CSC540- Programming Project 2:

Systems Project- File and Memory Management

Team 7

Akond Rahman (aarahman)

Saurav Shekhar(sshekha3)

Mithilesh Jaywant Gawande (mgawand)

Anuja Chandrashekhar Yawalkar(ayawalk)

Table of Contents

Java classes that are modified- 3

1. Changes in the BasicBufferMgr.java class 3
2. Changes made in the Page.java class 5

Graph and Analysis- 9

- We Used BasicBufferMgr with linear search strategy first 9
- We then ran 1000 random select statements from the client. The time Taken is noted down in the following table. 9
- The above step is repeated three times and the average values are noted down. 9
- We implemented the BasicBufferMgr with Hashed buffer strategy. 9
- We then took readings for the time averages in the hashed buffer scenario and tabulated the results as follows. 9

Graph plotted- 10

Analysis of graph- 11

Populate db tables with random values to perform the analysis- 11

Testing task 2- 12

Conclusion- 15

Table of Figures

Figure 1: Comparison of Sequential Access vs Mapping 10

Table of Tables

Table 1: Sequential Scan Results 9

Table 2: HashMap Results 9

Table 3: Average Time for both the methods 10

Java classes that are modified-

1. Changes in the BasicBufferMgr.java class

- Hash Map was created using the following syntax:-

```
Map<Block, Buffer> bufferPoolMap = new HashMap<Block, Buffer>() ;
```

The name of the Hash Map used is - **bufferPoolMap**

- Changes made in the Buffer pin function

```
synchronized Buffer pin(Block blk)

{

Buffer buff = findExistingBuffer(blk);

if (buff == null)

{  buff = chooseUnpinnedBuffer();

if (buff == null)

{ return null; }

buff.assignToBlock(blk);

/ * creating the mapping here */

assignBuffToblock(blk, buff); } else

{ / * Just keeping track of when the buffer is obtained from the mapping via console message */  }

if (!buff.isPinned())
        { numAvailable--; }
```

- Changes made to buffer pin New function

```
synchronized Buffer pinNew(String filename, PageFormatter fmtr) {

Buffer buff = chooseUnpinnedBuffer();

if (buff == null)

return null;
```

```
buff.assignToNew(filename, fmtr);
/* assigning the appropriate  block to a buffer */
assignBuffToblock(buff.block(), buff);
numAvailable--;
buff.pin();
return buff;
```

- Changes made to the unpin function

```
synchronized void unpin(Buffer buff)
{ Block blockObj =  buff.block();
buff.unpin();
if (!buff.isPinned())
{  /* First checking if there is an exiting mapping */
if(containsMapping(blockObj))
{  /* if there is a mapping then un-allocate the buffer*/
unassignBlockToBuff( buff);  }
numAvailable++;  }
```

- Changes made in the find existing buffer function:-

```
private Buffer findExistingBuffer(Block blk)
{  /*The variable to return the buffer */
Buffer buffToret = null ;
/ * Remove the Sequential accessing  because it is not used anymore */
/*Looking up the map the find an existing buffer*/
if(containsMapping(blk))
{ buffToret =  getMapping(blk);}
return buffToret ; }
```

- Created the method assignBuffToblock

/* **We created this method to assign a buffer to a block from the map created** */

```
private void assignBuffToblock( Block blockParam, Buffer bufferParam)

{bufferPoolMap.put( blockParam, bufferParam) ;  }
```

- Created this method to unassignBlockToBuff

/* **We created this method to un-map a block for a buffer .If the object is a buffer , then we get the corresponding block, and then remove the block.** */

```
private void unassignBlockToBuff(Buffer bufferParam )

{bufferPoolMap.remove(bufferParam.block()); }
```

2. Changes made in the Page.java class

- The following String was added to take the date input in string format and then store it in the form of Date Object.

```
String theFormat="yyyyMMdd";

SimpleDateFormat dateFormatObj = new SimpleDateFormat(theFormat);
```

- The following functions were added to set Date and get Date .

```
public synchronized void setDate(int offset, Date val)
{
contents.position(offset);
String dateStr = convertDateToStr(val) ;
byte[] byteval = dateStr.getBytes();
contents.putInt(byteval.length);
contents.put(byteval);
}
public synchronized Date getDate(int offset)
{
contents.position(offset);
int len = contents.getInt();
byte[] byteval = new byte[len];
```

```
contents.get(byteval);
String dateStringToSend = new String(byteval);
Date returnDateParam = convertStringToDate(dateStringToSend);
return returnDateParam ;
}
```

- The following functions were added to set Byte and get byte:-

```
{
contents.position(offset);
byte[] byteval = convertShrotValToByteArray(shortParam);
contents.putInt(byteval.length);
contents.put(byteval);
}
public synchronized short getShort(int offset)
{
Short shortValToret  ;
contents.position(offset);
int len = contents.getInt();
byte[] byteval = new byte[len];
contents.get(byteval);
shortValToret = convertByteArrayToShort(byteval);
return shortValToret ;
}
```

- The following classes were added to handle Boolean Data Types:-

```
public synchronized void setBoolean(int offset, boolean boolParam)
{
String strToStore="";
contents.position(offset);
if(boolParam)
{
strToStore = "true" ;
}
else
{
strToStore = "false" ;
}
byte[] byteval = strToStore.getBytes();
contents.putInt(byteval.length);
contents.put(byteval);
}
```

```
public synchronized boolean getBoolean(int offset)
{
boolean boolToret;
contents.position(offset);
int len = contents.getInt();
byte[] byteval = new byte[len];
contents.get(byteval);
String strToUse= new String(byteval);
if(strToUse.equals("true"))
{
boolToret = true ;
}
else
{
boolToret = false;
}
return boolToret ;
}
```

- The following methods were added to handle Short Data Type:-

```
public synchronized void setShort(int offset, short shortParam)
{
contents.position(offset);
byte[] byteval = convertShrotValToByteArray(shortParam);
contents.putInt(byteval.length);
contents.put(byteval);
}
public synchronized short getShort(int offset)
{
Short shortValToret  ;
contents.position(offset);
int len = contents.getInt();
byte[] byteval = new byte[len];
contents.get(byteval);
shortValToret = convertByteArrayToShort(byteval);
return shortValToret ;
}
```

- The following methods were added to convert the Short Data Type to Byte Data Type:-

```
private byte[] convertShrotValToByteArray(short value) {
int byteLen = 2 ;
byte[] bytesToret = new byte[byteLen];
ByteBuffer bufferInUse = ByteBuffer.allocate(bytesToret.length);
```

```
bufferInUse.putShort(value);
return bufferInUse.array();
}
```

- The following methods were added to convert the Byte array to the Short Array.

```
private  short convertByteArrayToShort(byte[] array)
{
ByteBuffer bufferToRet = ByteBuffer.wrap(array);
return bufferToRet.getShort();
}
```

- The following methods were added to convert the Date to string data type:-

```
private String convertDateToStr(Date dateParam)
{
String strToret ="" ;
strToret = dateFormatObj.format(dateParam);
return strToret ;
}
```

- The following methods were added to convert the Date to string data type:-

```
private Date convertStringToDate(String strParam)
{
Date dateObjToRet = new Date();
try
{
dateObjToRet = dateFormatObj.parse(strParam);
}
catch (ParseException e)
{
System.err.println("Error log " + e.getMessage());
e.printStackTrace();
}
return dateObjToRet ;
}
}
```

Graph and Analysis-

- The following measurements are taken.

- **We Used BasicBufferMgr with linear search strategy first**
- **We then ran 1000 random select statements from the client. The time Taken is noted down in the following table.**
- **The above step is repeated three times and the average values are noted down.**
- **We implemented the BasicBufferMgr with Hashed buffer strategy.**
- **We then took readings for the time averages in the hashed buffer scenario and tabulated the results as follows.**

The Buffer Sizes are in **bytes** and the time is in **seconds**.

Sequential Scan Results

Buffer Sizes	Sequential Time 1	Sequential Time 2	Sequential Time 3
8	1.721	1.716	1.716
16	1.715	1.704	1.723
50	1.716	1.762	1.724
500	1.888	1.767	1.801
10000	2.656	2.596	2.674
25000	3.962	3.994	4.08
50000	8.078	8.161	8.211
100000	16.932	17.66	16.578

Table 1: Sequential Scan Results

HashMap Results

Buffer Sizes	HashMap Time 1	HashMap Time 2	HashMap Time 3
8	1.721	1.716	1.716
16	1.715	1.704	1.723
50	1.716	1.762	1.724
500	1.888	1.767	1.801
10000	2.656	2.596	2.674
25000	3.962	3.994	4.08
50000	8.078	8.161	8.211
100000	16.932	17.66	16.578

Table 2: HashMap Results

The tables above show the experimental results obtained for the change in the time measurements for the given buffer size in case of sequential scan and hash map respectively.

The average results from the above tables can be seen as follows-

Buffer Sizes	Average Sequential Time	Average HashMap Time
8	1.717	1.698
16	1.714	1.712
50	1.734	1.732
500	1.818	1.751
10000	2.642	1.732
25000	4.012	1.827
50000	8.15	2.113
100000	17.116	1.805

Table 3: Average Time for both the methods

Graph plotted-

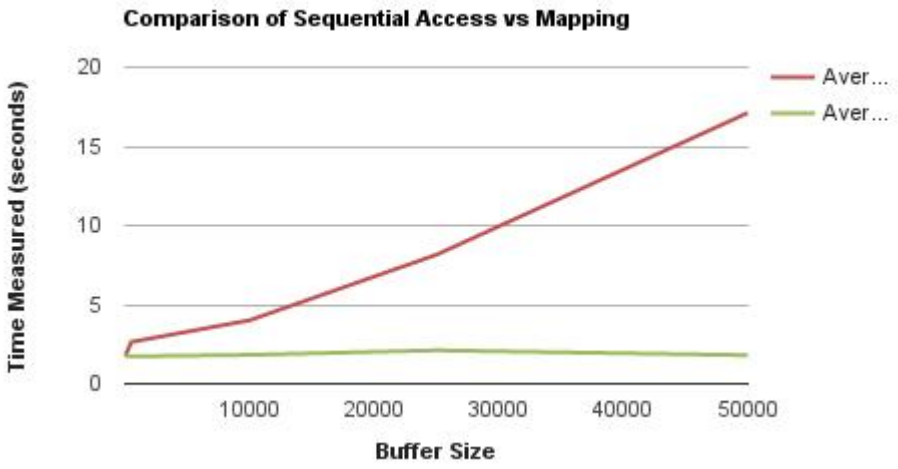

Figure 1: Comparison of Sequential Access vs Mapping

Analysis of graph-

The graph above shows the plotting of buffer size on the X axis and the Time Measurement on Y axis.

As shown in the figure the Green Line and the Pink Line indicate the increases in the size of the buffer as the time increases.

Case 1: When the sequential scan is used, the time goes on increasing as the buffer size increases indicated by the pink line in the figure.

Case 2: When the Hashmap is used, the time remains constant as the buffer size goes on increasing. This is because of the usage of hashmap to map a particular value from the given database.

Thus we see that the HashMap approach gives us a better running time as compared to the traditional approach. During small buffer sizes the changes are not that evident but that is due to lesser address space. The true behavior can be found when the buffer size increases.

There is also one observation we made that the timings are not consistent throughout. According to us it may arise due to the system configurations and the various processes running on the system at that particular time. Also the select statements which we have used will play a major part in time. However the trend is that in sequential access the time increases as the buffer size increases and in HashMap technique it remains more or less around 1.8-2.1 seconds.

Populate db tables with random values to perform the analysis-

DB Tables are populated using--> CreateStudentDB.java file in client. In FindMajors.java file we used for loop and run it 1000 times for select statements. In the code we have included comments which will relate to the same.

The following changes have been done in

Lines 21-24 - Select statement

Lines 29-35 – Loop to run code 1000 times.

Testing task 2-

The following code was used to pass values and test the getter and setter methods.

First we created a new file called MyFileTester. We included the following methods to send data of various data types to the page file at the desired memory locations and retrieved it

public class MyFileTester { public static void main(String[] args) {….}

Then we create a Database testingDB and initialized the offsets using the following code.

```
String dirToCreateDB="testingDB";
SimpleDB.init(dirToCreateDB);
int intOffset=99;
int strOffset=20;
 int dateOffset=1;
 int byteAOffset =2 ;
 int boolOffset = 3 ;
 int shortOffset =  4 ;
```

We then created New pages to send data from this class to page class and retrieve back.

```
Page tempPage1 = new Page();

Block tempBlock = new Block("MyTempBlock", 6);
```

- Testing the string data handling:-

String strName = new String("Anuja Chandrashekhar Yawalkar");

tempPage1.setString(strOffset, strName);

String stringToRet = tempPage1.getString(strOffset);

tempPage1.write(tempBlock);

System.out.println("String to return ... " + stringToRet);

- Testing the Int Data Handling:-

tempPage1.setInt(intOffset, 99);

int n = tempPage1.getInt(intOffset);

```
System.out.println("Value of n=" + n);

tempPage1.setInt( n+1, 100);

tempPage1.write(tempBlock);

int newN = tempPage1.getInt(n+1);

System.out.println("Value @ 'n+1'=" + newN);
```

- Testing the Date Data Handling:-

```
String dateStr="19870819";

String theFormat="yyyyMMdd";

DateFormat   formatter = new SimpleDateFormat(theFormat);

Date testDateObj = new Date();

try {testDateObj = (Date) formatter.parse(dateStr); }

catch (ParseException e)

{e.printStackTrace();}

tempPage1.setDate(dateOffset, testDateObj);

tempPage1.write(tempBlock);

System.out.println("The date was: " +          tempPage1.getDate(dateOffset));

//byte[] byteArrayExample = new byte[]{3,2,5,4,1};

byte [] byteArrayExample = "Akond".getBytes();

String strToRet="" ;

tempPage1.setBytArray(byteAOffset, byteArrayExample);

strToRet = new String(tempPage1.getByteArray(byteAOffset));

System.out.println("Byte array cotents as string=" + strToRet );
```

- Testing the Boolean Data Handling:-

```
boolean boolToInsert = true ;

System.out.println("bools=" +  boolToInsert);

tempPage1.setBoolean(boolOffset, boolToInsert);

tempPage1.write(tempBlock);

System.out.println("boolean returned = " +  tempPage1.getBoolean(boolOffset));
```

- Testing the short Data Handling

```
short shortValToRet = 21474 ;

tempPage1.setShort(shortOffset, shortValToRet);

tempPage1.write(tempBlock);

System.out.println("short value to return = " + tempPage1.getShort(shortOffset));}}
```

Conclusion-

We had performed both the tasks and also noted the results in this report. Task 1 proved to be a useful implementation as it reduced the execution time. Task 2 was more concerned towards adding functionality to the system which we achieved using getter and setter methods. Overall it was a good learning experience and we got to explore new domains related to Database Systems i.e. file and memory management.