CLOUD COMPUTING

PROJECT-1

RAJA JAYA CHANDRA MANNEM - rxm124330@utdallas.edu

1. HDFS attributes from given cluster:

General:

Block size: 128MB - 134217728Bytes from `dfs.block.size` property in *hdfs-site.xml*.Minimum split size = 0MB from *`Mapred.min.split.size = 0`*;

Replication factor =3 from `dfs.replication=3`

Sorting and shuffling parameters:

Io sort factor: 10

io.sort.spill.percent 0.80

io.sort.record.percent 0.05

io.sort.mb 100 MB

```
cloud@master:~/hadoop_logs/userlogs/job_2014021502
17_0012$ cat /proc/meminfo

MemTotal:       16344888 kB
MemFree:          552168 kB
Buffers:          486060 kB
Cached:         12724464 kB
SwapCached:            0 kB
Active:         11473648 kB
Inactive:        3322444 kB
Active(anon):    1520496 kB
Inactive(anon):   302224 kB
Active(file):    9953152 kB
Inactive(file):  3020220 kB
Unevictable:           0 kB
Mlocked:               0 kB
SwapTotal:      16654332 kB
SwapFree:       16654244 kB
Dirty:               104 kB
Writeback:             0 kB
AnonPages:       1585644 kB
Mapped:           111404 kB
Shmem:            237152 kB
```

```
cloud@master:~/hadoop_logs/userlogs/job_2014021502
17_0012$  cat /proc/cpuinfo

8 Cores
processor       : 0,1,2,3,4,5,6,7
vendor_id       : GenuineIntel
cpu family      : 6
model           : 42
model name      : Intel(R) Core(TM) i7-2600 CPU @
3.40GHz
stepping        : 7
cpu MHz         : 1600.000
cache size      : 8192 KB
clflush size    : 64
cache_alignment : 64
address sizes   : 36 bits physical, 48 bits virtual

Linux version 3.0.0-12-generic (buildd@crested)
(gcc version 4.6.1 (Ubuntu/Linaro 4.6.1-9ubuntu3)
) #20-Ubuntu SMP Fri Oct 7 14:56:25 UTC 2011
```

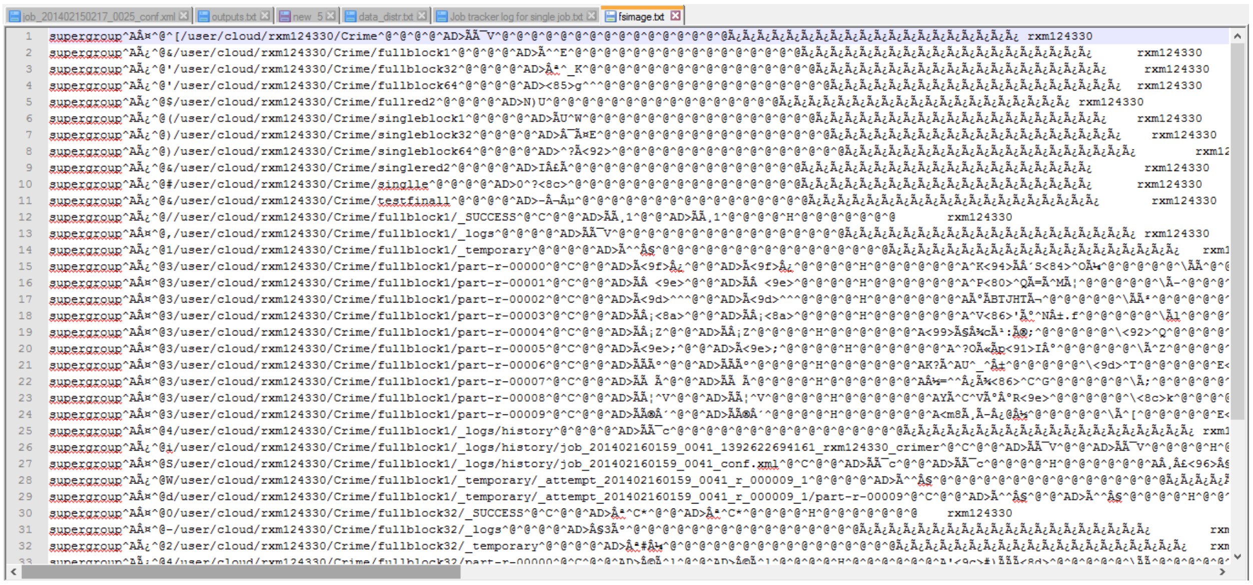
2. Data Distribution:

Specification for Given Cluster: cluster 05

1. 4 nodes   master: namenode, jobtracker, secondarynamenode
   slave1, 2, 3: datanode, tasktracker
2. hadoop version: 1.1.0

`${dfs.`**`name`**`.dir}/`
`└── current/`
`    ├── VERSION`
`    ├── edits`
`    ├── fsimage`
`    └── fstime`
	`${dfs.`**`data`**`.dir}/`
	`└── current/`
	`    ├── VERSION`
	`    ├── blk_<id_1>`
	`    ├── blk_<id_1>.meta`
	`    ├── blk_<id_2>`
	`    ├── blk_<id_2>.meta`
	`    ├── ...`
	`    ├── blk_<id_64>`
	`    ├── blk_<id_64>.meta`
	`    ├── subdir0/`
	`    ├── subdir1/`
	`    ├── ...`
	`    └── subdir63/`

When Hadoop –put is used, the files (2.18GB) is split and distributed into several files in datanode . The corresponding replication and locations of these files are maintained by name node. These files are replicated **3** times into different slave nodes.


job_201402150217_0025_conf.xml
outputs.txt
new 5
data_distr.txt
Job tracker log for single job.txt
fsimage.txt

```
supergroup^AÂ¤^@^[/user/cloud/rxm124330/Crime^@^@^@^@^AD>ÃÃ¯V^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿ rxm124330
supergroup^AÃ¿^@&/user/cloud/rxm124330/Crime/fullblock1^@^@^@^@^AD>Ã^^E^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿      rxm124330
supergroup^AÃ¿^@'/user/cloud/rxm124330/Crime/fullblock32^@^@^@^@^AD>Âª^_K^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿     rxm124330
supergroup^AÃ¿^@'/user/cloud/rxm124330/Crime/fullblock64^@^@^@^@^AD><85>g^^^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿  rxm124330
supergroup^AÃ¿^@$/user/cloud/rxm124330/Crime/fullred2^@^@^@^@^AD>N)U^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿ rxm124330
supergroup^AÃ¿^@(/user/cloud/rxm124330/Crime/singleblock1^@^@^@^@^AD>ÃU^W^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿    rxm124330
supergroup^AÃ¿^@)/user/cloud/rxm124330/Crime/singleblock32^@^@^@^@^AD>Ã¯Ã¤E^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿    rxm124330
supergroup^AÃ¿^@)/user/cloud/rxm124330/Crime/singleblock64^@^@^@^@^AD>^?Ã<92>^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿       rxm12
supergroup^AÃ¿^@&/user/cloud/rxm124330/Crime/singlered2^@^@^@^@^AD>IÂ£Ã^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿      rxm124330
supergroup^AÃ¿^@#/user/cloud/rxm124330/Crime/singlle^@^@^@^@^AD>0^?<8c>^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿      rxm124330
supergroup^AÃ¿^@&/user/cloud/rxm124330/Crime/testfinall^@^@^@^@^AD>-Â¬Âµ^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿       rxm124330
supergroup^AÃ¿^@//user/cloud/rxm124330/Crime/fullblock1/_SUCCESS^@^C^@^@^AD>ÃÃ¸1^@^@^AD>ÃÃ¸1^@^@^@^@^H^@^@^@^@^@^@       rxm124330
supergroup^AÂ¤^@,/user/cloud/rxm124330/Crime/fullblock1/_logs^@^@^@^@^AD>ÃÃ¯V^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿ rxm124330
supergroup^AÃ¿^@1/user/cloud/rxm124330/Crime/fullblock1/_temporary^@^@^@^@^AD>Ã^^Â§^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿   rxm1
supergroup^AÃ¿^@3/user/cloud/rxm124330/Crime/fullblock1/part-r-00000^@^C^@^@^AD>Ã<9f>Â¿^@^@^AD>Ã<9f>Â¿^@^@^@^@^H^@^@^@^@^@^A^K<94>ÃÂ´S<84>^OÃ¼^@^@^@^@^@^\ÃÃ^@^@
supergroup^AÂ¤^@3/user/cloud/rxm124330/Crime/fullblock1/part-r-00001^@^C^@^@^AD>ÃÂ <9e>^@^@^AD>ÃÂ <9e>^@^@^@^@^H^@^@^@^@^@^A^P<80>^QÃ=Ã^MÃ¦^@^@^@^@^@^\Ã-^@^@^@
supergroup^AÂ¤^@3/user/cloud/rxm124330/Crime/fullblock1/part-r-00002^@^C^@^@^AD>Ã<9d>^^^@^@^AD>Ã<9d>^^^@^@^@^@^H^@^@^@^@^@^AÃ°ÃBTJHTÃ¬^@^@^@^@^@^\ÃÃª^@^@^@^@
supergroup^AÂ¤^@3/user/cloud/rxm124330/Crime/fullblock1/part-r-00003^@^C^@^@^AD>ÃÂ¡<8a>^@^@^AD>ÃÂ¡<8a>^@^@^@^@^H^@^@^@^@^@^A^V<86>'Ã°^NÂ±.f^@^@^@^@^@^\Ã1^@^@^@
supergroup^AÂ¤^@3/user/cloud/rxm124330/Crime/fullblock1/part-r-00004^@^C^@^@^AD>ÃÂ¡Z^@^@^AD>ÃÂ¡Z^@^@^@^@^H^@^@^@^@^@^A<99>Ã§Â¾cÃ¹:Ã®;^@^@^@^@^@^\<92>^Q^@^@^@
supergroup^AÂ¤^@3/user/cloud/rxm124330/Crime/fullblock1/part-r-00005^@^C^@^@^AD>Ã<9e>;^@^@^AD>Ã<9e>;^@^@^@^@^H^@^@^@^@^@^A^?OÃ«Ãp<91>IÂ°^@^@^@^@^@^\Ã^Z^@^@^@
supergroup^AÂ¤^@3/user/cloud/rxm124330/Crime/fullblock1/part-r-00006^@^C^@^@^AD>ÃÃÃ°^@^@^AD>ÃÃÃ°^@^@^@^@^H^@^@^@^@^@^AK?Ã^AU^_^Â±^@^@^@^@^@^\<9d>^T^@^@^@^@^E<
supergroup^AÂ¤^@3/user/cloud/rxm124330/Crime/fullblock1/part-r-00007^@^C^@^@^AD>ÃÃ Ã^@^@^AD>ÃÃ Ã^@^@^@^@^H^@^@^@^@^@^AÃ½=^^Ã¿Ã¾<86>^C^G^@^@^@^@^@^\Ã;^@^@^@^@
supergroup^AÂ¤^@3/user/cloud/rxm124330/Crime/fullblock1/part-r-00008^@^C^@^@^AD>ÃÃ¦^V^@^@^AD>ÃÃ¦^V^@^@^@^@^H^@^@^@^@^@^AYÃ^C^VÃ°Â°R<9e>^@^@^@^@^@^\<8c>k^@^@^@
supergroup^AÂ¤^@3/user/cloud/rxm124330/Crime/fullblock1/part-r-00009^@^C^@^@^AD>ÃÃ®Â´^@^@^AD>ÃÃ®Â´^@^@^@^@^H^@^@^@^@^@^A<m8Ã¸Ã-Ã¿@Ã½^@^@^@^@^@^\Ã^[^@^@^@^@^E<
supergroup^AÂ¤^@4/user/cloud/rxm124330/Crime/fullblock1/_logs/history^@^@^@^@^AD>ÃÃ¯c^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿ rxm1
supergroup^AÃ¿^@i/user/cloud/rxm124330/Crime/fullblock1/_logs/history/job_201402160159_0041_1392622694161_rxm124330_crimer^@^C^@^@^AD>ÃÃ¯V^@^@^AD>ÃÃ¯V^@^@^@^@^H^@
supergroup^AÂ¤^@S/user/cloud/rxm124330/Crime/fullblock1/_logs/history/job_201402160159_0041_conf.xml^@^C^@^@^AD>ÃÃ¯c^@^@^AD>ÃÃ¯c^@^@^@^@^H^@^@^@^@^@^AÃ¸Â£<96>Â§
supergroup^AÃ¿^@W/user/cloud/rxm124330/Crime/fullblock1/_temporary/_attempt_201402160159_0041_r_000009_1^@^@^@^@^AD>Ã^^Â§^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@Ã¿Ã¿Ã¿Ã¿
supergroup^AÂ¤^@d/user/cloud/rxm124330/Crime/fullblock1/_temporary/_attempt_201402160159_0041_r_000009_1/part-r-00009^@^C^@^@^AD>Ã^^Â§^@^@^AD>Ã^^Â§^@^@^@^@^H^@^@
supergroup^AÂ¤^@0/user/cloud/rxm124330/Crime/fullblock32/_SUCCESS^@^C^@^@^AD>Âª^C*^@^@^AD>Âª^C*^@^@^@^@^H^@^@^@^@^@^@    rxm124330
supergroup^AÂ¤^@-/user/cloud/rxm124330/Crime/fullblock32/_logs^@^@^@^@^AD>Â§3Ã°^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿        rxn
supergroup^AÃ¿^@2/user/cloud/rxm124330/Crime/fullblock32/_temporary^@^@^@^@^AD>Âª#Â¼^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿Ã¿   rxn
supergroup^AÃ¿^@4/user/cloud/rxm124330/Crime/fullblock32/part-r-00000^@^C^@^@^AD>Â©Ã^1^@^@^AD>Â©Ã^1^@^@^@^@^H^@^@^@^@^@^A'<9c>#)ÃÃÃ<8d>^@^@^@^@^@^\ÃÃ^@^@^@^@
```

Figure 1 FS Image generated by my netid – rxm124330 containing data path , metadata . See fsimage.txt

HDFS will create files up to the size of the HDFS block size as well as a meta file that contains CRC32 checksums for that block. The *fsimage* file in namenode contains a serialized form of all the directory and file inodes in the filesystem.

3. LARGE VS SMALL INPUT FILES:

Two files are given to be analyzed-

1. Single Large file - merged_crime.csv
2. Multiple large files - *.csv

SINGLE FILE

Filesystem check (fsck) for INPUT:

```
rxm124330@master:/home/cloud/hadoop-
1.1.0/bin$ hadoop fsck
/user/cloud/full_input/single -files -
blocks -racks
```

```
FSCK started by rxm124330 from /192.168.1.107 for path
/user/cloud/full_input/single at Tue Feb 25 22:22:01 CST 2014
.Status: HEALTHY
 Total size:    2183794617 B
 Total dirs:    1
 Total files:   1
 Total blocks (validated):      17 (avg. block size 128458506 B)
 Minimally replicated blocks:   17 (100.0 %)
 Over-replicated blocks:        0 (0.0 %)
 Under-replicated blocks:       0 (0.0 %)
 Mis-replicated blocks:         0 (0.0 %)
 Default replication factor:    3
 Average block replication:     3.0
 Corrupt blocks:                0
 Missing replicas:              0 (0.0 %)
 Number of data-nodes:          3
 Number of racks:               1
```

Distribution of OUTPUT:

```
rxm124330@master:/home/cloud/hadoop-1.1.0/bin$ hadoop fsck
/user/cloud2014/rxm124330/Crime/test1/part-r-00000 -files -blocks -
racks
FSCK started by rxm124330 from /192.168.1.107 for path
/user/cloud2014/rxm124330/Crime/test1/part-r-00000 at Wed Feb
26 02:02:26 CST 2014
/user/cloud2014/rxm124330/Crime/test1/part-r-00000 1884632
bytes, 1 block(s):  OK
0. blk_-4987648222896431600_363750 len=1884632 repl=3
[/default-rack/192.168.1.109:50010, /default-
rack/192.168.1.110:50010, /default-rack/192.168.1.108:50010]


Status: HEALTHY
 Total size:    1884632 B
 Total dirs:    0
 Total files:   1
 Total blocks (validated):      1 (avg. block size 1884632 B)
 Minimally replicated blocks:   1 (100.0 %)
 Over-replicated blocks:        0 (0.0 %)
 Under-replicated blocks:       0 (0.0 %)
 Mis-replicated blocks:         0 (0.0 %)
 Default replication factor:    3
```

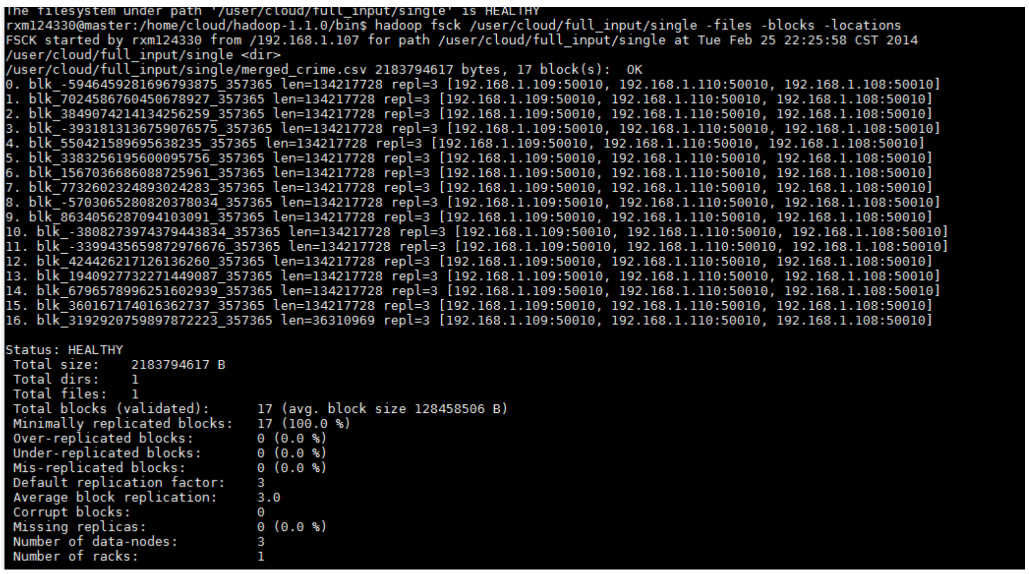
MULTIPLE FILES

Filesystem check (fsck) for INPUT:

```
 rxm124330@master:/home/cloud/hadoop-
1.1.0/bin$ hadoop fsck
/user/cloud/full_input/data/data -files -
blocks -racks
```

```
Status: HEALTHY
Total size:    2183910056 B
 Total dirs:    1
 Total files:   1341
 Total blocks (validated):      1341 (avg. block size 1628568 B)
 Minimally replicated blocks:   1341 (100.0 %)
 Over-replicated blocks:        0 (0.0 %)
 Under-replicated blocks:       0 (0.0 %)
 Mis-replicated blocks:         0 (0.0 %)
 Default replication factor:    3
 Average block replication:     3.0
 Corrupt blocks:                0
 Missing replicas:              0 (0.0 %)
 Number of data-nodes:          3
 Number of racks:               1
Replication = 3
```

Average block replication: 3.0 Corrupt blocks: 0 Missing replicas: 0 (0.0 %) Number of data-nodes: 3 Number of racks: 1 FSCK ended at Wed Feb 26 02:02:26 CST 2014 in 1 milliseconds	
1. Split size = BlockSize = 128MB, Number of splits = 2.03GB/128MB = 17 splits . Each split will have a Mapper assigned. 2. MapReduce works best when it can operate at the transfer rate of the disks in the cluster. As the splits are nearly equal to the block size, it can have max efficiency.	1. Number of splits :Multiple Files less than 128MB are assigned a mapper . As the files are less than 128MB , A mapper is assigned for each file. 2. Processing many small files increases the number of seeks that are needed to run a job. Also, storing large numbers of small files in HDFS is wasteful of the NameNode's memory. 3. Here **1341** blocks took 1344 mappers and the job time is tens or hundreds of times slower than the equivalent one with a single input file and 13 map tasks.



```
The filesystem under path '/user/cloud/full_input/single' is HEALTHY
rxm124330@master:/home/cloud/hadoop-1.1.0/bin$ hadoop fsck /user/cloud/full_input/single -files -blocks -locations
FSCK started by rxm124330 from /192.168.1.107 for path /user/cloud/full_input/single at Tue Feb 25 22:25:58 CST 2014
/user/cloud/full_input/single <dir>
/user/cloud/full_input/single/merged_crime.csv 2183794617 bytes, 17 block(s):  OK
0. blk_-5946459281696793875_357365 len=134217728 repl=3 [192.168.1.109:50010, 192.168.1.110:50010, 192.168.1.108:50010]
1. blk_7024586760450678927_357365 len=134217728 repl=3 [192.168.1.109:50010, 192.168.1.110:50010, 192.168.1.108:50010]
2. blk_3849074214134256259_357365 len=134217728 repl=3 [192.168.1.109:50010, 192.168.1.110:50010, 192.168.1.108:50010]
3. blk_-3931813136759076575_357365 len=134217728 repl=3 [192.168.1.109:50010, 192.168.1.110:50010, 192.168.1.108:50010]
4. blk_550421589695638235_357365 len=134217728 repl=3 [192.168.1.109:50010, 192.168.1.110:50010, 192.168.1.108:50010]
5. blk_3383256195600095756_357365 len=134217728 repl=3 [192.168.1.109:50010, 192.168.1.110:50010, 192.168.1.108:50010]
6. blk_1567036686088725961_357365 len=134217728 repl=3 [192.168.1.109:50010, 192.168.1.110:50010, 192.168.1.108:50010]
7. blk_7732602324893024283_357365 len=134217728 repl=3 [192.168.1.109:50010, 192.168.1.110:50010, 192.168.1.108:50010]
8. blk_-5703065280820378034_357365 len=134217728 repl=3 [192.168.1.109:50010, 192.168.1.110:50010, 192.168.1.108:50010]
9. blk_8634056287094103091_357365 len=134217728 repl=3 [192.168.1.109:50010, 192.168.1.110:50010, 192.168.1.108:50010]
10. blk_-3808273974379443834_357365 len=134217728 repl=3 [192.168.1.109:50010, 192.168.1.110:50010, 192.168.1.108:50010]
11. blk_-3399435659872976676_357365 len=134217728 repl=3 [192.168.1.109:50010, 192.168.1.110:50010, 192.168.1.108:50010]
12. blk_4244262171261362 60_357365 len=134217728 repl=3 [192.168.1.109:50010, 192.168.1.110:50010, 192.168.1.108:50010]
13. blk_1940927732271449087_357365 len=134217728 repl=3 [192.168.1.109:50010, 192.168.1.110:50010, 192.168.1.108:50010]
14. blk_6796578996251602939_357365 len=134217728 repl=3 [192.168.1.109:50010, 192.168.1.110:50010, 192.168.1.108:50010]
15. blk_3601671740163627 37_357365 len=134217728 repl=3 [192.168.1.109:50010, 192.168.1.110:50010, 192.168.1.108:50010]
16. blk_3192920759897872223_357365 len=36310969 repl=3 [192.168.1.109:50010, 192.168.1.110:50010, 192.168.1.108:50010]

Status: HEALTHY
 Total size:    2183794617 B
 Total dirs:    1
 Total files:   1
 Total blocks (validated):      17 (avg. block size 128458506 B)
 Minimally replicated blocks:   17 (100.0 %)
 Over-replicated blocks:        0 (0.0 %)
 Under-replicated blocks:       0 (0.0 %)
 Mis-replicated blocks:         0 (0.0 %)
 Default replication factor:    3
 Average block replication:     3.0
 Corrupt blocks:                0
 Missing replicas:              0 (0.0 %)
 Number of data-nodes:          3
 Number of racks:               1
```

Figure 2 Data distribution of a Single Large file

Virtual Nodes:

If virtual nodes are enabled on hadoop node, it will lower performance of small Hadoop jobs by raising the number of mappers to at least the number of virtual nodes. Those mappers might have too little data to process and the server would spend most of the time managing those tasks instead of doing useful work.

On a decent hardware, a job with *no data* and 256 vnodes may take about 5-10 minutes, contrary to the same job requiring only about 20-40 seconds when configured without vnodes.

4. NUMBER OF MAP TASKS:

	Map tasks=17	**Map tasks=2087**
SLOTS_MILLIS_MAPS (Total time spent by all maps in occupied slots (ms))	78822	1152601
Spilled Records	7887751	4569682
CPU Time	123710	698210
Physical memory	5398331392	488735756288
mapSlotSeconds	78	1152

```
mapred.min.split.size = 0
```

`mapred.max.split.size` = Long.MAX_VALUE

```
minimumSize < blockSize < maximumSize
```

The number of map tasks depends on the number of input splits. Input split size can be altered using ***setMaxInputSplitSize*** parameter.

Default split size = Block size = 128MB, Number of map tasks for Single File with 2.03 GB = 2.03GB/128MB=17Mappers. If splitsize = 1MB, then 2.03GB/1MB =2087 Mappers are scheduled.

```
FileInputFormat.setMaxInputSplitSize(job, 1048576); //set maximum split size
to  1048576=1MB
```

The number of mappers effects the performance param's like **Job time, Number of seeks and namenode memory.**

5. NUMBER OF REDUCER TASKS:

	Reducers=2	**Reducers=10**
SLOTS_MILLIS_REDUCES (Total time spent by all reducers in occupied slots (ms))	29418	114566
committed heap usage (bytes)=	3958898688	5569511424
CPU time spent (ms)	108110	123710
Physical memory (bytes) snapshot	4331442176	5398331392
Virtual memory (bytes) snapshot	11229290496	16529448960
reduceSlotsSeconds	29	114

```
//Set number of reducer tasks
//job.setNumReduceTasks(2);
```

The number of reducer tasks can be set explicitly for the current job in the java code using parameter- `setNumReduceTasks` . The **optimal** number of reducers is related to the total number of available reducer slots in the cluster. The total number of slots is found by multiplying the number of nodes in the cluster and the number of slots per node (which is determined by the value of the `mapred.tasktracker.reduce.tasks.maximum =2` property, described in Environment Settings.

One setting is to have slightly fewer reducers than total slots, which gives one wave of reduce tasks (and tolerates a few failures, without extending job execution time). If the reduce tasks are very big, it makes sense to have a larger number of reducers (resulting in two waves, for example) so that the tasks are more fine-grained and failure doesn't affect job execution time significantly. As the reducers are more , the Total time spent by all reducers are increased considerably .

The **right number** of reduces seems to be 0.95 or 1.75 multiplied by (<*no. of nodes*> * mapred.tasktracker.reduce.tasks.maximum).

6. Distribution of Tasks:

```
rxm124330@master:/home/cloud/hadoop_logs$ pwd
/home/cloud/hadoop_logs
Vi Hadoop-hadoop-jobtracker-master.log
```

Map and Reduce tasks are distributed and scheduled by **Jobtracker** . The jobtracker combines the updates from tasktracker to produce a **global view** of the status of all the jobs being run and their constituent tasks.

The log file Hadoop-hadoop-jobtracker-master.log records all such scheduling of job setup, map tasks, reduce tasks, job cleanup etc.
Job tracker log for a MAPRED JOB job_201402160159_0015 for a Single large File (Some lines omitted)

```
2014-02-16 22:52:19,923 INFO org.apache.hadoop.mapred.JobTracker: Adding task (JOB_SETUP)
'attempt_201402160159_0015_m_000018_0' to tip task_201402160159_0015_m_000018, for tracker
'tracker_slave3:localhost/127.0.0.1:58963'
2014-02-16 22:52:21,132 INFO org.apache.hadoop.mapred.JobTracker: Adding task (MAP)
'attempt_201402160159_0015_m_000000_0' to tip task_201402160159_0015_m_000000, for tracker
'tracker_slave3:localhost/127.0.0.1:58963'
----------
2014-02-16 22:52:27,466 INFO org.apache.hadoop.mapred.JobTracker: Adding task (REDUCE)
'attempt_201402160159_0015_r_000000_0' to tip task_201402160159_0015_r_000000, for tracker
'tracker_slave3:localhost/127.0.0.1:58963'
----------
2014-02-16 22:52:49,481 INFO org.apache.hadoop.mapred.JobTracker: Adding task (JOB_CLEANUP)
'attempt_201402160159_0015_m_000017_0' to tip task_201402160159_0015_m_000017, for tracker
'tracker_slave3:localhost/127.0.0.1:58963'
```

These map/reduce tasks are run by **Task tracker .** The Tasktracker also sends progress to jobtracker and maintains record of overall progress.

Task Assignment and Execution:
A Tasktracker for the given HDFS is set to run two map tasks and two reduce tasks simultaneously.

```
<property><name>mapred.tasktracker.map.tasks.maximum</name><value>2</value></property>
<property><name>mapred.tasktracker.reduce.tasks.maximum</name><value>2</value></property>
```

A **scheduler** fills empty map task slots before reduce task slots. So if the tasktracker has at least one empty map task slot, the jobtracker will select a map task; otherwise, it will select a reduce task.
-To choose a **reduce task**, the jobtracker simply takes the next in its list of yet-to-be-run reduce tasks, since there are no data locality considerations.

-For a **map task**, it takes into account the tasktracker's network location and picks a task whose input split is as close as possible to the tasktracker. In this log, the job tracker chooses **data-local** which is close in the network.

```
2014-02-16 22:52:21,132 INFO org.apache.hadoop.mapred.JobInProgress: Choosing data-local task
task_201402160159_0015_m_000000
```

7. The performance Metrics:

Different phases in map-reduce job include-

1 –Input for Map phase

2 –**Computation for Map phase**

3 – Partition and sort for Map phase

4 –Output for Map phase

5 – Copying Map output

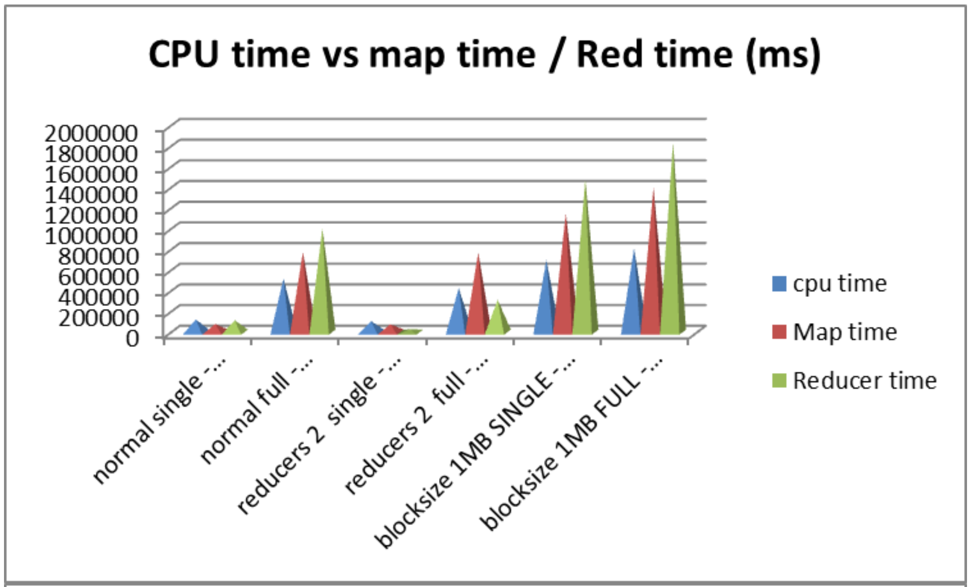
6 – **Shuffling and sorting during the Reduce phase**

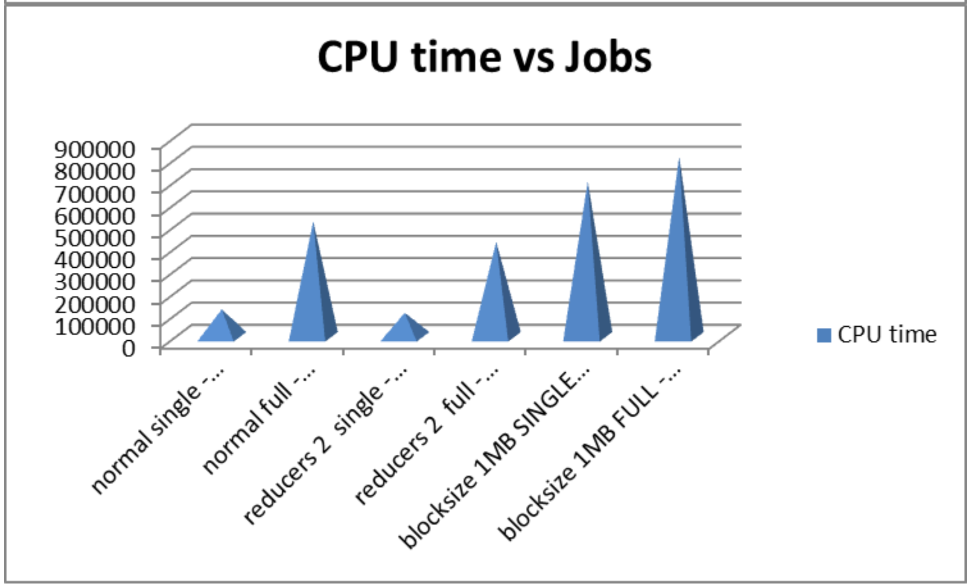
7 –**Computation for the Reduce phase**

8 –Output of the Reduce phase

Performance metrics are considered for map, reduce, Shuffling&Sorting phases:

1. CPU time: (map phase+ Shuffling&sort phase + reducer phase)

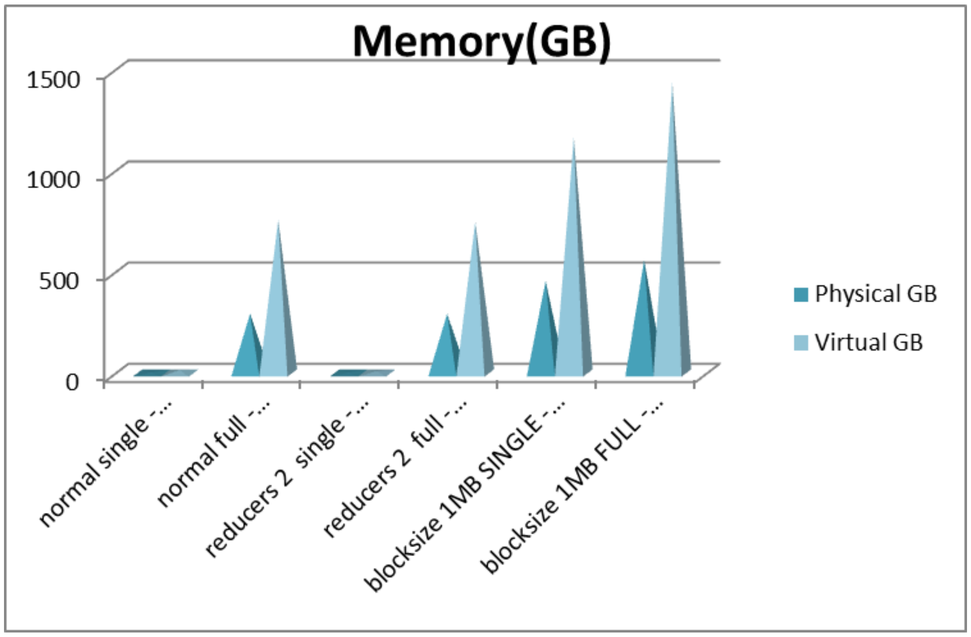




OBSERVATIONS:

- When the block size is reduced to 1MB, the CPU time is more than other cases. So as block size decreases, the time spent is more.
- Total Reducer time is always more than Total map time.
- As the number of mappers for small files are lower, the time taken is also less .

2. Memory usage:



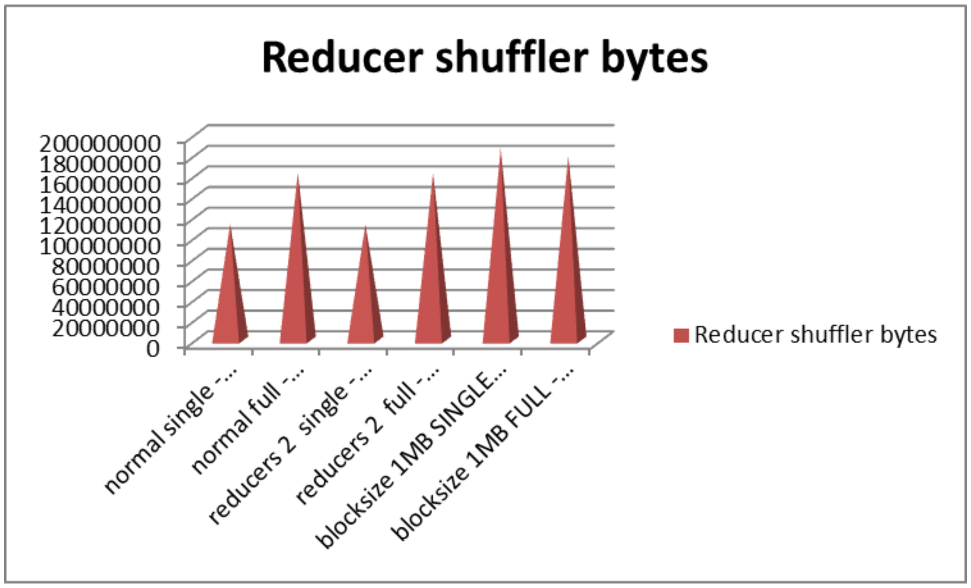


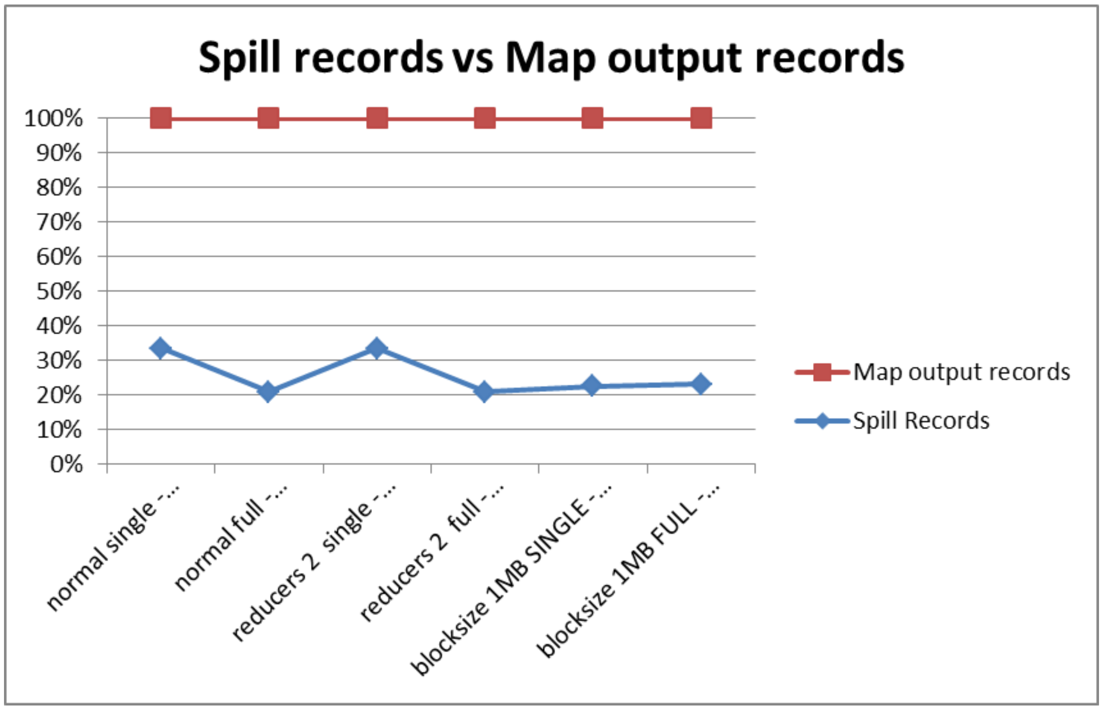
Figure 3 Amount of data shuffled

Observations:

As the number of mappers are increased the memory snapshot increases considerably.
The number of map/reduce tasks are to be carefully chosen, as too many task slots can cause memory exceptions and repeated job failures. Adjust sort buffer size for each map task such that sufficient memory is available for the operating system.

Map tasks's output are stored in a local memory, but not distributed. Each map task has a circular memory buffer that it writes the output to. The buffer is 100 MB by default, a size that can be tuned by changing the io.sort.mb property. When the contents of the buffer reaches a certain threshold size (io.sort.spill.percent, which has the default 0.80, or 80%), a background thread will start to *spill* the contents to disk. Map outputs will continue to be written to the buffer while the spill takes place, but if the buffer fills up during this time, the map will block until the spill is complete.

4. Disk IO:



Observations:

- Spill becomes less, when block size is reduced.
- It is higher for Single large file than large number of small files. So in terms of splills , single files create stress than large number of small files.

A Map task can spill (meaning, write data to the file system or disk) a large amount of data during the internal **sorting** process. Therefore, your goal for optimization should be to ensure that records are spilled only once. Multiple data spills generate a lot of **I/O stress** and slow down overall job performance. To reduce the amount of data spilled during the intermediate Map phase, the following param's are to be changed.

io.sort.mb — Size, in megabytes, of the memory buffer to use while sorting map output.

io.sort.record.percent — Proportion of the memory buffer defined in io.sort.mb that is reserved for storing record boundaries of the Map outputs. The remaining space is used for the Map output records themselves.

io.sort.spill.percent — value that represents the soft limit in either the buffer or record collection buffers. Once this threshold is reached, a thread will begin to spill the contents to disk. Applies to both the map output memory buffer and the record boundaries index.

On the **reduce side**, the best performance is obtained when the intermediate data can reside entirely in memory. This does not happen by default, since for the general case all the memory is reserved for the reduce function. But if your reduce function has light memory requirements, setting the `mapred.inmem.merge.threshold` to `0` and the `mapred. job.reduce.input.buffer.percent` to `1.0` may bring a performance boost.

5. Network Traffic:

Since Hadoop operates on large chunk sizes, significant network traffic during shuffle phase and replication is probable. Block compression offers an attractive solution to reduce network traffic and disk IO without burdening the CPU.

The network traffic is short-lived flows coming from several different servers all at the same time to reducers. These flows may cause temporary congestion at the receiving node either in form of high utilization of port or throughput reduction.

Reading Data:

Network traffic is limited during the importing from HDFS. The reading time depends of several factors. First the reading is performed serially; secondly the data size to be read and finally the disk IO capacity of the compute node.

Writing Data:

Hadoop by default has replication set to 3: that is, any data imported into Hadoop has one copy stored locally on the receiving data node and two copies sent to two other nodes in the cluster. Therefore, a simple import of data into a Hadoop cluster also results in bursts of traffic

8. Handling corrupt and bad records:

Corrupt records are introduced in the data by deleting some fields , records and changing the order of the fields . The given data also contains some inbuilt errors in the Easting, Northing , Crime_type fields.

1. Handling Different formats:

Detect the bad record and ignore it, or abort the job by throwing an exception

```
	if((fields[4].length() == 6)&&(fields[4].matches("[0-
9]+")==true)&&(fields[5].length()==6)&&(fields[5].matches("[0-9]+")==true))
```

Checking number of characters and check whether the Easting, Northing values are indeed numbers but not chars.

2. Handling Missing Fields *– USING try, catch. All errors are catched and those fields are set as 'Missing data'*

3. *Count* the total number of bad records in the job using counters to see how the problem is widespread.

4. Hadoop's optional ***skipping mode*** for automatically skipping bad records caused by bugs in third-party libraries. When skipping mode is enabled, tasks report the records being processed back to the tasktracker. When the task fails, the tasktracker retries the task, skipping the records that caused the failure. Because of the extra network traffic and bookkeeping to maintain the failed record ranges, skipping mode is turned on for a task only after it has failed twice.Bad records that have been detected by Hadoop are saved as sequence files in the job's output directory under the *_logs/skip* subdirectory.

Large datasets are messy. They often have corrupt records. They often have records that are in a **different format**. They often have **missing fields**. These can be identified, ignored and rectified in the Code. Depending on the analysis being performed, if only a small percentage of records are affected, then skipping them may not significantly affect the result. if a task trips up when it encounters a bad record—by throwing a runtime exception—then the task fails. Failing tasks are **retried**, but if a task fails four times, the whole job is marked as **failed**. If it is the data that is causing the task to throw an exception, rerunning the task won't help, since it will fail in exactly the same way each time.

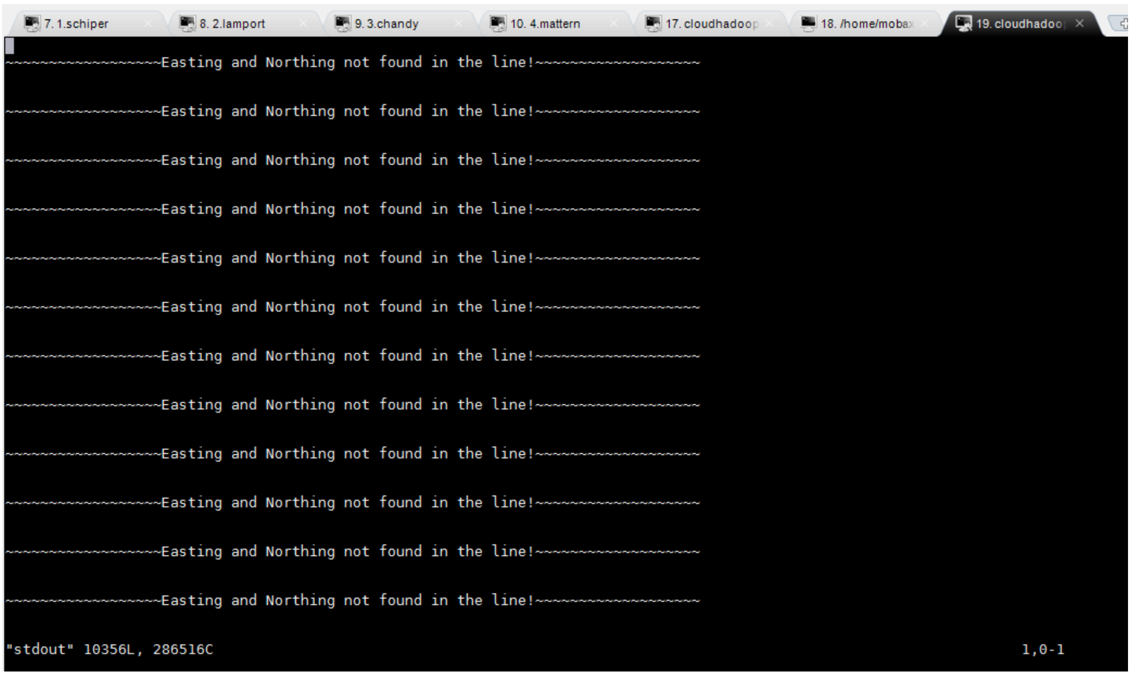
By setting `mapred.linerecordreader.maxlength` to a value in bytes that fits in memory (and is comfortably greater than the length of lines in your input data), the record reader will skip the (long) corrupt lines without the task failing.

Errors are also logged on the tasktracker:

```
/var/log/hadoop
              /hadoop-* => daemon logs
              /userlogs
                      /attempt_*
                               /stderr => standard error logs
```

```
/stdout => standard out logs
```

cloud@master:~/hadoop_logs/userlogs/job_201402150217_0012/attempt_201402150217_0012_m_000004_0$vi stdout



```
~~~~~~~~~~~~~~~~~~Easting and Northing not found in the line!~~~~~~~~~~~~~~~~~~~

~~~~~~~~~~~~~~~~~~Easting and Northing not found in the line!~~~~~~~~~~~~~~~~~~~

~~~~~~~~~~~~~~~~~~Easting and Northing not found in the line!~~~~~~~~~~~~~~~~~~~

~~~~~~~~~~~~~~~~~~Easting and Northing not found in the line!~~~~~~~~~~~~~~~~~~~

~~~~~~~~~~~~~~~~~~Easting and Northing not found in the line!~~~~~~~~~~~~~~~~~~~

~~~~~~~~~~~~~~~~~~Easting and Northing not found in the line!~~~~~~~~~~~~~~~~~~~

~~~~~~~~~~~~~~~~~~Easting and Northing not found in the line!~~~~~~~~~~~~~~~~~~~

~~~~~~~~~~~~~~~~~~Easting and Northing not found in the line!~~~~~~~~~~~~~~~~~~~

~~~~~~~~~~~~~~~~~~Easting and Northing not found in the line!~~~~~~~~~~~~~~~~~~~

~~~~~~~~~~~~~~~~~~Easting and Northing not found in the line!~~~~~~~~~~~~~~~~~~~

~~~~~~~~~~~~~~~~~~Easting and Northing not found in the line!~~~~~~~~~~~~~~~~~~~

~~~~~~~~~~~~~~~~~~Easting and Northing not found in the line!~~~~~~~~~~~~~~~~~~~

"stdout" 10356L, 286516C                                                                                   1,0-1
```

Figure 4 Standard out log for a task attempt_201402150217_0012_m_000004_0

9. References:

1. **Hadoop definitive guide, 3rd edition.**
2. **http://blog.cloudera.com/blog/2009/09/apache-hadoop-log-files-where-to-find-them-in-cdh-and-what-info-they-contain/**