1) QUESTION: As instructed in the assignment fill in the table with the total runtime associated with each cache hit rate.

ANSWER:

Total Run time=Hit Ratio*Cache hit time+ Miss Ratio(cache and memory access time)
Given cache access time=1ns
Main memory access time=100ns
Miss ratio = 1- hit ratio

Hit rate:

1) 0%
   Total run time for one L1 and main memory access with 0% hit rate=
   0*1ns+100/100(1ns+100ns) =101ns

2) 10%
   Total run time for one L1 and main memory access with 10% hit rate
   0.1*1ns+90/100(1ns+100ns) = 91ns

3) 25%
   Total run time for one L1 and main memory access with 25% hit rate
   0.25*1ns+75/100(1ns+100ns) = 76ns

4) 50%
   Total run time for one L1 and main memory access with 50% hit rate
   0.5*1ns+50/100(1ns+100ns) =51ns

5) 75%
   Total run time for one L1 and main memory access with 75% hit rate
   0.75*1ns+25/100(1ns+100ns) = 26ns

6) 90%
   Total run time for one L1 and main memory access with 90% hit rate
   0.90*1ns+10/100(1ns+100ns) = 11ns

7) 100%
   Total run time for one L1 and main memory access with 100% hit rate
   1*1ns+0/100(1ns+100ns) = 1ns

If algorithm requires 10^9 such then,

1) 0%
   Total run time for 10^9 access between L1 and main memory with 0% hit rate=
   101ns*10^9=101sec

2) 10%
Total run time for 10^9 access between L1 and main memory with 10% hit rate= 91ns*10^9=91sec

3) 25%
Total run time for 10^9 access between L1 and main memory with 25% hit rate= 76ns*10^9=76sec

4) 50%
Total run time for 10^9 access between L1 and main memory with 50% hit rate= 51ns*10^9=51sec

5) 75%
Total run time for 10^9 access between L1 and main memory with 75% hit rate= 26ns*10^9=26sec

6) 90%
Total run time for 10^9 access between L1 and main memory with 90% hit rate= 11ns*10^9=11sec

7) 100%
Total run time for 10^9 access between L1 and main memory with 100% hit rate= 1ns*10^9=1sec

2) ***The Wages of Non-Sequential Disk I/O is Death:*** Imagine we have 10GB of data stored on two disks, each of which was written by a different data management subsystem. Peter Ponderous naively implemented his system in such a way that the 10GB is broken up into 10,000 chunks, each of 100,000 bytes, with each chunk stored on disk in a physically contiguous fashion, after which the disk head must perform a seek to move to the next chunk and continue reading. The other disk's data was written via a storage layer implemented by Sally Speed-Demon, who was more conscientious in designing her implementation. Sally's system stored the 10GB as 100 contiguous chunks, each of 10,000,000 bytes. Assume that we can read the data in a contiguous chunk at 100MB/sec, and that a seek requires 10 msec.

a) How long does it take us to read all of the data in each of Peter's and Sally's schemes?

For Peter's implementation: with the decrease in total bytes per chuck the random read time increases

Read time=Chunks*(seek time+bytes per chunk/read chunck rate)

=10,000*(seek time+ebt)=10,000(10ms+10^5/100Mb)

=27.7 min

To read the same size of Sally's data where there is increase in data size and reduced number of chunck's the read time decreases.

Read time =100(10ms+10^7/100Mb)=10.1sec

By what factor does Sally's scheme outperform Peter's?
As Peters read time is 27.7 min that is 100,010 sec and Sally's read time is 10.1 sec Sallys scheme is better by 99% and makes a significant read time difference.

c) Which would you rather wait around for?

Sally's read time is better.

***3) Failure in Large Datacenters*:** To perform large scale analyses on their social graph, Facecreep.com has built a data center with 10,000 servers, all of which need to be operational to complete a single run of the analysis. Each server is a Commercial Off the Shelf (COTS) unit purchased from The Dole Technologies with a MTTF (mean time to failure) of 36 months. Assume a month has 30 days.

a) What is the approximate mean time until the next server fails in the data center?

Solution:

The MTTF of a single server is 36 months.
1 month = 30 days
=36moths*30 days
=1080 days.

Total available servers 10,000
MTTF until the next server fails in the data center is 1080/10000=.108days
=.108 days* 24 hours
=2.592 hours.

b) Suppose that a single run of Facecreep's analysis requires 6 hours to complete, and that a failed server can be replaced in 30 minutes (the datacenter's ops staff are really on the ball and they keep spare machines in a closet on-site). Does the above scenario bode well for Facecreep's ability to generate analyses? Why or why not?

Solution:
Since there are 10,000 servers each server may take its turn and fail in 3 hours. The probability of a server failing in any given time is .0001. Since it requires 10,000 servers and all of them are running in parallel operation then we can keep the replica of data in a central or the main server and when the given server is failed the previous time stamp before failing operation can be stored in the main server and once the server is fixed in 30 min it can start its operation from its time stamped failure point.

c) List at least three ways you might deal with the problem illuminated by this question. Feel free to let your imagination run wild within reason.

Solution:

If a server fails, it's better to maintain clone of data in different servers. So, when there is failure there won't be much impact on the performance. This avoids single point failure. Geo distribution of servers and maintaining copies of data in many servers will be helpful.

4) More Failure in Large Datacenters: Through a shadowy deal arranged with the help of ethically dubious lobbyists, Facecreep has gained access to a secret government warehouse that contains many powerful objects, including x86 compatible servers that have a MTTF of 3000 years. These servers are great in that we don't expect to see any failures within the runtime of a single 6-hour graph analysis[1]. As before, all 10,000 machines are needed to complete a job. Much to their chagrin, Facecreep's engineers observe that one of the 10,000 machines is cursed. It sometimes becomes a *straggler* and takes 600 hours to do its part of the job, rather than the usual 6 hours.

a) What effect does the curse have on the completion time of the job, assuming we naively schedule 1/10,000-th of the job on each of the machines and that we can't report a result until they all finish their parts?

Solution:

Since the time taken for completion of a task is delayed even though the probability of the server failing MTTF is 3 years. It will impact the business as it takes 594 hours more to complete a single task. We have to wait for all servers to come to a completion, then it's possible to integrate the data from different servers. This process will take more time than usual as a result, the performance will be poor.

b) How might you remedy this situation to make things more tolerable?

Solution:

If possible, addition of better performing servers may improve it's run time performance.

5) ***Summing Cubes:*** Imagine we have the numbers [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] in a collection and we want to compute the sum of the cubes of then numbers,i.e.$1^3+2^3+3^3+...+10^3$ in the MapReduce style. Express the calculation in terms of map and reduce steps.

a) What function would you map on to the original input collection?

Map is a function which transforms items in some kind of list to another item and put them back in same kind of list

--input-1--|f(x)(Map)|---output

Where input-1=[1,2,3,4,5,6,7,8,9,10]

A = [1, 2, 3,4,5,6,7,8,9,10]

foreach (item in A) A[item] = A[item] * A[item]* A[item]

b) Show the collection that results from mapping your function on to the original input.

Map is a function which transforms items in some kind of list to another item and put them back in same kind of list

--input-1--|f(x)(Map)|---output

Where input-1=[1,2,3,4,5,6,7,8,9,10]

A = [1, 2, 3,4,5,6,7,8,9,10]

foreach (item in A) A[item] = A[item] * A[item]* A[item]

//map function

A= [1, 2, 3,4,5,6,7,8,9,10].Map(x => x * x * x)

Which in turn produces output:

1 : 1 => 1 * 1*1 : 1

2 : 2 => 2 * 2 *2: 8

c) What function would you reduce the intermediate result collection with?

As shown above have used the lambda function to reduce to the intermediate result

A= [1, 2, 3,4,5,6,7,8,9,10].Map(x => x * x * x)

d) Show the reduction and final result of the reduce step.

The final result is:

1 : 1 => 1 * 1*1 : 1

2 : 2 => 2 * 2 *2: 8

3 : 3 => 3 * 3 *3: 27

-

-

-
10:10=> 10*10*10:1000

6) At the Keyboard
QUESTION: Essential Scala 2.1.4 Exercises
ANSWERS:

```
scala> 1+2
res0: Int = 3

scala> "3".toInt
res1: Int = 3
```

```
scala> "foo".toInt
java.lang.NumberFormatException: For input string: "foo"
  at java.lang.NumberFormatException.forInputString(NumberFormatException.java:65)
  at java.lang.Integer.parseInt(Integer.java:580)
  at java.lang.Integer.parseInt(Integer.java:615)
  at scala.collection.immutable.StringLike$class.toInt(StringLike.scala:272)
  at scala.collection.immutable.StringOps.toInt(StringOps.scala:30)
  ... 33 elided

scala> println("foo")
foo

scala> println("foo".toInt)
java.lang.NumberFormatException: For input string: "foo"
  at java.lang.NumberFormatException.forInputString(NumberFormatException.java:65)
  at java.lang.Integer.parseInt(Integer.java:580)
  at java.lang.Integer.parseInt(Integer.java:615)
  at scala.collection.immutable.StringLike$class.toInt(StringLike.scala:272)
  at scala.collection.immutable.StringOps.toInt(StringOps.scala:30)
  ... 33 elided
```

QUESTION: Essential Scala 2.2.5 Exercise
ANSWERS:

```
scala> "foo".take(1)
res6: String = f

scala> 1.+(2).+(3)
res7: Int = 6

scala> 1+2+3
res8: Int = 6
```

QUESTION: Essential Scala 2.3.8 Exercises
ANSWERS:

```
scala> 42
res9: Int = 42

scala> true
res10: Boolean = true

scala> 123L
res11: Long = 123

scala> 42.0
res12: Double = 42.0
```

```
scala> 'a'
res13: Char = a

scala> "a"
res14: String = a

scala> "Hello World"
res15: String = Hello World

scala> println("Hello World")
Hello World
```