INTERNATIONAL JOURNAL OF RESEARCH IN COMPUTER APPLICATIONS AND ROBOTICS
ISSN 2320-7345

DATA EXTRACTION - LINUX COMMANDS FOR DATA DRILLING WITH PERFORMANCE MAP

Manpreet Singh Sandhu[1], Dr. Saurabh Srivastava[2]

[1]Corporate Trainer & Consultant, M.C.A. & Pursuing PhD from Mewar University
[2]Deparment of Computer Application, BU Jhansi (India)
Author Correspondence: 41, Canara Apartments, Sector 13, Rohini, Delhi 110 085, 09811218512, manu_ya@yahoo.com

Abstract: -This paper is summary of linux commands capability to extract data from file formats like csv or uniform logical structured Modelling. This paper is a genuine attempt to prove the capability of GNU utilities with the help of experimenting and comparing with equivalent structured query command along-with time taken to perform the same action using linux / gnu command. The idea behind this is not to replace the database or data-mining tools but to perform the same using gnu/linux based systems as it is free open-source widely ***been*** available and can be used at small to medium volume load.

Keywords: GNU Linux commands, Linux commands performance, response time – Linux text *extraction*, Linux apart from OS, GNU Power, data drilling using linux.

I INTRODUCTION

In one of the published paper(s), about power of GNU tools / utilities under the heading "G'NOO – THE POWER BEHIND LINUX (GNU/LINUX)"[1], and also, another paper with title "STUDY LINUX POWER – BY DESIGN AND IMPLEMENTATION OF COMMANDS AS QUERIES FOR READING DATA" [2], mentioning about the use of linux based commands to extract designed data which work more like SQL – commands for extraction or reading of data, set the base for this experiment results. In this paper the intention is to experiment and share results of linux commands / GNU tools which are equivalently been used for extraction of data along-with time taken.

Data drilling, data processing for research and analysis is something which is talk of the town these days using some very expensive hardware and software, but all these times we don't have this much large data and budget to do the same.

This is a gentle attempt to go on the foot prints of doing something on the path of Big Data by processing comparatively less data without using anything like Hadoop or HBASE.

II LINUX FOR WHOM

Linux is hugely accepted not only by educationalists but by industry as well. Many corporates use linux or linux based systems as servers and host their respective server-based applications on it. Starting from finance, automobile, and even telecommunication world also use it as the base – android is all based on linux kernel[3], of course minus other utilities. Apart from software solutions, database world is also hugely rely and work on linux or linux based systems – a very good example of this is "Big Data – Hadoop". To work on framework like Hadoop one need to have linux or linux based systems [4].

III QUESTION OF HOW?

The big question here is that, is it possible to use linux or linux based commands / utilities for filter and extract data, data sets. If the answer is yes then how and what about the performance of the commands which tend or pose to act like sql commands equivalent.

IV UNDERSTAND BIT OF BIG DATA JARGON

If don't talk about HBASE of Hadoop but Hive based system of Hadoop then it is clear that it works on symmetric and non-symmetric based data-sets and after filter or extract the desired data pass on to next phase for reducing and compilation or analysing.[4]

The Apache™ Hadoop® is an open-source software with reliable, scalable, distributed computing.[4]. In "Big Data Hadoop", the Apache Hadoop is software library which in turn is framework that facilitates the distributed processing for large data spanned across various computer clusters with the help of simple programming models. The purpose of this design is to scale up from single servers to multiple machines for computation and storage at local level. The library is designed to identify & detect with the purpose to handle failures at the application layer, providing highly-available service on top of a computer-clusters, each of which may be prone to failures [4]

For the purpose of distributed file system and faster access Hadoop have **HDFS™** and for data summarization & ad hoc querying[5] Hive™. Hive also provides way to query data just like SQL like language called HiveQL[5]

Hadoop is a batch processing system. Hive is mainly targeted to provide acceptable latency for interactive data browsing, queries over small data sets or test queries.[6] and is not designed for online transaction processing like row level updates.

Let us illustrate the capabilities of the QL language with the help of some examples.[6]

Simple Query

For all the active users, one can use the query of the following form:

```
INSERT OVERWRITE TABLE user_active
SELECT user.*
FROM user
WHERE user.active = 1;
```

Note that unlike SQL, we always insert the results into a table. We will illustrate later how the user can inspect these results and even dump them to a local file. You can also run the following query on Hive CLI:

```
SELECT user.*
FROM user
WHERE user.active = 1;
```

This will be internally rewritten to some temporary file and displayed to the Hive client side.

Joins

In order to get a demographic breakdown (by gender) of page_view of 2015-03-03 one would need to join the page_view table and the user table on the userid column. This can be accomplished with a join as shown in the following query:

```
INSERT OVERWRITE TABLE pv_users
SELECT pv.*, u.gender, u.age
FROM user u JOIN page_view pv ON (pv.userid = u.id)
WHERE pv.date = '2015-03-03';
```

In order to do outer joins the user can qualify the join with LEFT OUTER, RIGHT OUTER or FULL OUTER keywords in order to indicate the kind of outer join (left preserved, right preserved or both sides preserved). For example, in order to do a full outer join in the query above, the corresponding syntax would look like the following query:

```
INSERT OVERWRITE TABLE pv_users
SELECT pv.*, u.gender, u.age
FROM user u FULL OUTER JOIN page_view pv ON (pv.userid = u.id)
WHERE pv.date = '2015-03-03';
```

In order check the existence of a key in another table, the user can use LEFT SEMI JOIN as illustrated by the following example.

```
INSERT OVERWRITE TABLE pv_users
SELECT u.*
FROM user u LEFT SEMI JOIN page_view pv ON (pv.userid = u.id)
WHERE pv.date = '2015-03-03';
```

In order to join more than one tables, the user can use the following syntax:

```
INSERT OVERWRITE TABLE pv_friends
SELECT pv.*, u.gender, u.age, f.friends
FROM page_view pv JOIN user u ON (pv.userid = u.id) JOIN friend_list f ON (u.id = f.uid)
WHERE pv.date = '2015-03-03';
```

Aggregations

In order to count the number of distinct users by gender one could write the following query:

```
INSERT OVERWRITE TABLE pv_gender_sum
SELECT pv_users.gender, count (DISTINCT pv_users.userid)
FROM pv_users
GROUP BY pv_users.gender;
```

Multiple aggregations can be done at the same time, however, no two aggregations can have different DISTINCT columns .e.g while the following is possible

```
INSERT OVERWRITE TABLE pv_gender_agg
SELECT pv_users.gender, count(DISTINCT pv_users.userid), count(*), sum(DISTINCT pv_users.userid)
FROM pv_users
GROUP BY pv_users.gender;
```

Union All

The language also supports union all, e.g. if we suppose there are two different tables that track which user has published a video and which user has published a comment, the following query joins the results of a union all with the user table to create a single annotated stream for all the video publishing and comment publishing events:

```
INSERT OVERWRITE TABLE actions_users
SELECT u.id, actions.date
FROM (
   SELECT av.uid AS uid
   FROM action_video av
   WHERE av.date = '2015-01-03'

   UNION ALL

   SELECT ac.uid AS uid
   FROM action_comment ac
   WHERE ac.date = '2015-03-03'
   ) actions JOIN users u ON(u.id = actions.uid);
```

- File / data-set (like tablename) F_DS
- Columns of F_DS COLS

V LINUX COMMANDS ONLY - FOR EXTRACTION OF DATA

If without using any other technology or product we can pull our desired data from repository, then what?

Yes – is the answer. Linux commands (GNU Tools) are well capable to perform this task effectively. There are many tools bundled and available in linux (GNU tools) which help to perform the above said task easily.

The table below demonstrate some operation which is generally completed using SQL queries, but now can be demonstrated using equivalent linux based GNU command with time taken.

Sno.	Activity / Operation	Sql Command	Linux Command / utility solution	Linux Command Timing
1	Selection of all column(s) data-set	Select * from dataset	cat dataset	with 500+ recs • Flush on screen (1 sec) • Flush in file (<1 sec) with 32000+ recs • Flush on screen (17 sec) • Flush in file (<1 sec)
2	Select few selected columns from dataset	Select col1, col3 from dataset	cut –d “,” –f 1,3 dataset	with 500+ recs • Flush on screen (1 sec) • Flush in file (<1 sec) with 32000+ recs • Flush on screen (11 sec) • Flush in file (<1 sec)
3	Filtering few rows data on matching	Select column from dataset where	grep –i “condition” dataset (anywhere in row)	with 500+ recs • Flush on screen (1 sec)

	condition (assuming one text column data)	col=condition	for column based condition use awk –F"," '{ if($3=<condition> {print $4, $5} }' dataset	• Flush in file (<1 sec) with 32000+ recs • Flush on screen (3 sec) • Flush in file (1 sec)
4	Filtering few rows data on matching condition with particular column data	Select * from dataset where col3=condition	awk –F"," '{ if($3=<condition> {print $4, $5} }' dataset	with 500+ recs • Flush on screen (1 sec) • Flush in file (<1 sec) with 32000+ recs • Flush on screen (1 sec) • Flush in file (1 sec)
5	Counting rows in dataset	Select count(*) from dataset	wc –l dataset	with 500+ recs • Flush on screen (1 sec) • Flush in file (<1 sec) with 32000+ recs • Flush on screen (1 sec) • Flush in file (1 sec)
6	Condition based counting of rows in dataset	Select count(*) from dataset where col=condition	grep –i "condition" dataset | wc –l or awk –F"," '{…}' dataset | wc –l	with 500+ recs • Flush on screen (1 sec) • Flush in file (<1 sec) with 32000+ recs • Flush on screen (2 sec) • Flush in file (1 sec)
7	Changing the sequence of cols in dataset result	Select cols2, col3,col1 from dataset	awk –F"," '{print $2, print $3, print $1}' dataset	with 500+ recs • Flush on screen (2 sec) • Flush in file (<1 sec) with 32000+ recs • Flush on screen (10 sec) • Flush in file (<1 sec)
8	Using concept of calculated columns or aggregated columns	Select sal, comm, sal+comm from dataset	awk –F"," '{print $4, print $5, print $4+$5}' dataset	with 500+ recs • Flush on screen (2 sec) • Flush in file (<1 sec) with 32000+ recs • Flush on screen (6 sec) • Flush in file (1 sec)
9	Using sum aggregate funciton for addition aggregation results	Select sum(sal) from dataset	awk '{ sum+=$1} END {print sum}' dataset	with 500+ recs • Flush on screen (<1 sec) • Flush in file (<1 sec) with 32000+ recs • Flush on screen (<1 sec) • Flush in file (<1 sec)
10	Data display in some order	Select col1, col2 from dataset order by col1	sort –k2 dataset	with 500+ recs • Flush on screen (1 sec) • Flush in file (<1 sec) with 32000+ recs • Flush on screen (19 sec) • Flush in file (1 sec)
11	Data display in high to low numeric order	Select col1, col2 from dataset order by col1 desc	sort –k2 –r –n dataset	with 500+ recs • Flush on screen (1 sec) • Flush in file (<1 sec) with 32000+ recs • Flush on screen (30 sec) • Flush in file (1 sec)

12	Displaying non-duplicate records from dataset	Select distinct col1 from dataset	awk –F”,” ‘{print $2}’ dataset |sort –u	with 500+ recs • Flush on screen (<1 sec) • Flush in file (<1 sec) with 32000+ recs • Flush on screen (2 sec) • Flush in file (<1 sec)
13	Using wildcard patterns in search for extraction of data	Select col1 from dataset where col1 like ‘A%’	grep ‘A.*’ dataset or awk –F”,” ‘$1~”^A.*” {print $1}’ dataset	with 500+ recs • Flush on screen (1 sec) • Flush in file (<1 sec) with 32000+ recs • Flush on screen (3 sec) • Flush in file (<1 sec)
14	Grouping data from data set for summarization using sum function Or for count occurance using count(*)	Select col2, sum(col2) from dataset group by col1 Or Select col2, count(col2) from dataset group by col1	awk '{arr[$2]+=$2} END {for (i in arr) {printf(“%d,\t%d\n”, i,arr[i])}}' dataset Or awk '{arr[$2]++} END {for (i in arr) {printf(“%d,\t%d\n”, i,arr[i])}}' dataset [7] http://www.gnu.org/software/datamash/alternatives/	with 500+ recs • Flush on screen (<1 sec) • Flush in file (<1 sec) with 32000+ recs • Flush on screen (<1 sec) • Flush in file (<1 sec)
15	Using filter after group by on dataset result	Select col1, sum(col2) from dataset group by col1 having sum(col2)>100	awk '{arr[$1]+=$2} END {for (i in arr) {print i,arr[i]}}' dataset | awk ‘$2>100{ print $1, $2}’	with 500+ recs • Flush on screen (1 sec) • Flush in file (<1 sec) with 32000+ recs • Flush on screen (1 sec) • Flush in file (<1 sec)
16	Function support and use	Select firstname, len(firstname) from dataset	awk –F”,” ‘{print length($1), $1}’ dataset	with 500+ recs • Flush on screen (2 sec) • Flush in file (<1 sec) with 32000+ recs • Flush on screen (9 sec) • Flush in file (<1 sec)
17		Select substr(firstname, 1,3) from dataset	awk –F”,” ‘{print substr($1,1,3), $1}’ dataset	with 500+ recs • Flush on screen (2 sec) • Flush in file (<1 sec) with 32000+ recs • Flush on screen (8 sec) • Flush in file (<1 sec)
18		Select rand() from dual	awk –F”,” ‘{print rand(), $0}’ dataset	with 500+ recs • Flush on screen (2 sec) • Flush in file (<1 sec) with 32000+ recs • Flush on screen (9 sec) • Flush in file (<1 sec)
19	Extraction of row on the basis of rownum	Select * from dataset where rownum<=10	awk –F”,” ‘NR<=10{print $0}’ dataset	with 500+ recs • Flush on screen (2 sec) • Flush in file (<1 sec) with 32000+ recs • Flush on screen (4 sec) • Flush in file (<1 sec)
20	Extraction of	Select * from	awk –F”,” ‘NR<=10 && NR>=5	with 500+ recs

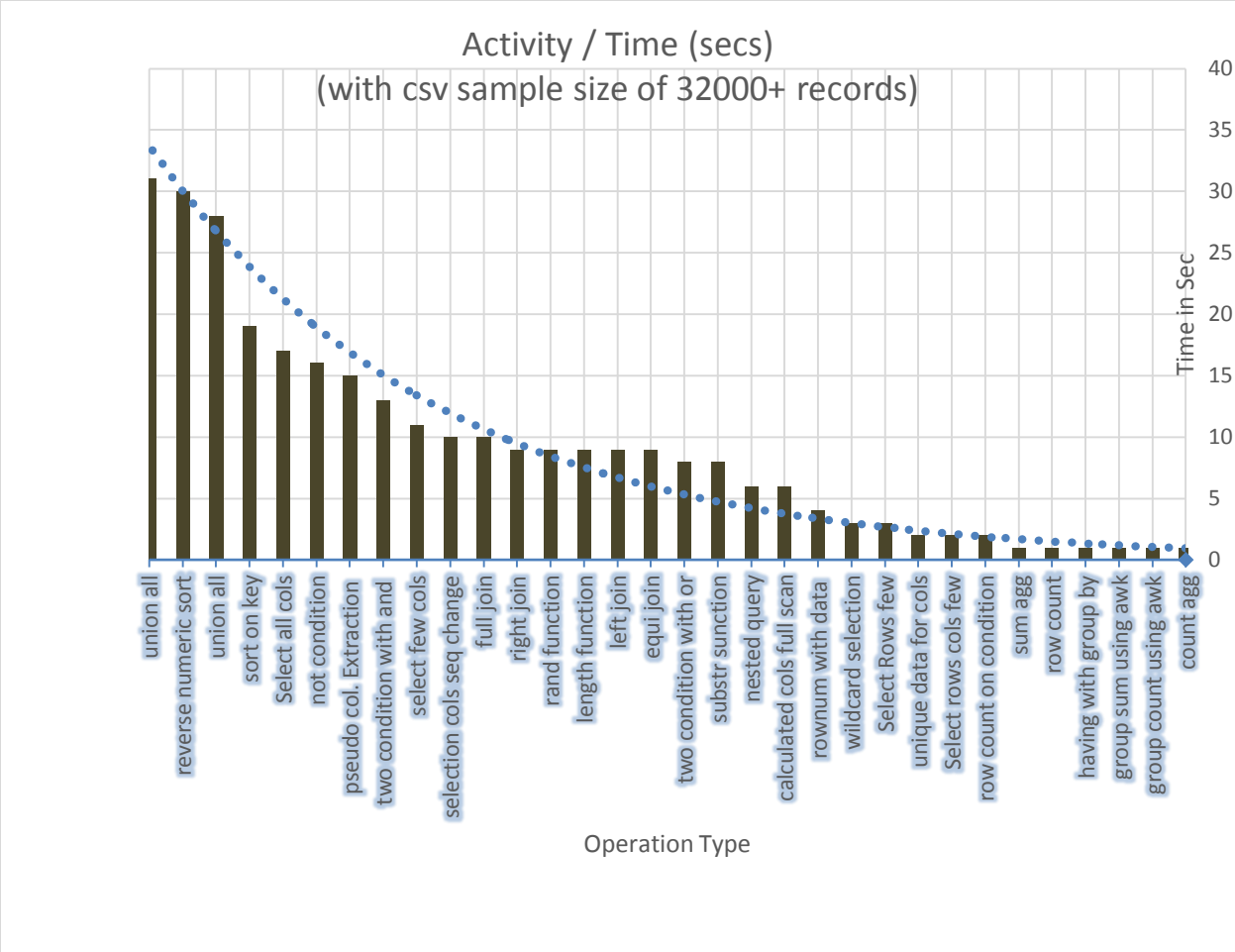
	data on the basis or two conditions	dataset where col1 = 10 and col2 =20	{print $0}' dataset	• Flush on screen (1 sec) • Flush in file (<1 sec) with 32000+ recs • Flush on screen (13 sec) • Flush in file (<1 sec)
21	Extraction of data on either of the condition true.	Select * from dataset where col1 = 10 or col2 = 20	awk –F"," '$1>=100 || NR<=10 {print $0}' dataset	with 500+ recs • Flush on screen (2 sec) • Flush in file (<1 sec) with 32000+ recs • Flush on screen (8 sec) • Flush in file (<1 sec)
22	Extraction of data when something is not equal to value	Select * from dataset where not (col1 = 10) Or Select * from dataset where col1 != 10	awk –F"," '$1!~10 {print $0}' dataset	with 500+ recs • Flush on screen (2 sec) • Flush in file (<1 sec) with 32000+ recs • Flush on screen (16 sec) • Flush in file (<1 sec)
23	Extract of data on the basis of output of nested query or sub-query	Select * from dataset where col1 = (select col1 from dataset where col2=10) and col1=100	awk –F"," '$2=10{ print $1}' | awk –F"," '$1=100{print $0}' or ls –l | awk –F " " '$2=1{print $9}'	with 500+ recs • Flush on screen (2 sec) • Flush in file (<1 sec) with 32000+ recs • Flush on screen (6 sec) • Flush in file (<1 sec)
24	Combining data one after the other result in rows Or Union all	Select * from dataset1 union all select * from dataset2	cat dataset1 dataset 2 (assuming equal and seq. of col in both dataset are accordingly)	with 32500+ recs • Flush on screen (19 sec) • Flush in file (<1 sec) with 64000+ recs • Flush on screen (31 sec) • Flush in file (<1 sec)
25	Combining data one after the other result in rows Or Union Removing duplicate	Select * from dataset1 union select * from dataset2	cat dataset1 dataset 2 | sort –k1 | uniq (assuming equal and seq. of col in both dataset are accordingly)	with 64000+ recs • Flush on screen (28 sec) • Flush in file (3 sec)
26	Displaying data using pseudo column rownum	Select rownum, * from dataset	cat –n dataset	with 32000+ recs • Flush on screen (15 sec) • Flush in file (1 sec)
27	Extarction of data from multiple source Equi join	Select col1, col2 from dataset1 as ds1 join dataset2 as ds2 where ds1.col3=ds2.col3	Join –t, -1 3 -2 3 dataset1 dataset2	with 32000+ and 500 recs • Flush on screen (9 sec) • Flush in file (1 sec)
28	Implementation of left outer join	Select col1, col2 from dataset1 as ds1 left outer join dataset2 as ds2	Join –t, -1 3 -2 3 –a 1 dataset1 dataset2	with 500 & 32000+ recs • Flush on screen (9 sec) • Flush in file (1 sec)

		where ds1.col3 = ds2.col3		
29	Implementation of right outer join	Select col1, col2 from dataset1 as ds1 right outer join dataset2 as ds2 where ds1.col3 = ds2.col3	Join –t, -1 3 -2 3 –a 2 dataset1 dataset2	with 500 & 32000+ recs • Flush on screen (9 sec) • Flush in file (1 sec)
30	Implementation of full outer join	Select col1, col2 from dataset1 as ds1 full outer join dataset2 as ds2 where ds1.col3 = ds2.col3	Join –t, -1 3 -2 3 –a1 –a2 –e "NA" dataset1 dataset2	with 500 & 32000+ recs • Flush on screen (10 sec) • Flush in file (1 sec)

Operation / Time taken (Table).

Sno	Operation	Time (secs)
1	count agg	1
2	group count using awk	1
3	group sum using awk	1
4	having with group by	1
5	row count	1
6	sum agg	1
7	row count on condition	2
8	Select rows cols few	2
9	unique data for cols	2
10	Select Rows few	3
11	wildcard selection	3
12	rownum with data	4
13	calculated cols full scan	6
14	nested query	6
15	substr sunction	8
16	two condition with or	8
17	equi join	9
18	left join	9
19	length function	9
20	rand function	9
21	right join	9
22	full join	10
23	selection cols seq change	10
24	select few cols	11
25	two condition with and	13
26	pseudo col. Extraction	15
27	not condition	16
28	Select all cols	17
29	sort on key	19

30	union all	28
31	reverse numeric sort	30
32	union all	31



VIII CONCLUSION

It is very clear from the above that structured data sets can be used in variety of ways to analyze data and all new era companies are using one or the other way to extract data from moderate to huge level in terms of volume to do the analysis and prediction. Also, the attempt to extract data without using SQL or involving any kind of program, but using only Linux / GNU tools is not only possible & feasible but not hard on time taken as well. This kind of technique of best suitable for low to moderate level data extraction, with minimal level of extra learning curve and costing (as GNU tools / Linux is free).

That way we can always see linux / GNU commands as some sort of alternative for SQL like commands to extract data with minimal impact on performance. So, Linux can act as powerful tool and is not merely an Operating System.

REFERENCE

[1] "G'NOO – THE POWER BEHIND LINUX (GNU/LINUX)"
   a. [International Journal Of Advance Research In Science And Engineering http://www.ijarse.com IJARSE, Vol. No.2, Issue No.12, December, 2013 ISSN-2319-8354(E)],
[2] "STUDY LINUX POWER – BY DESIGN AND IMPLEMENTATION OF COMMANDS
   a. AS QUERIES FOR READING DATA"
   b. [International Journal Of Research In Computer Application & Management http://ijrcm.org.in/ VOL. 4 (2014), ISSUE NO. 01 (JANUARY) ISSN 2231-1009]
[3] http://www.linux.org/threads/the-linux-kernel-android.5459/
[4] http://hadoop.apache.org/
[5] http://hive.apache.org/
[6] https://cwiki.apache.org/confluence/display/Hive/Tutorial
[7] http://www.gnu.org/software/datamash/alternatives

A Brief Author Biography

MANPREET SINGH SANDHU

41, Canara Apartments, Sector 13, Rohini – 110 085, 9811218512, worked as consultant and corporate trainer for CMM Level 5 giants at international level.

DR. SAURABH SRIVASTAVA, 9415504462, working in BU, Jhansi