NFS over TCP, Again
Connectathon 2006
March 1

Mike Eisler

email2mre-cthon2006@yahoo.com

Introduction

- Significance of “Again”
  - Third time I’ve presented on this topic at cthon
    - 1993
    - 1996
      http://www.connectathon.org/talks96/nfstcp.pdf
- Why bore everyone a third time?
  - 10+ years of experience and UDP persists
    - 55% of NFS operations sent to NetApp filers use UDP, 45% TCP
  - Interesting interoperability issues still exist

Why NFS/TCP?

- Until we get NFSv4.1 sessions, Exactly Once Semantics are approximated via the duplicate request cache
  - Use of TCP reduces the chance that the duplicate request cache will be needed, hence reduces the chance of a bad miss
- TCP is more secure: attackers can't just spoof a source IP address and send a UDP message that vandalizes data
  - TCP requires a round trip to establish connection
- TCP versus UDP performance is no longer an issue for most vendors
  - http://www.spec.org/sfs97r1/results/sfs97r1.html
  - Usually UDP numbers aren't posted, or if they are, they are about 5% better than TCP

The difference between NFS/UDP and NFS/TCP

- An NFS operation over UDP usually starts with a low RPC timeout
  - At each timeout, the operation is retried (retransmitted), with the same XID, and the timeout doubled
  - Once the maximum number of retransmissions is reached, either a failure is reported to the application (soft mount) or the operation is tried again with the low timeout
  - E.g., Solaris 10. Initial timeout of 1.1 seconds, retrans count of 5. On retransmit, timeout is doubled only if less than 20 seconds.
    - 1.1 + 2 * 1.1 + 4 * 1.1 + 8 * 1.1 + 16 * 1.1 + 20 = 54.1 seconds of total timeout
    - I.e. if hard mounted, about every minute we should see "NFS server not responding"
  - E.g. Linux 2.6. Initial timeout is about 100 milliseconds. The total timeout after retransmissions works about to about a minute. (Source: Chuck Lever)

The difference between NFS/UDP and NFS/TCP (continued)

- An NFS operation over TCP usually starts with a large NFS/RPC timeout
  - Retransmissions at RPC level are zero
    - TCP itself has re-transmissions if needed
  - Once the operation times out, either a failure is reported to the application (soft mount) or the operation is tried again with the original timeout
  - E.g., Linux 2.6. Timeout of 60 seconds.
    - I.e. if hard mounted, about every 60 seconds we should see “NFS server not responding”
  - E.g. Solaris 10. Timeout is 60 seconds. But effectively this is tripled to 180 seconds, though user will see “NFS server not responding” every 60 seconds

Solaris NFS/TCP timeout history

- First prototypes had the 1.1 sec timeout (by accident)
  - In 1993, NFSv2 WRITEs over 10 mbit/sec to servers with one spindle with no NVRAM was really slow.
  - Metadata intensive operations like NFSv3 READDIR+ were a particular cause of problems
    - The time to read a block of directory entries and load each entry's inode would sometimes exceed 1.1 seconds
  - The re-issuing of the operation after 1.1 sec started a snowball effect that eventually choked bandwidth
- Quickly increased timeout to 10 seconds
- Even 10 seconds turned out to be too low; eventually led to the 60/180 second model (Solaris 2.6 and up)

What is magic about 60 and 180 seconds?

- 60 seconds: is about what NFS/UDP requests take to timeout (with 1 try + 5 retransmissions)
- 180 seconds: what was necessary to allow streaming I/O file copies to progress over network links ranging from 14.4 kbits/sec through 100 mbits/sec
- Chuck Lever put it succinctly: NFS needs two timeouts: one for network wait, one for storage subsystem wait
  - Using TCP allows TCP to manage the network wait (via TCP's own back off and retransmission algorithm) and NFS to manage the storage wait (via the timeo= mount option)

In a 1 gigabit/sec world isn't 10 seconds long enough for an NFS/TCP timeout?

Not always

- **Disk access times aren't improving as fast as networks, processors, and DRAM**
- **It is easy to find workloads (e.g. database) that are disk bound and can't benefit from server or client caching**
- **Besides, any TCP-based application should adapt to slower and/or higher latency media**
  - **10 second timeouts impairs operation over slower links**
  - **We don't see timeo= options on ftp, sftp, scp, etc.**

Downside of long (60 seconds+) timeouts: Availability

- The storage industry is under pressure to drive availability higher
  - .99999 availability is about 5 minutes per year of down time
  - .999999 availability is about 30 seconds per year of down time
- Client takes longer to detect server failover/reboot
  - Time t: client sends request, server ACKs at TCP level
  - Time t+1: server reboots/fails over without sending a FIN/RST – a disconnect indication – to client
  - Time t+60: client retries, and this triggers a TCP connection reset
  - Detecting server crash 59 seconds after it happens is incompatible with 5-6 nines of availability
- This is sometimes mitigated when there are N threads/processes using the same TCP connection
  - So time t+60 becomes t + 60/N
- Mitigation might be better done via NULL procedure “pings” (per RFC3530)

Lessons learned from NFSv4

- RFC3530 requires NFSv4 server to disconnect any time it detects an NFSv4 client sending a retry over the same connection
- Applying this rule to NFSv3/TCP turns out to be a bad idea
  - Nothing is written saying NFSv3/TCP clients cannot retry requests over the same connection
  - If the client has a very low timeout (real example: 100 milliseconds), and there's a little bit of disk wait,
  - we end up breaking connections when server detects a retry of an in progress request
  - We thus see many TCP disconnections/connections per second and very little progress (at best)

Lessons learned from NFSv4 (continued)

- RFC3530 requires NFSv4 client to disconnect any time it wants to send a retry
- Applying this rule to NFSv3/TCP can be a good idea
  - Unless the NFS/TCP timeout is as high as the TCP-level connection timeout, packet traces show the NFS client re-sending requests at the NFS level that TCP might re-send at the TCP level.
  - By disconnecting, the previous instance of the TCP connection isn't re-sending at the TCP level, resulting in less stress on network and processors

Lessons learned from NFSv4 (continued)

- A client that disconnects after an NFS-level timeout needs to be careful:
  - As soon as the client re-connects, it should start re-sending requests for incomplete RPCs
    - Otherwise throughput can degrade significantly for low NFS/TCP timeouts
- Workaround: mount –o timeo=600

Re-connecting Issues

- When a server reboots, every client wants to connect
- Clients will get ECONNREFUSED if the pending connection queue is full
- Lessons for client:
  - Avoid tight loops trying to re-connect to a server that returns ECONNREFUSED
    - Solaris seems to do fine with a 10 second delay
    - Exponential back off might be better
  - Re-connect as soon as possible after a connection is reset or timed out
    - Having interfaces that can discern ECONNREFUSED from ECONNRESET is goodness
    - If interfaces don't have this flexibility, pursue an exponential back off
- Lessons for server: that second parameter to listen() doesn't have to be 5. Longer queues are better.

Other Lessons

- **EJUKEBOX needs careful handling**
  - When the client gets NFS3ERR_JUKEBOX/NFS4ERR_DELAY, after a delay, it needs to send the retry with new XID
    - Otherwise, even after the EJUKEBOX-induced event is over, the client will hit the server's duplicate request cache
  - Exponential back off after receiving EJUKEBOX is not a good idea
    - A couple clients will potentially wait years if they get enough EJUKEBOX errors in succession
- **Not specific to TCP, but xid generation remains an issue:**
  - Still some clients that try the random/pseudo random approach for seeding the xid
    - It just leads to premature xid re-use and bad hits in the duplicate request cache
    - Starting the xid with time of day in seconds, shifted to the left, has stood the test of time

Summary advice for NFS client developers

- 60 second+ timeouts. Good for the net, and they avoid potential problems with clients and servers
- If no timeo= option shows up on the mount command, the default value should be determined inside the kernel, not in the mount command
  - Make default timeout a tunable parameter
- NFSv3/TCP servers must not disconnect when they see retries
- Use NULL pings to probe whether connection is alive
- Aim for a retry timeout at the NFS/RPC level that is higher than TCP-level re-transmit interval
- Assume that when a connection is broken the server has no plans to respond without a retry

Advice for NFS users

- Determine what your default NFS/TCP timeout is. E.g.
  - `mount -o proto=tcp server:path /mnt`
  - Start a packet trace:
    - `tethereal -w /tmp/dump.trc -f "src server-name or dst server-name" &`
  - Force a tcp connection to be made:
    - `ls /mnt`
  - Force a timeout
    - Break network path (e.g. disconnect client from switch)
    - `ls /mnt`
  - Wait 10 minutes, kill tethereal, and examine dump.trc with ethereal
    - Look for timestamp of first NFS/RPC level retry (it will have the same xid, but a different TCP sequence number) and compare to original's timestamp
- If the timeout is under 60 seconds, consider specifying timeo=600 [600 tenths of a second] to the mount command

Pointers

- http://cvs.opensolaris.org/source/xref/on/usr/sr
- Ric Werme's XID talk
  - http://www.connectathon.org/talks96/werme1.html