NFS Tuning for Oracle: Introducing DTrace

http://dboptimizer.com

Kyle Hailey
June 2011
Intro

- Who am I
  - why am I interested in NFS tuning for Oracle?
- DAS vs NAS vs SAN
  - Throughput
  - Latency
- NFS configuration issues for non-RAC, non-dNFS
  - Network topology
  - TCP configuration
  - NFS Mount Options
Fast, Non-disruptive Deployment

- Production
- Development
- Q/A
- Reporting
- UAT

Sync via standard APIs

Provision and refresh from any time or SCN

1 TB

300MB

http://dboptimizer.com
Combine Prod Support and DR

Production SAP Landscape

PRIMARY DATACENTER

Standby via DataGuard

Dev: Prod Support

QA: Prod Support

Dev: Project

QA: Project

R/3 ERP

BW

CRM

GTS

T&E

Sales CRM

STAPLES

that was easy:

Corporate Express

http://dboptimizer.com

>> Strictly Confidential
Which to use?
DAS is out of the picture
Fibre Channel

http://dboptimizer.com
Manly men only use Fibre Channel
NFS - available everywhere
NFS is attractive but is it fast enough?
# DAS vs NAS vs SAN

<table>
<thead>
<tr>
<th></th>
<th>attach</th>
<th>Agile</th>
<th>expensive</th>
<th>maintenance</th>
<th>speed</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DAS</strong></td>
<td>SCSI</td>
<td>no</td>
<td>no</td>
<td>difficult</td>
<td>fast</td>
</tr>
<tr>
<td><strong>NAS</strong></td>
<td>NFS - Ethernet</td>
<td>yes</td>
<td>no</td>
<td>easy</td>
<td>??</td>
</tr>
<tr>
<td><strong>SAN</strong></td>
<td>Fibre Channel</td>
<td>yes</td>
<td>yes</td>
<td>difficult</td>
<td>fast</td>
</tr>
</tbody>
</table>
speed

Ethernet
• 100Mb 1994
• 1GbE - 1998
• 10GbE – 2003
• 40GbE – est. 2012
• 100GE – est. 2013

Fibre Channel
• 1G 1998
• 2G 2003
• 4G – 2005
• 8G – 2008
• 16G – 2011
Ethernet vs Fibre Channel
Throughput vs Latency

Throughput
- Width
- netio

Latency
- Length
- traceroute
Throughput: netio

- **100MbE** \( \approx 10 \text{MB/sec} \)
- **1GbE** \( \approx 100 \text{MB/sec} \) (125MB/sec max)
  - 30-60MB/sec typical, single threaded, mtu 1500
- **10GbE** \( \approx 1 \text{GB/sec} \)
  - \( b = \text{bits}, B = \text{bytes} \) (ie 8 bits)

Server machine

Test with

```
netio -s -b 32k -t -p 1234
```

Target

```
netio -b 32k -t -p 1234 delphix_machine
Receiving from client, packet size 32k ... 104.37 MByte/s
Sending to client, packet size 32k ... 109.27 MByte/s
Done.
```
Wire Speed – where is the hold up?

<table>
<thead>
<tr>
<th>ms</th>
<th>us</th>
<th>ns</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.000</td>
<td>000</td>
<td>000</td>
</tr>
</tbody>
</table>

Light travels at 0.3m/ns
If wire speed is 0.2m/ns

Data Center 10m = 50ns

LA to London is 30ms
LA to SF is 3ms
(5us/km)

Physical 8K random disk read 6-8ms
Physical small write 1-2ms sequential
4G FC vs 10GbE

Why would FC be faster?

8K block transfer times

- 8GB FC = 10us
- 10G Ethernet = 8us
More stack more latency

dNFS

NFS only

Not on FC
Oracle and SUN benchmark

200us overhead of NFS over DAS

8K blocks 1GbE with Jumbo Frames, Solaris 9, Oracle 9.2

<table>
<thead>
<tr>
<th>I/O Response Time</th>
<th>UFS</th>
<th>NFS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>7.19 ms</td>
<td>7.39 ms</td>
</tr>
</tbody>
</table>

Database Performance with NAS: Optimizing Oracle on NFS
Revised May 2009 | TR-3322

80us is from wire transfer which goes down to 8us on 10GbE (like talking faster)

Latency has gotten even better.
8K block NFS latency overhead

- 1GbE -> 80us
- 10GbE -> 8us
- 200us on 1GbE = 128us on 10GbE
- If spindle I/O is 7ms
- Then NFS is overhead is

\[
\frac{0.128\text{ms}}{7\text{ms}} \times 100 = 1.8\% \text{ latency increase over DAS}
\]

Worth choosing FC?

FC also has a latency overhead so the difference between FC and NFS is even smaller than 1.8%
NFS why the bad reputation?

- Given 1.8% overhead why the reputation?
- Historically slower
- Setup can make a big difference
  1. Network topology and load
  2. NFS mount options
  3. TCP configuration
- Compounding issues
  - Oracle configuration
  - I/O subsystem response
Network Topology

- Hubs
- Routers
- Switches
- Hardware mismatch
- Network Load
## HUBs

<table>
<thead>
<tr>
<th>Layer</th>
<th>Name</th>
<th>Routers</th>
<th>IP addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>Application</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Presentation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Session</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Transport</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Network</td>
<td>Routers</td>
<td>IP addr</td>
</tr>
<tr>
<td>2</td>
<td>Datalink</td>
<td>Switches</td>
<td>mac addr</td>
</tr>
<tr>
<td>1</td>
<td>Physical</td>
<td>Hubs</td>
<td>Wire</td>
</tr>
</tbody>
</table>

- Broadcast, repeaters
- Risk collisions
- Bandwidth contention
Routers

- Routers can add 300-500us latency
- If NFS latency is 350us (typical non-tuned) system
- Then each router multiplies latency 2x, 3x, 4x etc

### Network Table

<table>
<thead>
<tr>
<th>Layer</th>
<th>Name</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Network</td>
<td>Routers</td>
<td>IP addr</td>
</tr>
<tr>
<td>2</td>
<td>Datalink</td>
<td>Switches</td>
<td>mac addr</td>
</tr>
<tr>
<td>1</td>
<td>Physical</td>
<td>Hubs</td>
<td>Wire</td>
</tr>
</tbody>
</table>
Routers: traceroute

$ traceroute 101.88.123.195
1 101.88.229.181 (101.88.229.181)  0.761 ms  0.579 ms  0.493 ms
2 101.88.255.169 (101.88.255.169)  0.310 ms  0.286 ms  0.279 ms
3 101.88.218.166 (101.88.218.166)  0.347 ms  0.300 ms  0.986 ms
4 101.88.123.195 (101.88.123.195)  1.704 ms  1.972 ms  1.263 ms
sums (not shown)  3.122  3.137  3.021

$ traceroute 172.16.100.144
1 172.16.100.144 (172.16.100.144)  0.226 ms  0.171 ms  0.123 ms

3.0 ms NFS on slow network
0.2 ms NFS good network
6.0 ms Typical physical read
Multiple Switches

- Two types of Switches
  - Store and Forward
    - 1GbE 50-70us
    - 10GbE 5-35us
  - Cut through
    - 10GbE 300-500ns

<table>
<thead>
<tr>
<th>Layer</th>
<th>Name</th>
<th>Sub-layer</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Network</td>
<td>Routers</td>
<td>IP addr</td>
</tr>
<tr>
<td>2</td>
<td>Datalink</td>
<td>Switches</td>
<td>mac addr</td>
</tr>
<tr>
<td>1</td>
<td>Physical</td>
<td>Hubs</td>
<td>Wire</td>
</tr>
</tbody>
</table>
Hardware mismatch

- Speeds and duplex are often negotiated

Example Linux:

```
$ ethtool eth0
Settings for eth0:
  Advertised auto-negotiation: Yes
  Speed: 1000Mb/s
  Duplex: Full
```

- Check that values are as expected
Busy Network

- Traffic can congest network
  - Caused drop packets
  - Out of order packets
  - Collisions on hubs, probably not with switches
Busy Network Monitoring

- Visibility difficult from any one machine
  - Client
  - Server
  - Switch(es)

```
$ nfsstat -cr
Client rpc:
Connection oriented:
badcalls  badxids  timeouts  newcreds  badverfs  timers
89101      6       0         5         0       0

$ netstat -s -P tcp 1
TCP       tcpRtoAlgorithm  =  4     tcpRtoMin  =  400
          tcpRetransSegs    = 5986    tcpRetransBytes = 8268005
          tcpOutAck        =49277329  tcpOutAckDelayed = 473798
          tcpInDupAck      =357980    tcpInAckUnsent =  0
          tcpInUnorderSegs =10048089  tcpInUnorderBytes =16611525
          tcpInDupSegs      = 62673    tcpInDupBytes = 87945913
          tcpInPartDupSegs  =  15      tcpInPartDupBytes =  724
          tcpRttUpdate      =4857114   tcpTimRetrans =  1191
          tcpTimRetransDrop =  6       tcpTimKeepalive =  248
```
Busy Network Testing

Netio is available here: http://www.ars.de/ars/ars.nsf/docs/netio

On Server box:
```
netio -s -b 32k -t -p 1234
```

On Target box:
```
netio -b 32k -t -p 1234 delphix-machine
NETIO - Network Throughput Benchmark, Version 1.31
(C) 1997-2010 Kai Uwe Rommel
TCP server listening.
TCP connection established ...
Receiving from client, packet size 32k ... 104.37 MByte/s
Sending to client, packet size 32k ... 109.27 MByte/s
Done.
```
TCP Configuration

- MTU
- Socket buffer sizes
- TCP window size
- TCP congestion window sizes
MTU 9000: Jumbo Frames

- MTU – maximum Transfer Unit
  - Typically 1500
  - Can be set 9000
  - All components have to support
    - If not error and/or hangs

Delayed Acknowledgement
Jumbo Frames: MTU 9000

8K block transfer

Default MTU 1500

<table>
<thead>
<tr>
<th>delta</th>
<th>send</th>
<th>recd</th>
</tr>
</thead>
<tbody>
<tr>
<td>152</td>
<td>132</td>
<td>--&gt; 164</td>
</tr>
<tr>
<td>40</td>
<td>1448</td>
<td>--&gt;</td>
</tr>
<tr>
<td>67</td>
<td>1448</td>
<td>--&gt;</td>
</tr>
<tr>
<td>66</td>
<td>1448</td>
<td>--&gt;</td>
</tr>
<tr>
<td>53</td>
<td>1448</td>
<td>--&gt;</td>
</tr>
<tr>
<td>87</td>
<td>1448</td>
<td>--&gt;</td>
</tr>
<tr>
<td>95</td>
<td>952</td>
<td>--&gt;</td>
</tr>
<tr>
<td>= 560</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Now with MTU 900

<table>
<thead>
<tr>
<th>delta</th>
<th>send</th>
<th>recd</th>
</tr>
</thead>
<tbody>
<tr>
<td>273</td>
<td></td>
<td>--&gt; 164</td>
</tr>
<tr>
<td>8324</td>
<td></td>
<td>--&gt;</td>
</tr>
</tbody>
</table>

Change MTU

# ifconfig eth1 mtu 9000 up

Warning: MTU 9000 can hang if any of the hardware in the connection is configured only for MTU 1500
TCP Sockets

- Memory allocated to TCP send and receive buffers.
- If maximum is reached packets are dropped.

Excellent book

**LINUX**

- Set the max OS send buffer size (wmem) and receive buffer size (rmem) to 8 MB for queues on all protocols:
  - `sysctl -w net.core.wmem_max=8388608`
  - `sysctl -w net.core.rmem_max=8388608`

- These specify the amount of memory that is allocated for each TCP socket when it is created. In addition, you should also use the following commands for send and receive buffers. They specify three values: minimum size, initial size, and maximum size:
  - `sysctl -w net.ipv4.tcp_rmem="4096 87380 8388608"`
  - `sysctl -w net.ipv4.tcp_wmem="4096 87380 8388608"`
TCP window sizes

- maximum amount of data to send or receive
- Subset of the TCP socket sizes

TCP window size

\[ \text{TCP window size} = \text{latency} \times \text{throughput} \]

For example, with 1ms latency over a 1Gb network

\[ \text{TCP window size} = 1\text{Gb/sec} \times 0.001\text{s} = 100\text{Mb/sec} \times 1\text{Byte}/8\text{bits} = 125\text{KB} \]

Optimal TCP window size is generally cited as being twice this value

Optimal TCP window size = \[2 \times \text{latency} \times \text{throughput} = \text{RTT} \times \text{throughput}\]
## Congestion Window

<table>
<thead>
<tr>
<th>Unack Bytes</th>
<th>Unack Byte</th>
<th>Delta</th>
<th>Bytes Sent</th>
<th>Bytes Received</th>
<th>Send Window</th>
<th>Receive Window</th>
<th>Cong Window</th>
</tr>
</thead>
<tbody>
<tr>
<td>139760</td>
<td>0</td>
<td>31</td>
<td>1448 \</td>
<td></td>
<td>195200</td>
<td>131768</td>
<td>144800</td>
</tr>
<tr>
<td>139760</td>
<td>0</td>
<td>33</td>
<td>1448 \</td>
<td></td>
<td>195200</td>
<td>131768</td>
<td>144800</td>
</tr>
<tr>
<td>144104</td>
<td>0</td>
<td>29</td>
<td>1448 \</td>
<td></td>
<td>195200</td>
<td>131768</td>
<td>146248</td>
</tr>
<tr>
<td>145552</td>
<td>0</td>
<td>31</td>
<td>/ 0</td>
<td></td>
<td>195200</td>
<td>131768</td>
<td>144800</td>
</tr>
<tr>
<td>145552</td>
<td>0</td>
<td>41</td>
<td>1448 \</td>
<td></td>
<td>195200</td>
<td>131768</td>
<td>147696</td>
</tr>
<tr>
<td>147000</td>
<td>0</td>
<td>30</td>
<td>/ 0</td>
<td></td>
<td>195200</td>
<td>131768</td>
<td>144800</td>
</tr>
<tr>
<td>147000</td>
<td>0</td>
<td>22</td>
<td>1448 \</td>
<td></td>
<td>195200</td>
<td>131768</td>
<td>76744</td>
</tr>
<tr>
<td>147000</td>
<td>0</td>
<td>28</td>
<td>/ 0</td>
<td></td>
<td>195200</td>
<td>131768</td>
<td>76744</td>
</tr>
<tr>
<td>147000</td>
<td>0</td>
<td>18</td>
<td>1448 \</td>
<td></td>
<td>195200</td>
<td>131768</td>
<td>76744</td>
</tr>
</tbody>
</table>

Unacknowledged bytes
- Hits the congestion window size
- Congestion window size is drastically lowered
# NFS mount options

- Forcedirectio
- Rsize / wsize
- Actimeo=0, noac

<table>
<thead>
<tr>
<th>OS</th>
<th>Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sun Solaris</td>
<td>rw,bg,hard,rsize=32768,wsize=32768,vers=3,[forcedirectio or llock],nointr,proto=tcp,suid</td>
</tr>
<tr>
<td>AIX</td>
<td>rw,bg,hard,rsize=32768,wsize=32768,vers=3,cio,intr,timeo=600,proto=tcp</td>
</tr>
<tr>
<td>HPUX</td>
<td>rw,bg,hard,rsize=32768,wsize=32768,vers=3,nointr,timeo=600,proto=tcp, suid, forcedirectio</td>
</tr>
<tr>
<td>Linux</td>
<td>rw,bg,hard,rsize=32768,wsize=32768,vers=3,nointr,timeo=600,proto=tcp,actimeo=0</td>
</tr>
</tbody>
</table>
Forcedirectio

- Causes UNIX file cache to be bypassed
- Data is read directly into UNIX
- Controlled by init.ora parameter
  - Filesystemio_options=SETALL or directio
  - Except HPUX where mount option is the only way
  - Solaris doesn’t require the mount option

<table>
<thead>
<tr>
<th>Sun Solaris</th>
<th>Forcedirectio – sets directio but not required</th>
<th>Filesystemio_options will set directio without mount option</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIX</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HPUX</td>
<td>Forcedirectio – only way to set directio</td>
<td>Filesystemio_options has no affect</td>
</tr>
<tr>
<td>Linux</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Direct I/O

query doing

77951 physical reads for the second execution
(ie when data should already be cached)

- 60 secs => direct I/O
- 5 secs => no direct I/O
- 2 secs => SGA

- Why use direct I/O?
Direct I/O

- **Advantages**
  - Faster reads from disk
  - Reduce CPU
  - Reduce memory contention
  - Faster access to data already in memory, in SGA

- **Disadvantages**
  - Less Flexible
  - More work
  - Risk of paging, memory pressure
  - Impossible to share memory between multiple databases

<table>
<thead>
<tr>
<th>Cache</th>
<th>OS</th>
<th>Rows/sec</th>
<th>Usr</th>
<th>sys</th>
</tr>
</thead>
<tbody>
<tr>
<td>FS</td>
<td>S9</td>
<td>287,114</td>
<td>71</td>
<td>28</td>
</tr>
<tr>
<td>DB</td>
<td>S9</td>
<td>695,700</td>
<td>94</td>
<td>5</td>
</tr>
</tbody>
</table>

http://blogs.oracle.com/glennf/entry/where_do_you_cache_oracle
UNIX Cache

- Hits
- Miss

< 0.2 ms

NAS/SAN Cache

- Hits
- Miss

< 0.5 ms

Disk Reads

- UNIX Cache
- NAS/SAN
- Disk Reads

Disk Read ~ 6ms

Storage Cache

- Hits
- Miss

SAN/NAS

- Hits
- Miss

SGA Buffer Cache

- Hits
- Miss

File System Cache

- Hits
- Miss
Direct I/O Challenges

Database Cache usage over 24 hours

DB1
Europe

DB2
US

DB3
Asia
ACTIMEO=0, NOAC

- Disable client side file attribute cache
- Increases NFS calls
- Significantly increases latency and reduces throughput
- Not required on single instance Oracle
- Metalink says it’s required on LINUX
- Another metalink it should be taken off

=> It should be take off
rsize/wsize

- NFS transfer buffer size
- Oracle says use 32K
- Platforms support higher values and can significantly impact throughput

<table>
<thead>
<tr>
<th></th>
<th>rsize=32768, wsize=32768, max is 1M</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sun Solaris</td>
<td></td>
</tr>
<tr>
<td>AIX</td>
<td></td>
</tr>
<tr>
<td>HPUX</td>
<td></td>
</tr>
<tr>
<td>Linux</td>
<td></td>
</tr>
</tbody>
</table>

On full table scans using 1M has halved the response time over 32K
Db_file_multiblock_read_count has to be large enough to take advantage of the size
NFS Overhead Physical vs Cached IO

100us extra over 6ms spindle read is small
100us extra over 100us cache read is 2x as slow
SAN cache is expensive – use it for write cache
Target cache is cheaper – put more on if needed

Storage (SAN)

Cache

Physical

- Physical vs NFS
- Physical multi switch
- Physical routers
- Cached vs NFS
- Cached vs switches
- Cached vs routers

NFS + network
Reads
Conclusions

- NFS performance can come close to FC
- Requires
  - Network topology be clean – no routers, fast switches
  - Mount options correct (and/or dNFS, version 11g)
    - Rsize/wsize at maximum
    - Avoid actimeo=0 and noac
  - TCP configuration – MTU 9000 (tricky)
- Drawbacks
  - NFS failover can take 10s of seconds
  - With Oracle 11g dNFS can be handled transparently

Conclusion: Give NFS some more love
gigabit switch can be anywhere from 10 to 50 times cheaper than an FC switch
**dtrace**

*List the names of traceable probes:*

```
dtrace -ln provider:module:function:name
```

- `-l` = list instead of enable probes
- `-n` = Specify probe name to trace or list
- `-v` = Set verbose mode

**Example**
```
dtrace -ln tcp:::send
$ dtrace -lvn tcp:::receive
5473 tcp ip tcp_output send
```

**Argument Types**
```
args[0]: pktinfo_t *
args[1]: csinfo_t *
args[2]: ipinfo_t *
args[3]: tcpinfo_t *
args[4]: tcpinfo_t *
```
typedef struct tcp pInfo {
    uintptr_t tcp_s_addr;
    int tcp_s_local;
    int tcp_s_active;
    uint16_t tcp_s_lport;
    uint16_t tcp_s_rport;
    string tcp_s_laddr;
    string tcp_s_raddr;
    int32_t tcp_s_state;
    uint32_t tcp_s_iss;
    uint32_t tcp_s_suna;
    uint32_t tcp_s_snxt;
    uint32_t tcp_s_rack;
    uint32_t tcp_s_rnxt;
    uint32_t tcp_s_swnd;
    int32_t tcp_s_snd_ws;
    uint32_t tcp_s_rwnd;
    uint32_t tcp_s_rcv_ws;
    uint32_t tcp_s_cwnd;
    uint32_t tcp_cwnd_ssthresh;
    uint32_t tcp_s_sack_fack;
} /* is delivered locally, boolean */
/* active open (from here), boolean */
/* local port */
/* remote port */
/* local address, as a string */
/* remote address, as a string */
/* TCP state */
/* Initial sequence # sent */
/* sequence # sent but unacked */
/* next sequence # to send */
/* sequence # we have acked */
/* next sequence # expected */
/* send window size */
/* send window scaling */
/* receive window size */
/* receive window scaling */
/* congestion window */
/* threshold for congestion avoidance */
/* SACK sequence # we have acked */
Dtrace

tcp:::send, tcp:::receive
{
    delta = timestamp - walltime;
    walltime = timestamp;
    printf("%6d %6d %6d %8d %8s %8d %8d %8d %8d %d  \n",
            delta / 1000, 
            ",",
            args[3] -> tcps_swnd, 
            args[3] -> tcps_rwnd, 
            args[3] -> tcps_cwnd, 
            args[3] -> tcps_retransmit
        );
}

tcp:::receive
{
    delta = timestamp - walltime;
    walltime = timestamp;
    printf("%6d %6d %6d %8s / %8d %8d %8d %8d %8d %d  \n",
            delta / 1000, 
            ",",
            args[3] -> tcps_swnd, 
            args[3] -> tcps_rwnd, 
            args[3] -> tcps_cwnd, 
            args[3] -> tcps_retransmit
        );
}
#!/usr/sbin/dtrace -s
#pragma D option quiet
#pragma D option defaultargs
inline string ADDR=$$1;
tcp::send, tcp::send
/ ( args[2]->ip_daddr == ADDR || ADDR == NULL ) /
{
    nfs[args[1]->cs_cid]=1; /* this is an NFS thread */
delta= timestamp- walltime;
    walltime=timestamp;
    printf("%6d %6d %6d %8s %8d %8d %8d %8d %d \n",
        args[3]->tcps_snxt - args[3]->tcps_suna,
        args[3]->tcps_rnxt - args[3]->tcps_rack,
        delta/1000,
        args[2]->ip_plength - args[4]->tcp_offset,
        "",
        args[3]->tcps_swnd,
        args[3]->tcps_rwnd,
        args[3]->tcps_cwnd,
        args[3]->tcps_retransmit
    );
}
tcp::receive
/ ( args[2]->ip_saddr == ADDR || ADDR == NULL ) && nfs[args[1]->cs_cid] /
{
    delta=timestamp- walltime;
    walltime=timestamp;
    printf("%6d %6d %6d %8s / %8d %8d %8d %8d %8d %d \n",
        args[3]->tcps_snxt - args[3]->tcps_suna,
        args[3]->tcps_rnxt - args[3]->tcps_rack,
        delta/1000,
        "",
        args[2]->ip_plength - args[4]->tcp_offset,
        ""
    );