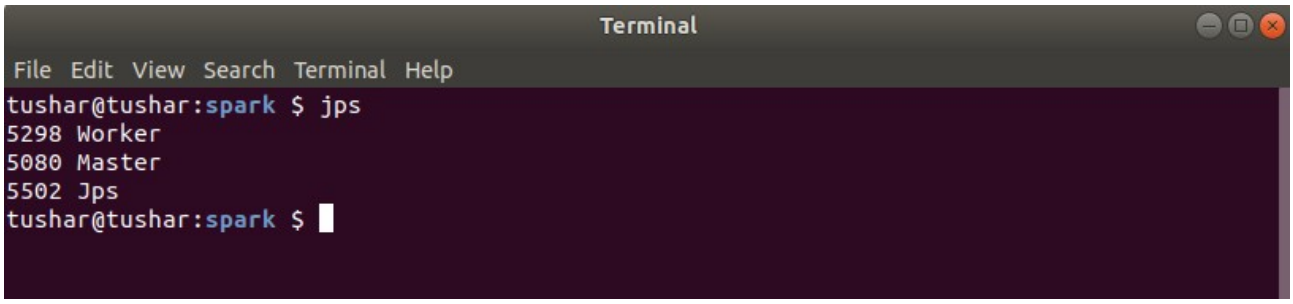
ASSIGNMENT

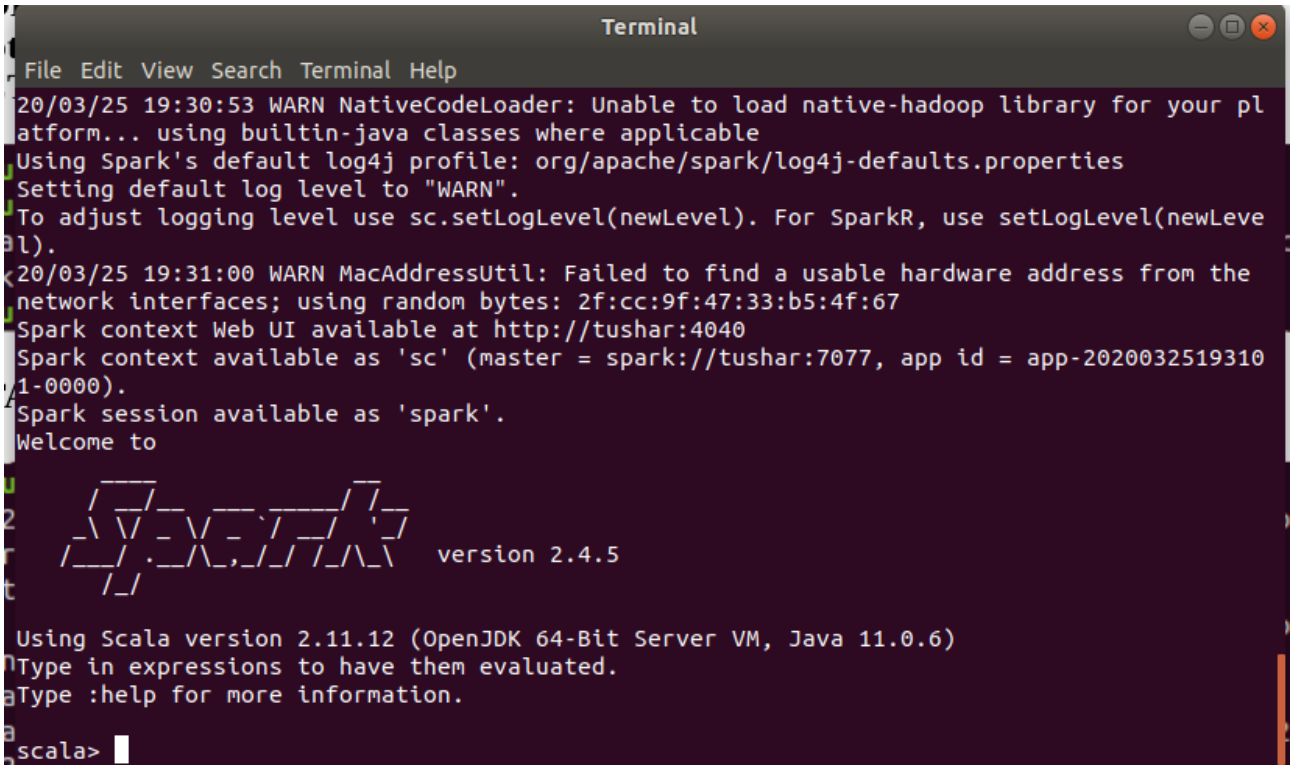
NAME-TUSHAR RAJ VERMA

Q1- Setup Spark stand alone cluster with 1 master and 2 worker nodes (refer the github link provided) and demonstrate running an application (using spark-shell) on this stand alone cluster.
This application should run minimum 4 tasks in 1 stage in 1 job . It will be good to see if this application tasks utilize both worker nodes. Display this output via spark UI.

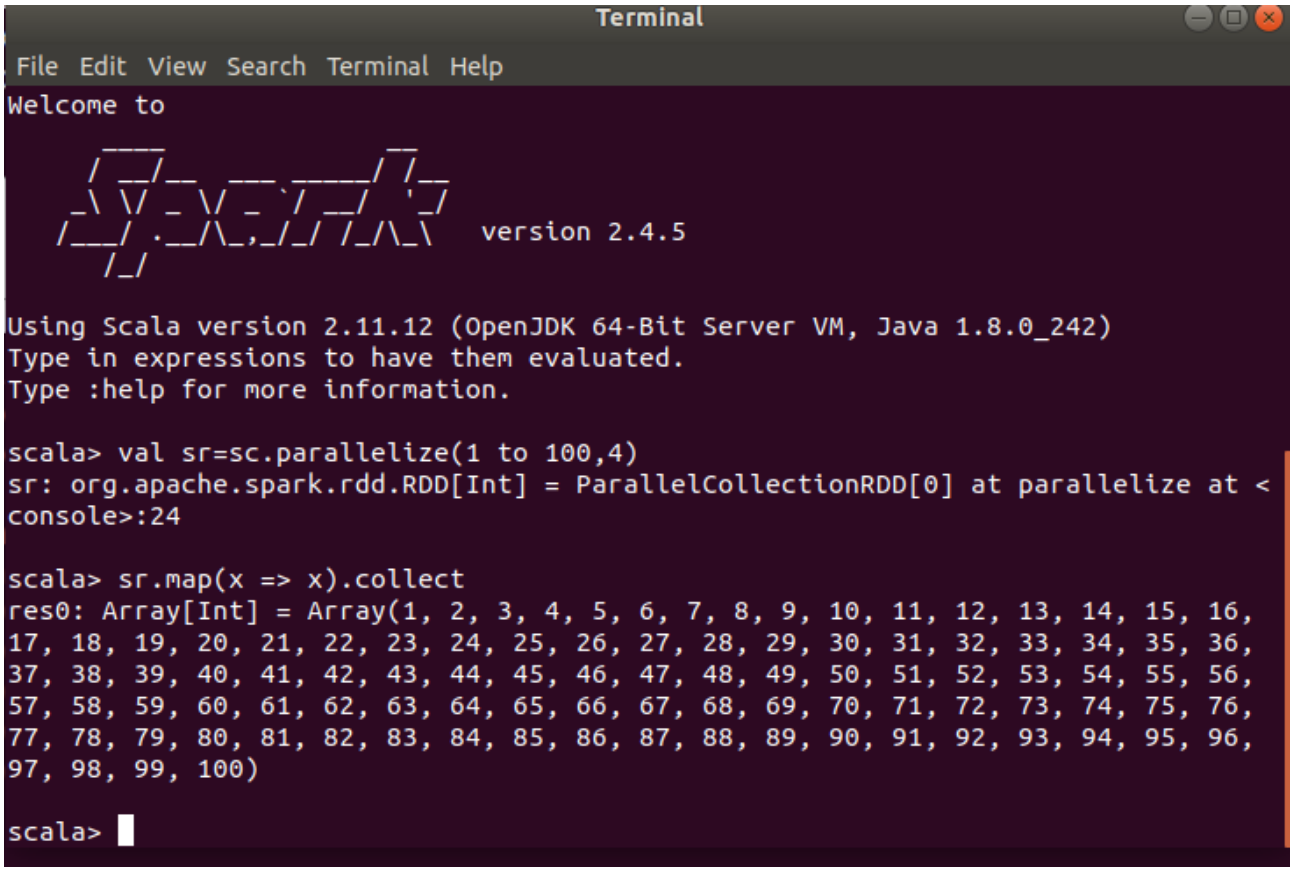
Ans-1
Spark stand alone cluster is set up with spark running



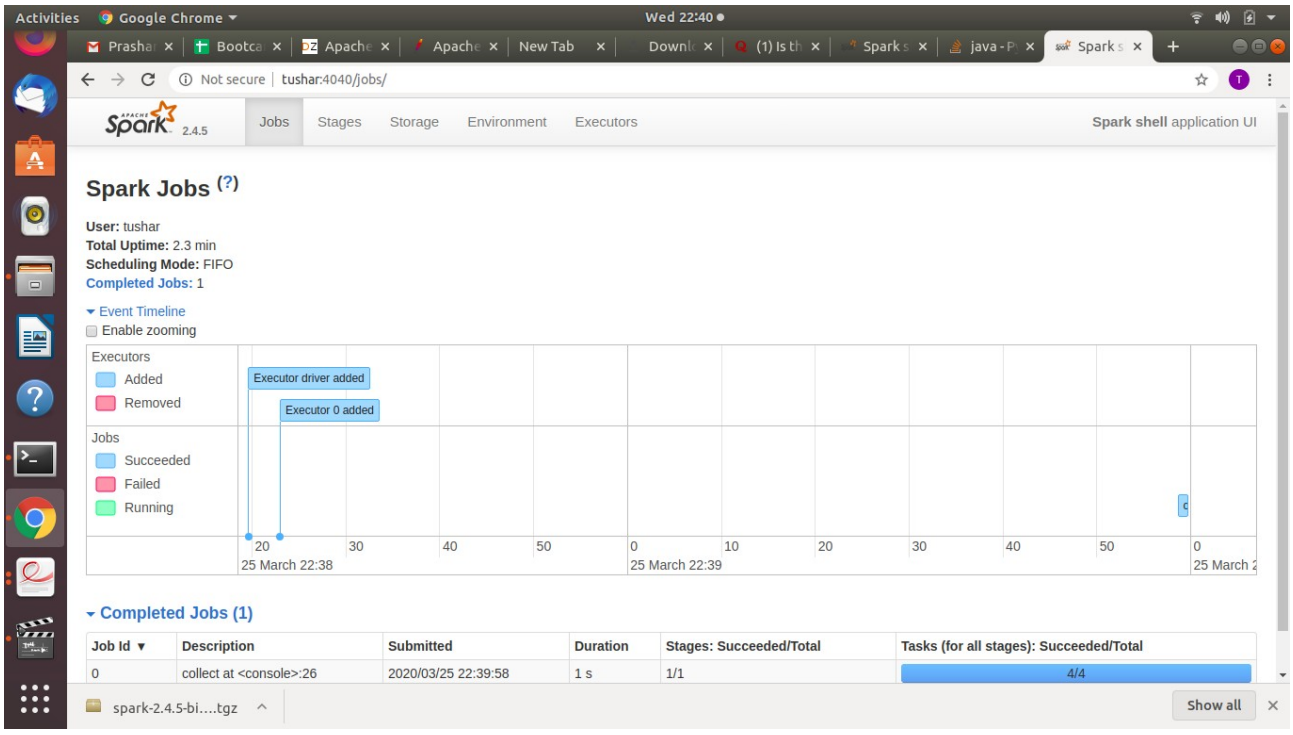
spark-shell running on both nodes



Application running



4 tasks in 1 stage in 1 job



Application tasks utilize both worker nodes

Spark 2.3.3 **Spark Master at spark://u1:7077**

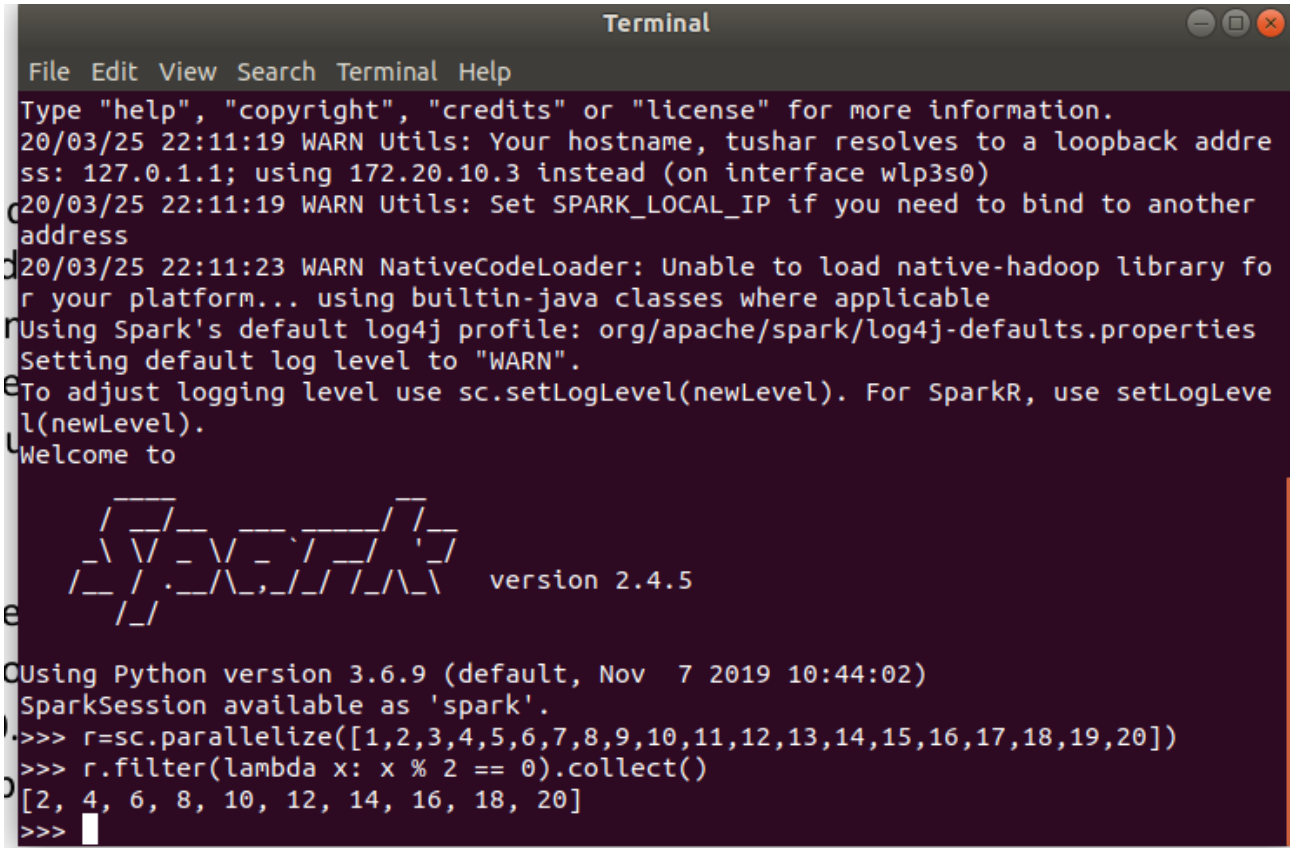
URL: spark://u1:7077
REST URL: spark://u1:6066 *(cluster mode)*
Alive Workers: 2
Cores in use: 2 Total, 2 Used
Memory in use: 2.8 GB Total, 2.0 GB Used
Applications: 1 Running, 1 Completed
Drivers: 0 Running, 0 Completed
Status: ALIVE

Workers (2)

Worker Id	Address	State	Cores	Memory
worker-20200321132116-192.168.56.101-35413	192.168.56.101:35413	ALIVE	1 (1 Used)	1443.0 MB (1024.0 MB Used)
worker-20200321132119-192.168.56.102-41313	192.168.56.102:41313	ALIVE	1 (1 Used)	1443.0 MB (1024.0 MB Used)

Running Applications (1)

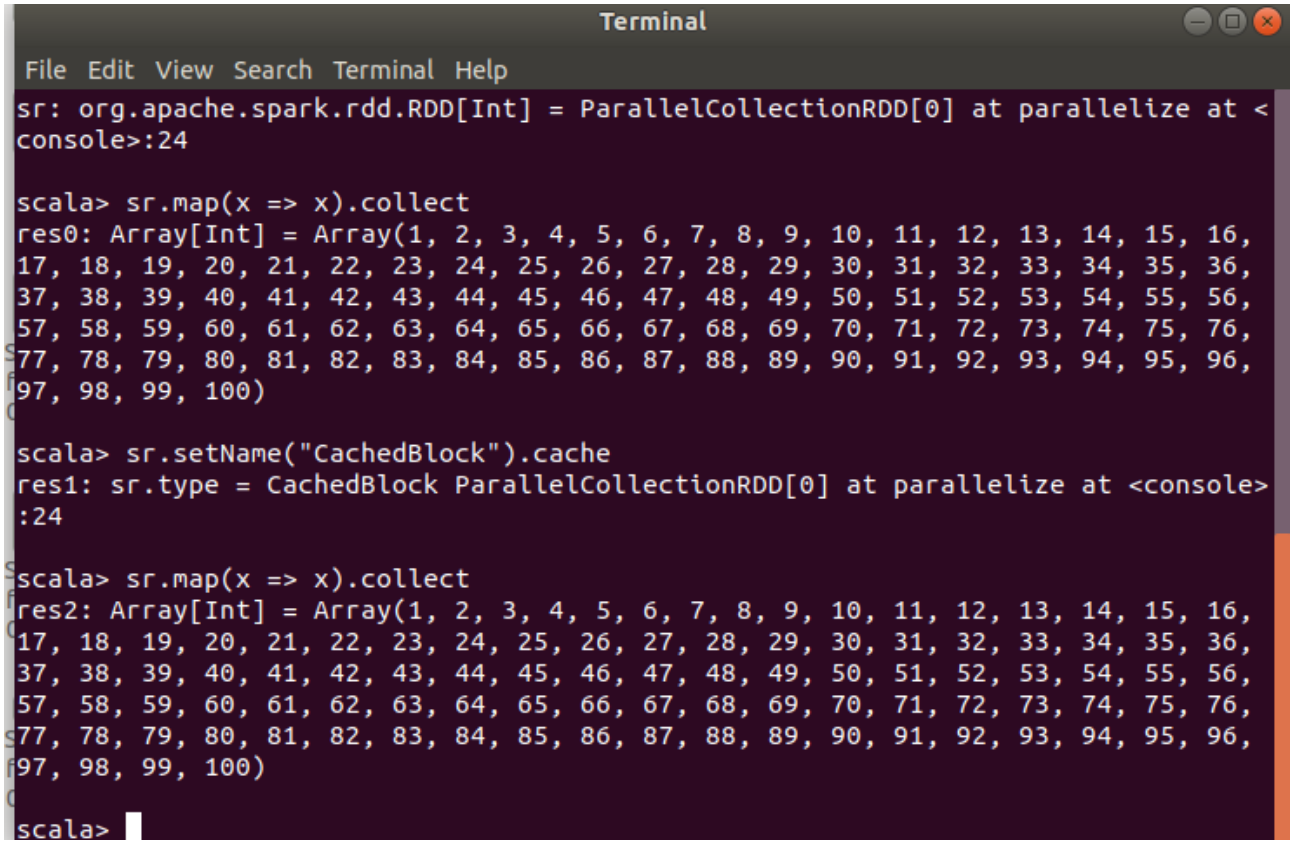
Q2. Using pyspark, demonstrate using a method of spark context that creates a series of RDDs
when working on a collection of numbers.Create a collection of numbers say list of 1 to 20. While using a method of spark context to create RDDs, print a list of even numbers. Demonstrate usage of cache and benefit in performance.
Ans-2 pyspark series(1 to 20) and printing even numbers using lambda function



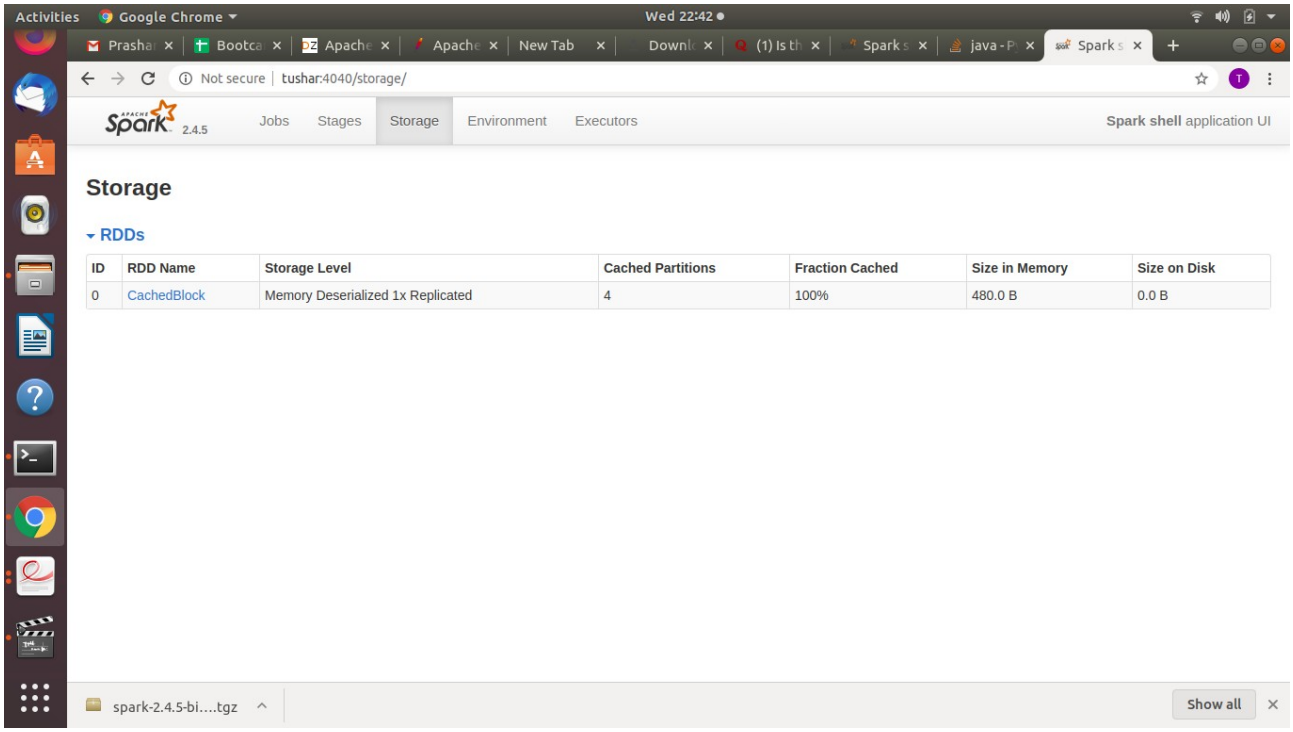
```
Terminal
File  Edit  View  Search  Terminal  Help
Type "help", "copyright", "credits" or "license" for more information.
20/03/25 22:11:19 WARN Utils: Your hostname, tushar resolves to a loopback addre
ss: 127.0.1.1; using 172.20.10.3 instead (on interface wlp3s0)
20/03/25 22:11:19 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another
address
20/03/25 22:11:23 WARN NativeCodeLoader: Unable to load native-hadoop library fo
r your platform... using builtin-java classes where applicable
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLeve
l(newLevel).
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 2.4.5
      /_/

Using Python version 3.6.9 (default, Nov  7 2019 10:44:02)
SparkSession available as 'spark'.
>>> r=sc.parallelize([1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20])
>>> r.filter(lambda x: x % 2 == 0).collect()
[2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
>>>
```

Usage of cache



“CacheBlock” in gui



BENEFITS IN PERFORMANCE:-

Caching requires a consideration for number of nodes available, the relative priority of each RDD,
and the amount of memory available for caching, it often requires a good deal of trial-and-error.
And the more RDDs you have to consider, all the more complex this method becomes.
That's where automation plays a role. A tool that provides full-stack performance intelligence can
tell you if caching is a viable option in any particular case, and, if it would, how you could know
what to cache to get maximum performance. It can tell you the performance you can achieve if you
had five additional servers available, and whether adding those five servers to the mix makes sense.
There are generally three factors to consider in tuning memory usage: the amount of memory used
by your objects, the cost of accessing those objects, and the overhead of "garbage collection" (if
you have high turnover in terms of objects). Automating performance management across the stack
takes these factors into consideration, relieving you of the burden of manually tuning your system.
And:-
Due to Caching we get:-
1.) It Reduced latency
2.) It provides Content availability
3.) It Avoids network congestion
4.) Increases Response time:
5.) Increases System throughput
6.) It reduces the overhead from server resources.
7.) It increases the performance of the application by serving user with cached output.