furnish these benchmarking experiments.
To conclude, this paper has examined a wide range of file system and storage benchmarks and described their positive and negative qualities [3]. Possible future storage research can also to help answer the following questions. The first question is how can we accurately portray various real-world workloads. The next question is how can we accurately compare results from benchmarks that were run on different machines and systems and at different times. To better help answer the first question, a method needs to be implemented to determine how close two workloads are to each other. To answer the second question will be a challenge [3]. It is difficult to accurately compare results from benchmarks. This is due to the fact that several operations are being performed constantly

throughout an operation system. It is also due to the fact that different machines have various hardware specifications and with these hardware specs, it is difficult to compare and contrast different benchmark results.

4 A summary paper that explains the key features and parameters of the bonnie++ tool

Bonnie++ is a program to test hard drive and file systems for performance. This bonnie++ tool can take on many parameters or arguments [1]. For example if one specifies -d, then the next thing that comes is the destination to the file being written to. If one specifies -s, then the next piece of information that is specified is the size of the test file. If one specifies -m, then one would be specifying a label which will be written out with the results [1].

5 A detailed report presenting your experimentation framework and analyzing your results.

In order to run my benchmarks successfully, I decided to use the computer that I possess. The specification of this computer is as follows:
Windows 10 Pro
Processor: Intel(R)Core(TM) i7-3770K CPU @3.50GHz 3.90GHz
Installed Memory (RAM): 16.0 GB
System type: 64-bit Operating System,x64-based processor
SSD: Samsung SSD 850 PRO 512GB
HDD: TOSHIBA DT01ACA200
As I thinking about the benchmarks that I would like to run, I decided to run the benchmarks on my solid state drive and my hard drive. I decided to test it on 2MB and 100MB of data. I decided on a very small amount of data to see how it would affect the cache. The following details were furnished after running the experiments 10 times for both read and write. These details furnished produced the Mean(Average) and Standard Deviation.
SSD 100MB Read:
1: 2153 Milliseconds
2: 2121 Milliseconds
3: 326 Milliseconds
4: 323 Milliseconds
5: 325 Milliseconds
6: 324 Milliseconds
7: 323 Milliseconds
8: 328 Milliseconds
9: 323 Milliseconds
10: 320 Milliseconds
Mean(Average): 686.6 Milliseconds

Standard Deviation: 764.46804

SSD 100MB Write:
1: 316 Milliseconds
2: 303 Milliseconds
3: 305 Milliseconds
4: 304 Milliseconds
5: 306 Milliseconds
6: 300 Milliseconds
7: 305 Milliseconds
8: 300 Milliseconds
9: 304 Milliseconds
10: 303 Milliseconds
Mean(Average): 304.6 Milliseconds
Standard Deviation: 4.4771

SSD 2MB Read:
1: 47 Milliseconds
2: 44 Milliseconds
3: 44 Milliseconds
4: 45 Milliseconds
5: 42 Milliseconds
6: 43 Milliseconds
7: 44 Milliseconds
8: 43 Milliseconds
9: 44 Milliseconds
10: 43 Milliseconds
Mean(Average): 43.9 Milliseconds
Standard Deviation: 1.37032

SSD 2MB Write:
1: 9 Milliseconds
2: 9 Milliseconds
3: 8 Milliseconds
4: 8 Milliseconds
5: 7 Milliseconds
6: 8 Milliseconds
7: 10 Milliseconds
8: 7 Milliseconds
9: 8 Milliseconds
10: 10 Milliseconds
Mean(Average): 8.4 Milliseconds
Standard Deviation: 1.07497

HDD 100MB Read:
1: 2247 Milliseconds

2: 2109 Milliseconds
3: 317 Milliseconds
4: 316 Milliseconds
5: 317 Milliseconds
6: 314 Milliseconds
7: 315 Milliseconds
8: 317 Milliseconds
9: 315 Milliseconds
10: 314 Milliseconds
Mean(Average): 688.1 Milliseconds
Standard Deviation: 785.92047

HDD 100MB Write:
1: 309 Milliseconds
2: 294 Milliseconds
3: 297 Milliseconds
4: 296 Milliseconds
5: 294 Milliseconds
6: 294 Milliseconds
7: 376 Milliseconds
8: 310 Milliseconds
9: 290 Milliseconds
10: 292 Milliseconds
Mean(Average): 305.2 Milliseconds
Standard Deviation: 25.78458

HDD 2MB Read:
1: 47 Milliseconds
2: 54 Milliseconds
3: 45 Milliseconds
4: 45 Milliseconds
5: 43 Milliseconds
6: 42 Milliseconds
7: 43 Milliseconds
8: 43 Milliseconds
9: 42 Milliseconds
10: 46 Milliseconds
Mean(Average): 45 Milliseconds
Standard Deviation: 3.59011

HDD 2MB Write:
1: 13 Milliseconds
2: 8 Milliseconds
3: 6 Milliseconds
4: 7 Milliseconds

5: 7 Milliseconds
6: 7 Milliseconds
7: 7 Milliseconds
8: 8 Milliseconds
9: 7 Milliseconds
10: 7 Milliseconds
Mean(Average): 7.7 Milliseconds
Standard Deviation: 1.94651

As a summary, here is a simplified results write up without each instance being shown:
SSD 100 MB Read
Mean(Average): 686.6 Milliseconds
Standard Deviation: 764.46804

HDD 100MB Read:
Mean(Average): 688.1 Milliseconds
Standard Deviation: 785.92047

SSD 100MB Write:
Mean(Average): 304.6 Milliseconds
Standard Deviation: 4.4771

HDD 100MB Write:
Mean(Average): 305.2 Milliseconds
Standard Deviation: 25.78458

SSD 2MB Read:
Mean(Average): 43.9 Milliseconds
Standard Deviation: 1.37032

HDD 2MB Read:
Mean(Average): 45 Milliseconds
Standard Deviation: 3.59011

SSD 2MB Write:
Mean(Average): 8.4 Milliseconds
Standard Deviation: 1.07497

HDD 2MB Write:
Mean(Average): 7.7 Milliseconds
Standard Deviation: 1.94651

From the results, the averages being generated, whether it is read or write to the drives, the solid state drives is running quicker in almost every instance. However, despite the SSD performs better as an average the difference between the hard drive and the solid state drive is not as different as I had thought.

6 A reflection on the challenges that you encountered when completing this assignment.

There was one particular challenge that I had encountered when completing this assignment. This included being able to complete this laboratory assignment on my MacBook Pro. In the beginning, I was starting to encounter several issues being able to run bonnie++. In the end, in order to run bonnie++ properly, I need to run the following command:
export PATH=/usr/local/bin:/usr/local/sbin:/usr/bin:/bin:/usr/sbin:/sin
After I run this command in my terminal, I was able to run bonnie++ successfully.

References

[1] James Coyle. Simple bonnie example — jamescoyle.net, Dec 2013. `http://www.jamescoyle.net/how-to/913-simple-bonnie-example`. Accessed 30 March 2016.

[2] Andrew S. Tanenbaum and Maarten van. Steen. *Distributed Systems: Principles and Paradigms*. Pearson Prentice Hall, 2007.

[3] Avishay Traeger, Erez Zadok, Nikolai Joukov, and Charles P. Wright. A nine year study of file system and storage benchmarking. *Trans. Storage*, 4(2):5:1–5:56, May 2008.