Programming Assignment 3: MapReduce Compute Framework

Rohan Sadale(sadal005@umn.edu)
Akshay Kulkarni (kulka182@umn.edu)

Description

In this programming assignment, we have implemented a simple MapReduce-like compute framework for implementing a sort program. The framework will receive jobs from a client, split job into multiple tasks, and assign tasks to compute nodes which perform the computations. This framework would be fault-tolerant and should be able to guarantee the correct result even in the presence of compute node faults.

Design

The system would consist of three components:

1. Client:
   The client will send a sort job to the Server. The job will include a filename which stores data to be sorted. When the job is done, the client will get the filename as a result of job which stores sorted data. The sorted file will be stored in Data/output/ folder

2. Server:
   The server receives the sorting job from the client. It splits the job (input data) into multiple tasks (chunks) and assigns each task to a compute node. Server will maintain the status of job such as number of tasks running, number of Compute Nodes, number of tasks running on each Compute Node. When the job is completed, the server will print the elapsed time to run it along with the output filename which includes sorted output returned back to the client.

3. Compute Nodes:
   Compute nodes will execute tasks (either sort or merge) sent to them by the server. Each compute node will run on different machines. Each compute node is a multi-threaded server serving multiple sort and merge tasks at same time.

Workflow

1. Client sends a filename to be sorted to the Server. The file should be present in Data/input/ folder.

2. Server analyzes the file size, calculates the number of sort tasks to be generated based on chunk size.
3. For each sort task server server creates a thread and sends request to compute node for sorting required portion of the file. To ensure that large number of threads are created simultaneously we have use ExecutorPool JAVA API which basically limits number of threads that could run simultaneously.
4. During this process some node may fail, server in that case re-assigns the task to different compute Node. Heartbeat is used to check whether node is alive or dead. Also before executing any task (sort or merge) compute node generates random number between 0-1 and checks if it is less than fail probability. If yes, then it abandons the computation.
5. Server waits till all the chunks are sorted. All the intermediate sorted file are stored in directory Data/intermediate/
6. If proactive fault tolerance is enabled then server, checks if task has been finished at any node. If yes then it sends signal to all nodes carrying out that task to abort.
7. Once all chunks are sorted, server merges these files. Number of files to be merged is govern by parameter MergeTaskSize specified in configuration file. This process of merging is done continuously until we have merged all our files.
8. Once all files have been merged, final output file is moved to Data/output/ folder.

<u>Implementation</u>

1. Config.txt
    a. ServerIP: Address of Server
    b. ServerPort: Port no for Server
    c. InputDirectory: Directory for input files
    d. IntermediateDirectory: Directory for intermediate files generated
    e. OutputDirectory: Directory for output file generated
    f. ChunkSize: Specify Chuck size in bytes
    g. MergeTaskSize: Specify the number of merge tasks to be combined
    h. TaskReplication: Specify number of tasks to be spawned (for ProActive fault tolerance)
    i. FailProbability: Specify fail probability between 0-1
    j. Proactive: The flag is set to 1 if we want our system to adjust for proactive fault tolerance else set to 0.

2. Interface exposed by Server
    a. doJob() : This function takes name of the file that is to be sorted and returns name of the output file. The function internally breaks files into chunks and sends those chunks to compute node for sorting. Once all the chunks are sorted, files are grouped and merge request is sent to compute nodes. This process is done recursively until all sorted chunks are merged.

    b. Join() : Allows new Compute Node to join the system

3. Interface exposed by Compute Node
    a. doSort() : Sorts a chunk of data specified by offset and count. This function takes filename, offset and size(no of bytes) as input, sorts the chunk and creates an intermediate file containing sorted chuck data. This file will be stored in Data/intermediate/ folder. The name of intermediate file will be returned back to server.
    b. doMerge() : Takes a list of files as input and merges them in a single file. We have used external sort-merge technique to merge these files. Thus our approach would work efficiently on merging very large files. The name of merged file will be returned back to server.
    c. ping() : is used to check connection between Server and Compute Node. Returns true if Server can call and get a reply from this function.

<u>How to Run Components</u>

1. Scp all files to server.
2. The file structure on server should be:-
    a. Project Folder
        i. jars(folder)
            1. slf4j-api-1.7.14.jar
            2. libthrift-0.9.1.jar
        ii. files(folder)
            1. all files
3. Generate stub files
    a. thrift --gen java JobStatus.thrift
    b. thrift --gen java JobTime.thrift
    c. thrift --gen java computeNode.thrift
    d. thrift --gen java node.thrift
    e. thrift --gen java server.thrift

4. Compile Files
   javac -cp .:../jars/libthrift-0.9.1.jar:../jars/slf4j-api-1.7.14.jar:gen-java/ *.java

   This will compile all the java files and create .class files. This assumes jars required to compile these files are placed in jars folder. If not, one should modify and provide appropriate path for jar.

5. Make sure that you start Server each Compute Node and Client in separate sessions(different terminal tabs)

6. Data Folder Structure
    a. Create a folder named Data.
    b. Create three subdirectories - input, intermediate and output
    c. Data
        i. Input
        ii. Intermediate
        iii. Output

7. Ensure that required parameters are filled in config.txt file

8. Start Server (Master)
   java -cp .:../jars/libthrift-0.9.1.jar:../jars/slf4j-api-1.7.14.jar:gen-java/ Server config.txt

9. Start Compute Nodes
   java -cp .:../jars/libthrift-0.9.1.jar:../jars/slf4j-api-1.7.14.jar:gen-java/ ComputeNode config.txt 9091

10. Start Client
    java -cp .:../jars/libthrift-0.9.1.jar:../jars/slf4j-api-1.7.14.jar:gen-java/ Client config.txt fileName

    The filename specified here should be present in Data/input/ folder.

11. Comparing sorted output
    diff -w outputFileFromProgram actualSortedFile

    Our system appends an extra space character at the end of sorted output file. This is the reason we use diff -w to compare the output file with actual sorted file.

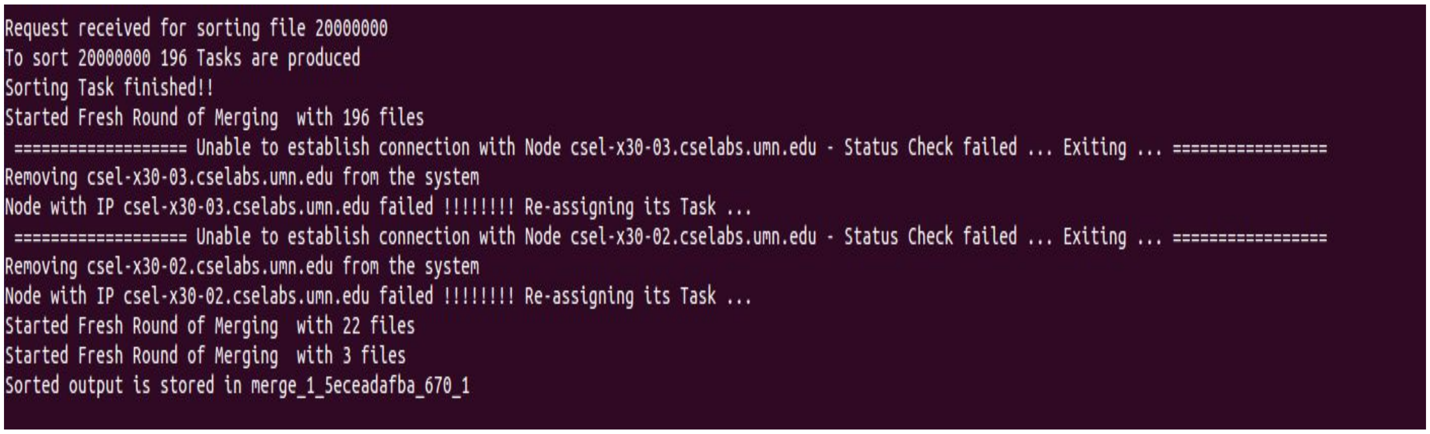
Testcases

*We have implemented our system in both ProActive and Non-ProActive mode. This is configurable through the file **config.txt**. In the file if Proactive=0, then the system will run without Proactive functionality. If Proactive=1, then the system will run with Proactive functionality. In this case, the TaskReplication parameter will come in effect.*

When system is in Proactive mode, the number of tasks generated would be significantly larger as tasks are replicated across ComputeNodes. Thus, mapreduce jobs will take longer time to run.

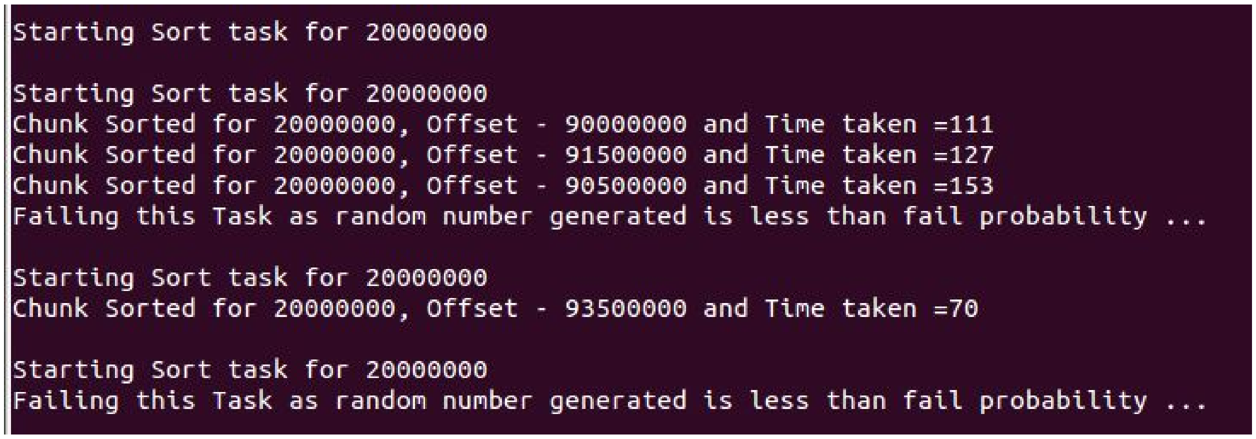
1. **Killing nodes:**

We have tested fault tolerance of our system by killing nodes. When node is killed, it's tasks will be assigned to another node. Node can be killed by pressing Ctrl+C.



```
Request received for sorting file 20000000
To sort 20000000 196 Tasks are produced
Sorting Task finished!!
Started Fresh Round of Merging  with 196 files
 ================== Unable to establish connection with Node csel-x30-03.cselabs.umn.edu - Status Check failed ... Exiting ... =================
Removing csel-x30-03.cselabs.umn.edu from the system
Node with IP csel-x30-03.cselabs.umn.edu failed !!!!!!!! Re-assigning its Task ...
 ================== Unable to establish connection with Node csel-x30-02.cselabs.umn.edu - Status Check failed ... Exiting ... =================
Removing csel-x30-02.cselabs.umn.edu from the system
Node with IP csel-x30-02.cselabs.umn.edu failed !!!!!!!! Re-assigning its Task ...
Started Fresh Round of Merging  with 22 files
Started Fresh Round of Merging  with 3 files
Sorted output is stored in merge_1_5eceadafba_670_1
```

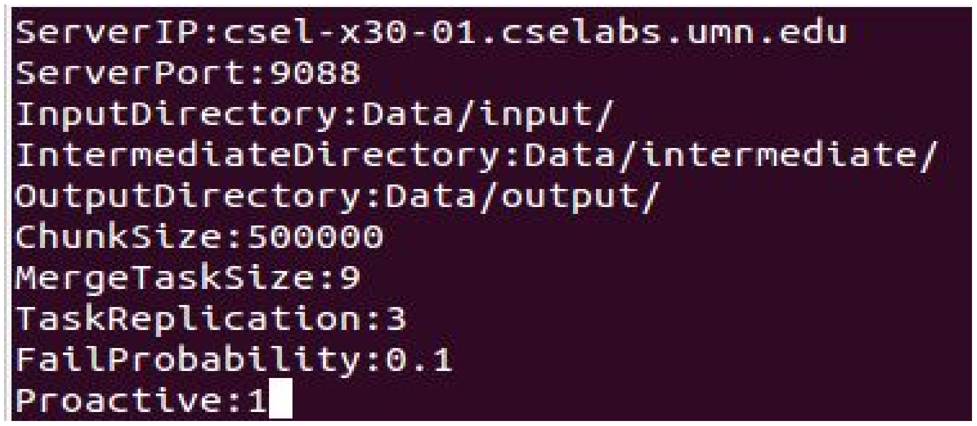
2. **<u>Fail Probability:</u>**
We have tested fail probability of tasks on a particular node. This is same as failing node and assigning its task to another node. The difference is that in this case, node will be still running even if certain task is failed. The failed task is assigned to different node. We have measured performance of multiple fail probabilities as shown in analysis section below.



```
Starting Sort task for 20000000

Starting Sort task for 20000000
Chunk Sorted for 20000000, Offset - 90000000 and Time taken =111
Chunk Sorted for 20000000, Offset - 91500000 and Time taken =127
Chunk Sorted for 20000000, Offset - 90500000 and Time taken =153
Failing this Task as random number generated is less than fail probability ...

Starting Sort task for 20000000
Chunk Sorted for 20000000, Offset - 93500000 and Time taken =70

Starting Sort task for 20000000
Failing this Task as random number generated is less than fail probability ...
```

3. **<u>Running various test cases:</u>**
   a. To run different combinations of test cases, you would need to make appropriate modifications in config.txt. Once changes are made in config.txt, all the Compute Nodes and Server should be restarted.
   b. Change chunk size: Specify the chunk size that you want in bytes. This would be used by doSort() in chunking data from input file.
   c. You can change number of files to merge at same time by changing value of MergeTaskSize

d. In ProActive mode, to incorporate number of task replications, you can specify the number of replications for each task. We have assumed that number of task replications will be always less than number of active nodes.
e. You can specify fail probability if you want to test the system in extreme conditions. We have tested on fail probabilities - 0.1, 0.2, 0.3, 0.4 and 0.5. As fail probabilities increase, number of task failures on each node will increase. This can significantly affect the expected output time.
f. Proactive: If this is set to 1, then our system enters into ProActive mode (Extra Credit). If this is set to 0, then our system will just handle fault tolerance and fail probability.



```
ServerIP:csel-x30-01.cselabs.umn.edu
ServerPort:9088
InputDirectory:Data/input/
IntermediateDirectory:Data/intermediate/
OutputDirectory:Data/output/
ChunkSize:500000
MergeTaskSize:9
TaskReplication:3
FailProbability:0.1
Proactive:1
```

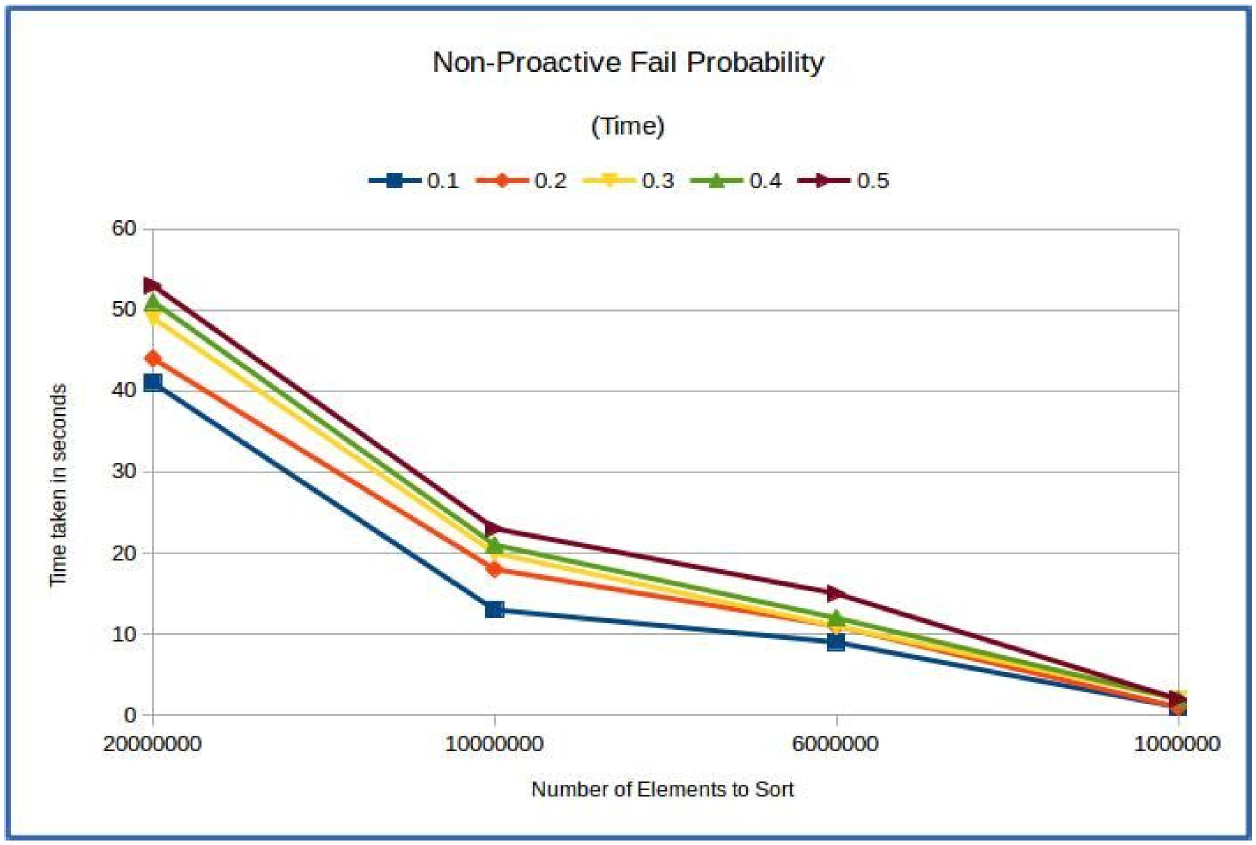
Performance Analysis

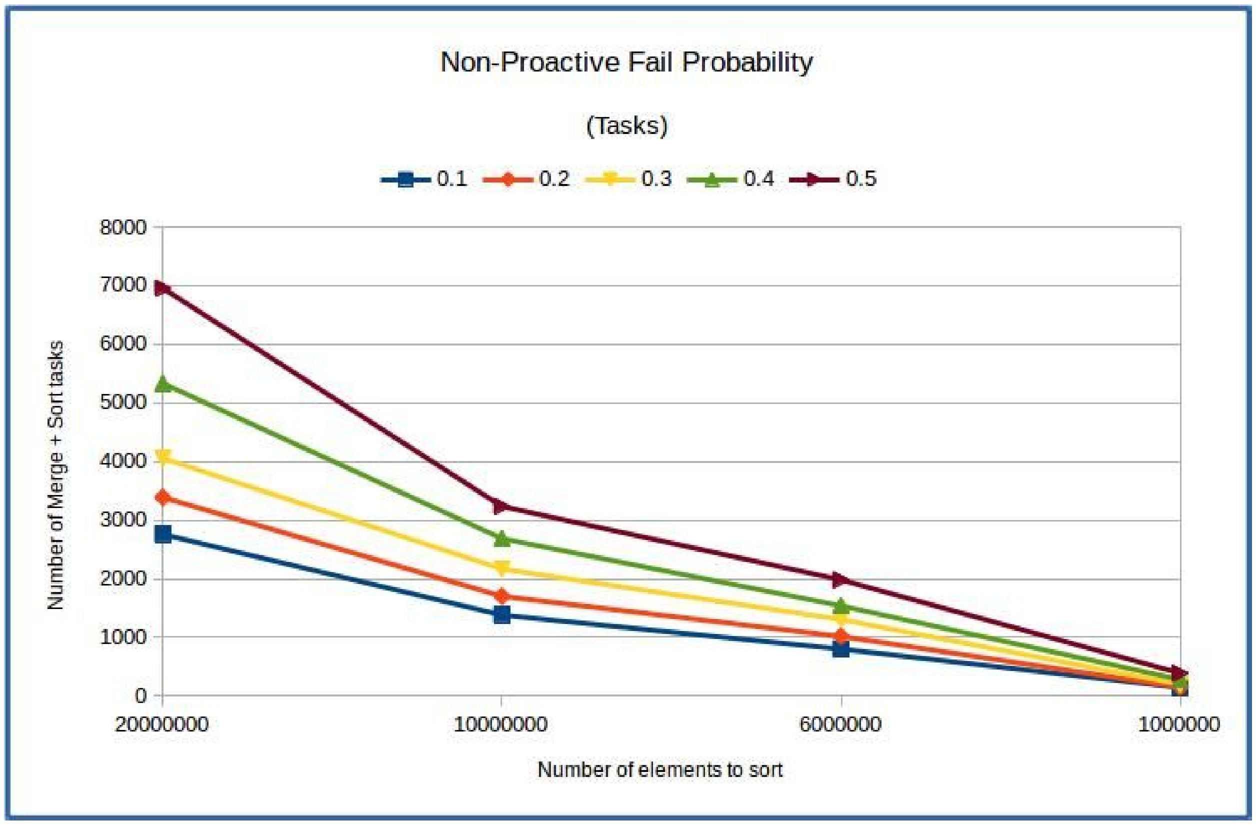
1. Basic Case (Non-ProActive mode)

Performance Analysis is done by keeping below parameters in config.txt file
Chunk-Size = 500000(Bytes)
Merge Task Size = 8
Number of compute Nodes = 8
Proactive = 0

FileName	Fail Probability	Total Sort Tasks	Failed Sort Tasks	Total Merge Task	Failed Merged Tasks	Time(in secs)
20000000	0.5	3994	2038	601	320	53
10000000	0.5	1908	930	268	126	23
6000000	0.5	1137	550	190	103	15
1000000	0.5	204	106	47	31	2

20000000	0.4	3311	1355	473	192	51
10000000	0.4	1667	689	235	93	21
6000000	0.4	958	371	148	61	12
1000000	0.4	170	72	27	11	2
20000000	0.3	2733	777	411	130	49
10000000	0.3	1428	450	215	73	20
6000000	0.3	862	275	127	40	11
1000000	0.3	132	34	25	9	2
20000000	0.2	2464	508	347	66	44
10000000	0.2	1240	262	170	28	18
6000000	0.2	739	152	106	19	11
1000000	0.2	113	15	20	4	1
20000000	0.1	2188	232	308	27	41
10000000	0.1	1085	107	165	23	13
6000000	0.1	636	49	101	14	9
1000000	0.1	112	14	18	2	1


Non-Proactive Fail Probability
(Time)
0.1
0.2
0.3
0.4
0.5
Time taken in seconds
0
10
20
30
40
50
60
20000000
10000000
6000000
1000000
Number of Elements to Sort


Non-Proactive Fail Probability
(Tasks)
0.1
0.2
0.3
0.4
0.5
Number of Merge + Sort tasks
0
1000
2000
3000
4000
5000
6000
7000
8000
20000000
10000000
6000000
1000000
Number of elements to sort

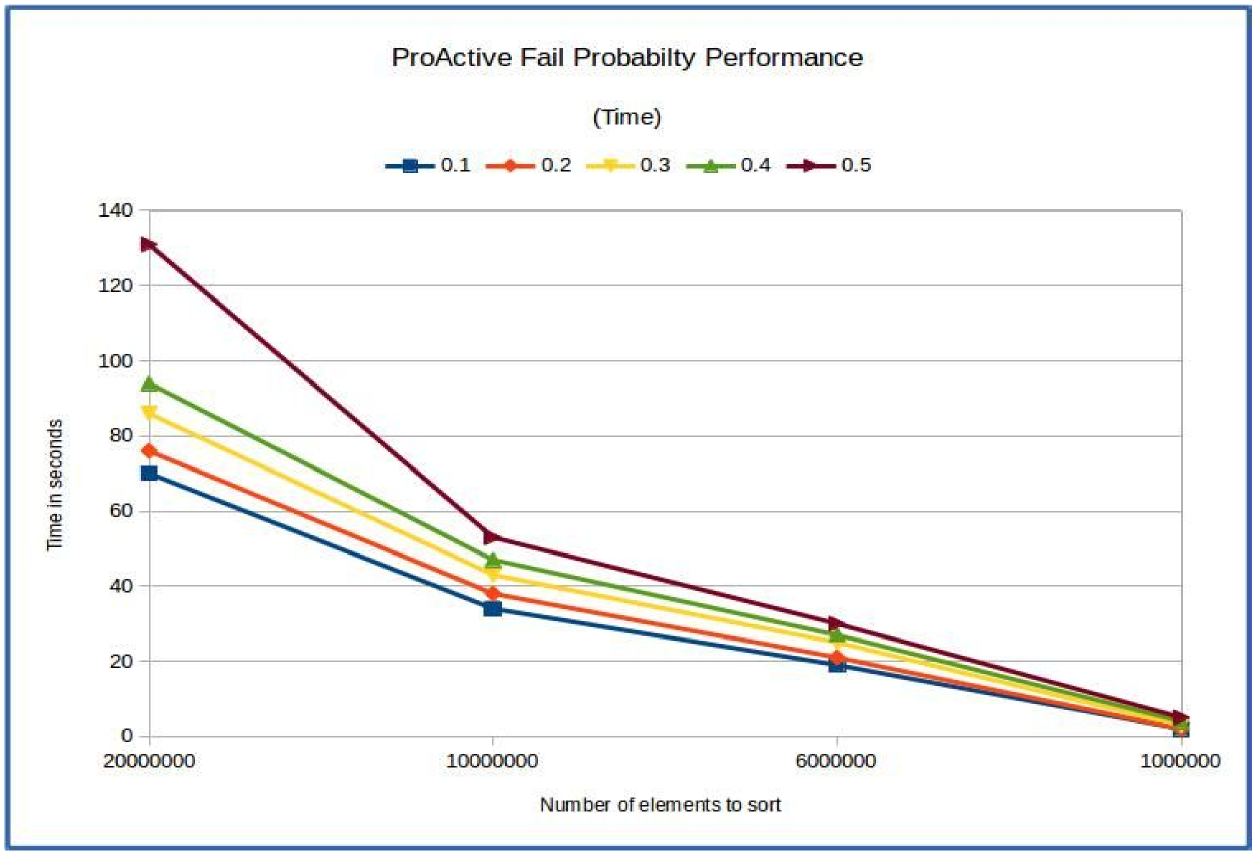
Performance Analysis is done by keeping below parameters in config.txt file
Chunk-Size = 1000000(Bytes)
Merge Task Size = 8;**Number of compute Nodes =** 8
Proactive = 0

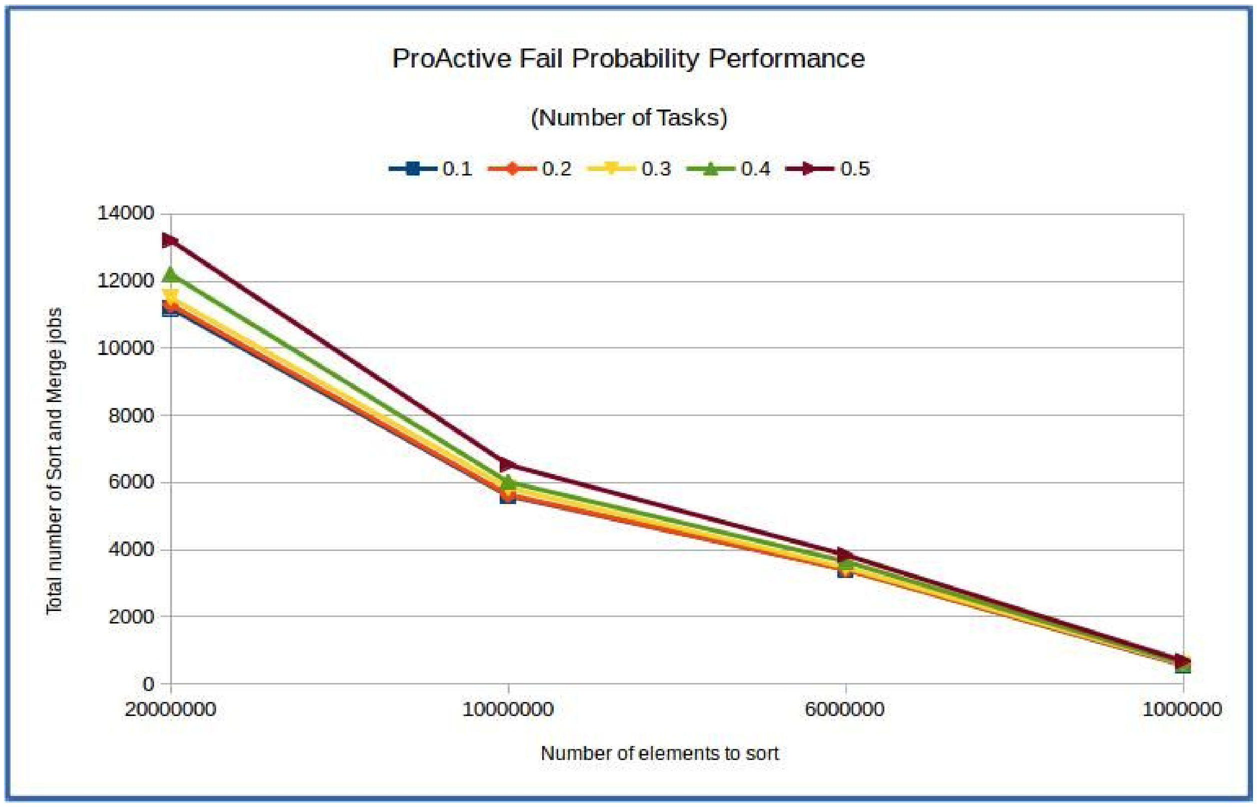
FileName	Fail Probability	Total Sort Tasks	Failed Sort Tasks	Total Merge Task	Failed Merged Tasks	Time(in secs)
20000000	0.5	194	96	38	22	58
10000000	0.5	90	41	23	15	15
6000000	0.5	48	18	5	0	10
1000000	0.5	12	7	1	0	1.3
20000000	0.4	163	65	28	12	55
10000000	0.4	89	40	11	3	13
6000000	0.4	50	20	7	2	10
1000000	0.4	8	3	1	0	1.3
20000000	0.3	145	47	22	6	53
10000000	0.3	68	19	9	1	12
6000000	0.3	47	17	7	2	10
1000000	0.3	9	4	1	0	1.12
20000000	0.2	121	23	24	8	51
10000000	0.2	62	13	8	0	11.3
6000000	0.2	38	8	6	1	9.78
1000000	0.2	5	0	1	0	1.1
20000000	0.1	112	14	17	1	50
10000000	0.1	55	6	10	2	11
6000000	0.1	33	3	5	0	9.4
1000000	0.1	5	0	2	1	1

2. <u>ProActive Mode</u>

Performance Analysis is done by keeping below parameters in config.txt file
Chunk-Size = 500000(Bytes)
Merge Task Size = 8;**Number of compute Nodes =** 8
Proactive = 1

FileName	Fail Probability	Total Sort Tasks	Failed Sort Tasks	Total Merge Task	Failed Merged Tasks	Time(in secs)
20000000	0.5	6756	4800	972	691	131
10000000	0.5	3300	2322	525	383	53
6000000	0.5	1965	1378	291	204	30
1000000	0.5	348	250	51	35	5
20000000	0.4	6309	4353	915	634	94
10000000	0.4	3123	2145	444	302	47
6000000	0.4	1881	1294	279	192	27
1000000	0.4	312	214	51	35	4
20000000	0.3	6015	4059	858	577	86
10000000	0.3	3027	2049	453	311	43
6000000	0.3	1803	1216	267	180	25
1000000	0.3	306	208	48	32	3
20000000	0.2	5925	3969	843	562	76
10000000	0.2	2949	1971	429	287	38
6000000	0.2	1773	1186	267	180	21
1000000	0.2	294	196	48	32	2
20000000	0.1	5874	3918	843	562	70
10000000	0.1	2937	1959	426	284	34
6000000	0.1	1780	1193	261	174	19
1000000	0.1	290	192	44	28	2


ProActive Fail Probabilty Performance
(Time)
0.1
0.2
0.3
0.4
0.5
Time in seconds
140
120
100
80
60
40
20
0
20000000
10000000
6000000
1000000
Number of elements to sort


ProActive Fail Probability Performance
(Number of Tasks)
0.1
0.2
0.3
0.4
0.5
Total number of Sort and Merge jobs
14000
12000
10000
8000
6000
4000
2000
0
20000000
10000000
6000000
1000000
Number of elements to sort

Performance Analysis is done by keeping below parameters in config.txt file
Chunk-Size = 1000000(Bytes)
Merge Task Size = 8;**Number of compute Nodes =** 8
Proactive = 1

FileName	Fail Probability	Total Sort Tasks	Failed Sort Tasks	Total Merge Task	Failed Merged Tasks	Time(in secs)
20000000	0.5	333	235	57	41	92
10000000	0.5	168	119	30	22	22
6000000	0.5	111	81	18	13	15
1000000	0.5	15	10	3	2	1.9
20000000	0.4	318	220	57	41	81
10000000	0.4	156	107	24	16	21
6000000	0.4	96	66	15	10	14.8
1000000	0.4	15	10	3	2	1.8
20000000	0.3	300	202	48	32	71
10000000	0.3	153	104	27	19	20.4
6000000	0.3	93	63	15	10	14
1000000	0.3	14	9	3	2	1.4
20000000	0.2	294	196	48	32	66
10000000	0.2	147	98	24	16	20
6000000	0.2	90	60	15	10	13.7
1000000	0.2	14	9	3	2	1.1
20000000	0.1	290	192	44	28	61
10000000	0.1	144	95	22	14	19
6000000	0.1	88	58	14	9	13.4
1000000	0.1	14	9	3	2	1