

# High Speed Network Implementation for Oracle SQL\*Loader Performance on Solaris

Husnu Sensoy  
VLDB Specialist  
Istanbul, Turkiye  
husnu.sensoy@gmail.com

**Abstract**—This article describes how to implement a high speed network between two servers within a LAN to achieve +1 TB/hour data loading rate by SQL\*Loader tool. The article includes end-to-end details of the implementation starting from the customization of TCP/IP stack at operating system level (Solaris 10), and going up to the modification of database server (Oracle 10g Release 2) parameters.

## I. INTRODUCTION

With an increasing demand, transferring large amount of data became an issue between our ETL and database servers. After the completion of extraction and transformation of raw data, load phase was taking a significant amount of time by dominating the overall ETL duration. This wait duration fundamentally defined by the *SQL\*Net more data from client* network wait event. By business analysis, it is set as a requirement to achieve a load rate of 500 GB/hour for startup and then to reach up a rate of 1 TB/hour gradually.

The best solution in terms of performance for the problem was thought to be a clustered file system implementation. The solution offers high data rates because both ETL would be writing to disk array and SQL\*Loader would be reading from the same disk array. However due to operational maintenance and extra licensing issues, the solution has been rejected.

Another proposed solution was to *trunk* two or more 1 GbE network cards to obtain a larger bandwidth. Unfortunately, the argument against this solution was the use of many host slots for just this purpose. As a result this solution has also been rejected.

Finally instead of trunking a number of network cards, use of single 10 GbE networking card[1] was proposed. Among other solutions this solution has been accepted because of its easy implementation and exceptable performance. In this paper you will find the performance evaluation of this solution at Solaris and Oracle level.

The rest of the paper is organized as follows. *Section 2* describes the modifications held on Solaris 10 TCP kernel parameters to adapt it to 10 GbE. *Section 3* describes the Oracle server parameter optimization for SQL\*Loader performance. *Conclusion* section summarizes the performance results and the overall work. There are also two appendix sections. *Appendix A* explains the method of measuring the UNIX level network performance. Whereas, *Appendix B* explains the measurement of the same value at Oracle level.

## II. TUNING SOLARIS 10 TCP STACK FOR 10 GbE

There are a number of advised Solaris configuration changes to increase TCP performance of 10 GbE

### A. Use of Jumbo Frames

The very first implementation at operating system level was to enable Jumbo frames. Standard ethernet frames are 1500 bytes long(MTU size). For high speed networks Solaris introduces the Jumbo frames of size 8000 bytes.

### B. Kernel Parameter Changes

<sup>1</sup> There are also a number of advised kernel parameter changes on Solaris 10 TCP[11]. Those are less significant in terms of performance with compared to enabling Jumbo frame because both servers are within the same LAN. Those parameters have significance on WAN or larger networks.

1) *tcp\_xmit\_hiwat*: Parameter defines the default send window size in bytes. Default value is defined to be 49,152 bytes. For high-speed networks the advised values is 400,000.

2) *tcp\_recv\_hiwat*: Parameter defines the default receive window size in bytes. Default parameter value is defined to be 49,152 bytes. For high-speed networks the advised values is 400,000.

3) *tcp\_deferred\_acks\_max*: Parameter specifies the maximum number of TCP segments received from remote destinations(not directly connected) before an ACK is generated. TCP segments are measured in units of maximum segment size (MSS) for individual connections. If set to 0 or 1, no ACKs are delayed, assuming all segments are 1 MSS long. The actual number is dynamically calculated for each connection. Default parameter value is defined to be 2 bytes. For high-speed networks the advised values is 16.

4) *tcp\_local\_dacks\_max*: Parameter specifies the maximum number of TCP segments received from directly connected destinations before an ACK is generated. TCP segments are measured in units of maximum segment size (MSS) for individual connections. If set to 0 or 1, no ACKs are delayed,

<sup>1</sup>Most of those parameters increase the number of frames send before waiting for an acknowledgement. This brings out the efficient use of backward channel and prevents from chattery TCP windowing activity.

assuming all segments are 1 MSS long. The actual number is dynamically calculated for each connection. Default parameter value is defined to be 8 bytes. For high-speed networks the advised values is 16.

5) *tcp\_max\_buf*: Parameter defines the maximum buffer size in bytes. This parameter controls how large the send and receive buffers are set to by an application that uses *setsockopt(3XNET)*. Default parameter value is defined to be 1,048,576 bytes. For high-speed networks the advised values is 2,097,152.

6) *tcp\_cwnd\_max*: Defines the maximum value for TCP congestion window(*cwnd*) in bytes. Default parameter value is defined to be 1,048,576 bytes. For high-speed networks the advised values is 2,097,152.

After performing the necessary changes on Solaris 10 TCP, network transfer rate of 413 MB/s has been achieved<sup>2</sup>. For the details of measuring network performance at operating system level refer to Appendix A

### III. TUNING ORACLE SERVER

After reaching a satisfactory result at Solaris level, the next step was to obtain the similar results (80%-90% of OS throughput) on Oracle SQL\*Loader. For this purpose we have worked on two classes of parameters on Oracle server. First class was the optimization of SQL\*Net configuration. The latter one was the customization of SQL\*Loader parameters.

#### A. SQL\*Net Optimization

Among various parameters the most effective one in changing the performance of data load was *Session Data Unit* (SDU) parameter. SDU is simply the data unit of SQL\*Net. Oracle SQL\*Net transfers at most SDU sized data units to TCP/UDP/SDP layer at which units are further aggregated or splitted to fit into a MTU size.

Default value of this parameter is 2K which is sufficient for many Oracle applications. But once the Jumbo frame size is concerned at TCP layer, we have found that 32K SDU size offers the best result. SDU parameter has to be set at both client and server site. On client, the entry in *tnsnames.ora* file has to be of the form

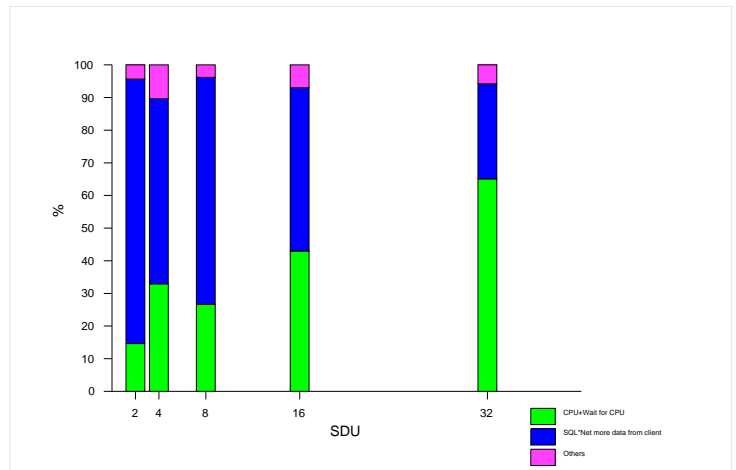


Fig. 1. SDU vs Wait Events

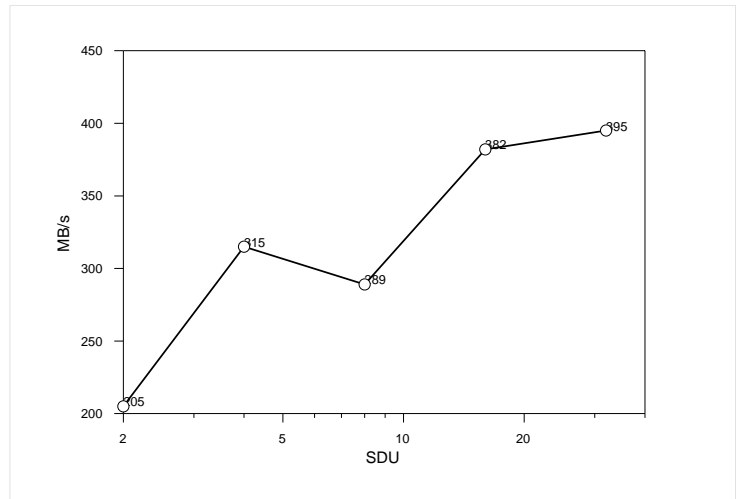


Fig. 2. SDU vs Average SQL\*Loader Throughput

```

DDS =
  (DESCRIPTION =
    (SDU=32767)
    (ADDRESS =
      (PROTOCOL = TCP)
      (HOST = dummy)
      (PORT = 1521)
    )
    (CONNECT_DATA =
      (SERVER = DEDICATED)
      (SERVICE_NAME = dummy_db)
    )
  )

```

Listing 1. Sample TNSNAMES.ORA entry

<sup>2</sup>The reason of this low transfer rate far from the theoretical maximum of 1280 MB/s was addressed to 66 MHz clock speed of slot used by the networking card which supports up to 133 MHz.

On server side, the entry in *listener.ora* file has to be of the form

```

LISTENER_DDSPRIV =
  (DESCRIPTION_LIST =
    (DESCRIPTION =
      (ADDRESS =
        (PROTOCOL = TCP)
        (HOST = dummy)
        (PORT= 1521)
      )
    )
  )
)

SID_LIST_LISTENER_DDSPRIV =
  (SID_LIST =
    (SID_DESC =
      (SDU = 32767)
      (SID_NAME = dummy_db)
      (ORACLE_HOME = /u01/app/oracle/product/10.2.0.3)
    )
  )
)

```

Listing 2. Sample LISTENER.ORA entry

Keep in mind that forgetting to set SDU value in one of those files may inhibit the increase in performance. That's because during the connection setup, Oracle will negotiate over the smallest SDU value in *tnsnames.ora* and *listener.ora*

### B. Number of Concurrent SQL\*Loader Sessions

Yet another important parameter for the parallel load performance is related to the number of concurrent SQL\*Loader sessions. After a certain point further increasing the number of sessions doesn't help in performance. We have found optimal number of parallel sessions to be **44** for our case.<sup>3</sup>

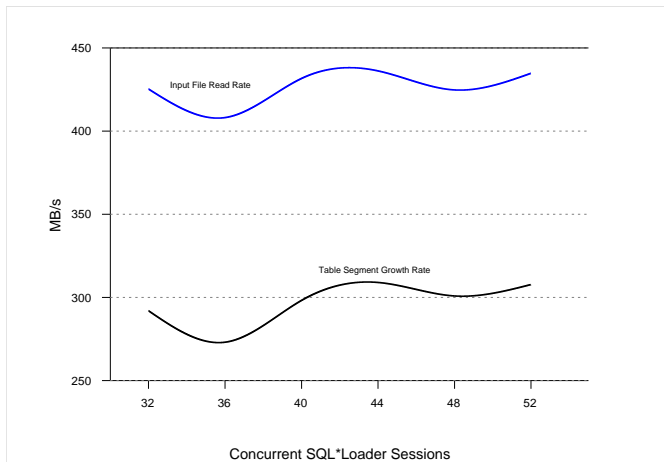


Fig. 3. Effect of the Number of Parallel SQL\*Loader Sessions on Load Performance

Keep in mind that increasing the number of parallel SQL\*Loader slaves further, not only increases the CPU consumption over the host systems but also gives no significant load performance improvement.

### C. Other Miscellaneous SQL\*Loader Parameters

There are also some other customizable parameters for parallel direct path SQL\*Loader performance. In [9], Oracle lists them as

- READSIZE
- COLUMNARRAYROWS
- STREAMSIZE

Although the customization of those parameters are told to help in increasing SQL\*Loader performance, we couldn't obtain a significant performance improvement by manipulating them.

Those optimizations over Oracle server let us obtain the desired SQL\*Loader throughput, even go beyond it. For the details of measuring network performance at Oracle level refer to Appendix B.

## IV. CONCLUSION

To sum up, we've reached an average load rate of 396 MB/s (1.4 TB/h). Critical changes in hardware, Solaris and Oracle configurations are:

- Use of 10 Gbit network card.
- Use of Jumbo frames (MTU=8000) instead of standard Ethernet frames (MTU = 1500)
- Change on advised Solaris 10 kernel parameters
- 32K Session data unit (SDU) parameter on *listener.ora* and *tnsnames.ora*.
- Use of 44 parallel SQL\*Loader sessions.

## APPENDIX A

### MEASURING NETWORK PERFORMANCE ON SOLARIS 10

Built-in tool to measure the network performance in Solaris is the *netstat* tool[2]. *netstat* allows to see the number of in/out TCP/UDP frames, ICMP packages, and many other things.

```

$ netstat -I ixge0 1
input  ixge0      output      input  (Total)  output
packets errs  packets  errs  colls  packets  errs  packets  errs  colls
15289  0    30058    0    0    45624    0    43122    0    0

```

Listing 3. Sample *netstat* Output

Although it is simple to use, *netstat* is not a network calibration tool. In other words, it just simply monitors the activity of network and it is not capable of generating any network traffic. People are using tools like FTP to generate this load. However FTP is a file transferring application and may not emulate the behavior of a tool like SQL\*Loader running on SQL\*Net layer which has its own payload size and so on.

There are a number of tools to generate TCP/UDP workload and test the performance of a network. Recently Microsystems performance team also introduces *uperf* (Ultimate Performance) [3][4] for this purpose. With compared to similar tools like TTCP[5] and IPerf[6] it offers a simpler usage and more flexibility in terms of defining the network workloads.

### A. Working with *uperf*

The effort to perform a network stress test with *uperf* is pretty simple. It consists of three steps

<sup>3</sup>Host machine has 56 CPUs

1) Create you workload definition XML: *uperf* uses an XML template to define a possible workload scenario. We have already prepared a simple file (*sqlloader.xml*) ready to use for SQL\*Loader type of traffic generation

```
<profile>
  <group nthreads="$nt">
    <transaction iterations="1">
      <flowop type="accept"
        options="remotehost=$h protocol=tcp
        wndsz=50k"/>
    </transaction>
    <transaction duration="90s">
      <flowop type="write" options="size=$sdu"/>
    </transaction>
    <transaction iterations="1">
      <flowop type="disconnect"/>
    </transaction>
  </group>
</profile>
```

Listing 4. *uperf* SQL\*Loader workload definition XML

2) Run *uperf* instance at server side: A *uperf* instance is started at the server side (for our test this the Solaris host on which database resides) by using *s* option

```
$ uperf -s
```

Listing 5. Starting server site *uperf*

3) Run *uperf* instance at client side: Final step is to run client side (the Solaris on which SQL\*Loader is being executed) *uperf* instance after setting required environment variables.

```
$ export 4nt=32
$ export 5h=192.168.10.2
$ export 6sdu=2k
$ uperf -m sqlloader.xml -a -p -f
```

Listing 6. Starting client site *uperf*

After running for ~90 seconds *uperf* outputs a detailed performance evaluation report

<sup>4</sup>*nt* is the concurrent threads generating network load  
<sup>5</sup>*h* is the IP of Solaris machine running the server instance of *uperf*  
<sup>6</sup>*sdu* is the size of each payload sent to TCP layer. 2K is the default value for SQL\*Net SDU

```
Starting 32 threads running profile:sqlloader ... 0.01 seconds
Txn1      0 /1.01(s) =          0          32op/s
Txn2 29.89GB /90.11(s) =    2.85Gb/s    10870op/s
```

---

```
Total 29.89GB /92.13(s) =    2.79Gb/s    10632op/s
** Error signalling strands
Group Details
```

---

```
Group0      0 /18445904148.24(s) =          0
```

---

Txn	Count	avg	cpu	max	min
Txn0	32	18.99ms	0.00 ns	29.19ms	16.41ms
Txn1	979505	2.94ms	0.00 ns	2.21s	640.00ns

---

Flowop	Count	avg	cpu	max	min
accept	32	18.98ms	0.00 ns	29.18ms	16.40ms
write	979474	2.94ms	0.00 ns	2.21s	38.73us

---

```
Netstat statistics for this run
```

---

Nic	opkts/s	ipkts/s	obits/s	ibits/s
ce0	267	285	1.17Mb/s	1.27Mb/s
dman0	0	0	48.02b/s	36.03b/s
eri0	0	0	48.02b/s	36.03b/s
ixge0	57745	23362	3.05Gb/s	12.53Mb/s

---

```
Run Statistics
```

Hostname	Time	Data	Throughput	Operations	Errors
192.168.10.2	92.13 s	29.89GB	2.79Gb/s	2088757	0.00
master	92.13 s	29.89GB	2.79Gb/s	979506	0.00

---

```
Difference() -0.00    -0.00 0.00    -113.25 0.00
```

Listing 7. Sample *uperf* Output

As you see measuring the capability of your network is that easy. For this test we can achieve up to 2.79 Gbit/s. This is very close to what you will see for a real SQL\*Loader load task.

## APPENDIX B

### MEASURING NETWORK PERFORMANCE ON ORACLE

In order to measure the loading performance of SQL\*Loader, we simply use SQL\*Loader log files.

```
$ cat /export/home/hsensoy/sql_loader_test/log/ctl* |
grep "Elapsed time was:"
Elapsed time was: 00:01:45.94
Elapsed time was: 00:01:37.38
Elapsed time was: 00:01:41.41
```

Listing 8. Sample *netstat* Output

By simply averaging over those load stream durations, we obtain an average loading rate which represents the average SQL\*Loader performance.

## REFERENCES

- [1] Sun Microsystems, *Sun Multithreaded 10 GbE (Gigabit Ethernet) Networking Cards*, <http://www.sun.com/products/networking/ethernet/10gigaethernet/index.xml>
- [2] Sun Microsystems, *System Administration Guide:IP Services*, <http://docs.sun.com/app/docs/doc/806-4075/6jd69oa>
- [3] Sun Microsystems Performance Availability Engineering Group, *Unified Performance tool for networking (uperf)*, <http://www.uperf.org/manual.html>
- [4] Eric He, *A Handy Tool for Network Performance Measurement-uPerf*, <http://blogs.sun.com/eh/>
- [5] PCAUSA, *TTCP Utility*, <http://www.pcausa.com/Utilities/pcattcp.htm>
- [6] NLANR/DAST, *Iperf*, <http://sourceforge.net/projects/iperf>
- [7] Oracle, *Metalink Note 222769.1*, <http://metalink.oracle.com>
- [8] Oracle, *Conventional and Direct Path Loads*, <http://tahiti.oracle.com>
- [9] Oracle, *Metalink Note 67983.1*, <http://metalink.oracle.com>
- [10] X. Lian and L. Chen, *Monochromatic and Bichromatic Reverse Skyline Search over Uncertain Databases*
- [11] Sun Microsystems, *TCP/IP Tunable Parameters for Solaris 10*, <http://docs.sun.com/app/doc/817-0404/appendixa-28>