Implementing a high-end NFSv4.1 service using a Java NIO framework

In 7500 lines to new RPC library
Tigran Mkrtchyan for dCache Team

EMI is partially funded by the European Commission under Grant Agreement RI-261611
dCache NFS vs. N.N

DESY GridLab:
- 50% T2 CPU
- 30% T2 Storage
(See poster 503)
The anatomy of NFS package

- NFSv41 (rfc 5661)
- RPC (rfc 1831)
- XDR (rfc 1832)
- RPCCES_GSS (rfc 2203)
A bit of ONC-RPC history

- Developed by Sun Microsystems in 1986
- First published in 1988 (as Sun RPC)
- Re-published as standard in 1995 (as ONC RPC)
- ~1600 registered services at IANA
  - NFS
  - NIS
- Widely used at HEP in 90’s
  - Control, DAQ, Monitoring, Data transfer
Today status

• Pushed back by new 'Buzz Words'
  • XML-RPC & JSON-RPC
  • SOAP & REST
• Performance still not bitten
• Google's Protobuff is real alternative
  • String type
  • Modern language friendly
  • No service version number
  • Encode/Decode only (more like XDR)
Why invent a new wheel?

- Not that many Java implementations
  - No bi-directional RPC support
  - No RPCSEC_GSS
  - Not up-to-date
- Official libtirpc not good enough
  - No bi-directional RPC
  - JAVA – C integration
Is it a square wheel?

- High performance network IO is not an RPC/NFS requirements
  - Network components from GlassFish Application Server
- RFC 1831 and RFC 2203 compliant
- IPv6 support
- GSS handling comes from Java Run-time Environment
  - jre 6 provides AES128 and AES256
  - Poll/epoll/select/p_threads handles by JVM
- We use high level abstractions
- Works on Linux, Solaris, OS X, Windows and Android
We are not doing it the typical JAVA way

- Single thread per connection
  - Thousand threads per server
- Request processed almost in a single thread
  - No thread fencing (till first shared resource)
- Simple to implement
  - Blocking reads
  - Blocking writes
  - Idle threads costs nothing (ok, 48k stack space)
RPC vs. Others

TCP

HTTP GET

- Many protocols are request-reply based
- No new requests as long as no reply
- Multiple requests processed sequentially

TCP

RPC CALL

- Possible multiple independent requests
  - Even in one TCP package
- Server may process requests out-of-order
  - Reply in asynchronous fashion
- THE way to go for some workloads
  - High latency High bandwidth NFS access
Our approach

- Poll of IO threads
  - Typically set to #Cores
- Pool of worker threads (if required)
- Processing per PRC packet
  - No binding to network connection
  - Can be used with other transport (RDMA)
- Event based
  - doOnRead if bytes arrived
  - doOnWrite if bytes sent
jRPC vs. Linux kernel

RPC requests per second

![Graph showing RPC requests per second for different number of client threads and systems (jRPC, linux-kernel, nfs-ganesha).]
jRPC vs. Linux kernel

Results are confirmed by Linux and tirpc developers
Chain of responsibilities

Call

RPC dispatcher

GSS encode/decoder

RPC validation

RPC Fragment collector/splitter

TCP

NIC/OS

Reply

RPCBIND

NFSv4
IO strategy: Same Thread

Single thread pick-ups an event and process it.

IO thread pool

NIC/OS
TCP
RPC validation
GSS encoder/decoder
RPC Fragment collector/splitter
RPC dispatcher
NFSv4
IO strategy: Worker Thread

A thread picks up an event and pushes it into the event queue.
Multi-Core

```
top - 13:39:55 up 7 days, 20:40,  3 users, load average: 8.38, 8.52, 9.27
Tasks: 279 total, 1 running, 278 sleeping, 0 stopped, 0 zombie
Cpu0 30.6%us, 18.6%sy, 0.0%ni, 45.5%id, 0.0%wa, 0.0%hi, 5.3%si, 0.0%st
Cpu1 24.7%us, 14.7%sy, 0.0%ni, 57.7%id, 0.0%wa, 0.0%hi, 3.0%si, 0.0%st
Cpu2 23.5%us, 14.2%sy, 0.0%ni, 59.6%id, 0.0%wa, 0.0%hi, 2.6%si, 0.0%st
Cpu3 24.5%us, 14.9%sy, 0.0%ni, 57.6%id, 0.0%wa, 0.0%hi, 3.0%si, 0.0%st
Cpu4 30.9%us, 20.6%sy, 0.0%ni, 43.5%id, 0.0%wa, 0.0%hi, 5.0%si, 0.0%st
Cpu5 22.9%us, 14.6%sy, 0.0%ni, 59.5%id, 0.0%wa, 0.0%hi, 3.0%si, 0.0%st
Cpu6 17.8%us, 10.9%sy, 0.0%ni, 69.3%id, 0.0%wa, 0.0%hi, 2.0%si, 0.0%st
Cpu7 25.5%us, 14.6%sy, 0.0%ni, 56.3%id, 0.0%wa, 0.0%hi, 3.6%si, 0.0%st
Cpu8 25.6%us, 20.6%sy, 0.0%ni, 49.2%id, 0.0%wa, 0.0%hi, 4.7%si, 0.0%st
Cpu9 22.8%us, 13.5%sy, 0.0%ni, 60.7%id, 0.0%wa, 0.0%hi, 3.0%si, 0.0%st
Cpu10 18.8%us, 11.6%sy, 0.0%ni, 67.7%id, 0.0%wa, 0.0%hi, 2.0%si, 0.0%st
Cpu11 18.8%us, 11.9%sy, 0.0%ni, 67.3%id, 0.0%wa, 0.0%hi, 2.0%si, 0.0%st
Cpu12 1.3%us, 4.0%sy, 0.0%ni, 0.7%id, 0.0%wa, 0.0%hi, 94.0%si, 0.0%st
Cpu13 14.2%us, 7.6%sy, 0.0%ni, 76.2%id, 0.0%wa, 0.0%hi, 2.0%si, 0.0%st
Cpu14 22.8%us, 14.9%sy, 0.0%ni, 58.9%id, 0.0%wa, 0.0%hi, 3.3%si, 0.0%st
Cpu15 21.5%us, 11.9%sy, 0.0%ni, 63.9%id, 0.0%wa, 0.0%hi, 2.6%si, 0.0%st
Mem:   66070260k total, 14979240k used, 51091020k free, 295776k buffers
Swap:  8008392k total, 0k used, 8008392k free, 13926660k cached

```

<table>
<thead>
<tr>
<th>PID</th>
<th>USER</th>
<th>PR</th>
<th>NI</th>
<th>VIRT</th>
<th>RES</th>
<th>SHR</th>
<th>S</th>
<th>%CPU</th>
<th>%MEM</th>
<th>TIME+</th>
<th>COMMAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>17425</td>
<td>root</td>
<td>16</td>
<td>0</td>
<td>16.3g</td>
<td>191m</td>
<td>9728</td>
<td>S</td>
<td>619.0</td>
<td>0.3</td>
<td>62:00:15</td>
<td>java</td>
</tr>
<tr>
<td>17618</td>
<td>root</td>
<td>15</td>
<td>0</td>
<td>6024</td>
<td>676</td>
<td>572</td>
<td>S</td>
<td>83.7</td>
<td>0.0</td>
<td>5:50:02</td>
<td>bitguard</td>
</tr>
<tr>
<td>17593</td>
<td>root</td>
<td>15</td>
<td>0</td>
<td>12892</td>
<td>1256</td>
<td>828</td>
<td>R</td>
<td>1.0</td>
<td>0.0</td>
<td>0:04:72</td>
<td>top</td>
</tr>
<tr>
<td>5463</td>
<td>root</td>
<td>18</td>
<td>0</td>
<td>21192</td>
<td>1388</td>
<td>548</td>
<td>S</td>
<td>0.3</td>
<td>0.0</td>
<td>0:09:80</td>
<td>pcsd</td>
</tr>
<tr>
<td>1</td>
<td>root</td>
<td>15</td>
<td>0</td>
<td>10368</td>
<td>684</td>
<td>572</td>
<td>S</td>
<td>0.0</td>
<td>0.0</td>
<td>0:03:27</td>
<td>init</td>
</tr>
</tbody>
</table>

```
How that looks like in the code

RpcDispatchable nfs4 = new NFSServerV41(....);
OncRpcSvc svc = new OncRpcSvcBuilder()
    .withTCP()
    .withAutoPublish()
    .withPort(2049)
    .withSameThreadIoStrategy()
    .build();
svc.register(nfs4_prot.NFS4_PROGRAM, nfs4);
svc.start();
Code re-use (and much more)

- All **Filters** have:
  - onRead to process receive
  - onWrite to process send

- **NIC/OS**
- **TCP**
- **RPC Fragment collector**
- **RPC validation**
- **GSS unpacker**

To send RPC calls

- **RPC client**
- **NFSv4 client**

To receive RPC calls

- **RPC dispatcher**
- **NFSv4 server**
Bi-directional RPC

- Client/server defined by connection initiator only.
- Any client can receive calls.
- Any server can send requests.

To send RPC calls

*NIC/OS*

*TCP*

*RPC Fragment collector*

*RPC validation*

*GSS unpacker*

To receive RPC calls

*RPC client*

*RPC dispatcher*

*NFSv4 client*

*NFSv4 server*
Security

- RPCSEC_GSS (krb5)
- Proofed to work with AD, MIT and Heimdal
- Supported Quality of protection:
  - NONE
  - INTEGRITY
  - PRIVACY
**QOP none**

Frame: 16: 354 bytes on wire (2832 bits), 354 bytes captured (2832 bits)
- Ethernet II, Src: IntelCor_a0:ca:f4 (00:1c:05:a0:ca:f4), Dst: Cisco_9f:e0:0f:00:00:00
- Internet Protocol Version 4, Src: 131.169.185.68 (131.169.185.68), Dst: 131.169.185.69
- Transmission Control Protocol, Src Port: ideafarm-panic (903), Dst Port: nfs

Remote Procedure Call, Type: Call XID: 0x2e1f3b38
- Fragment header: Last fragment, 284 bytes
  - XID: 0x2e1f3b38 (773798712)
  - Message Type: Call (0)
  - RPC Version: 2
  - Program: NFS (100003)
  - Program Version: 4
  - Procedure: COMPOUND (1)
    - [The reply to this request is in frame 20]

- Credentials
  - Flavor: RPCSEC_GSS (6)
    - Length: 36
    - GSS Version: 1
      - GSS Procedure: RPCSEC_GSS_DATA (0)
      - GSS Sequence Number: 1
      - GSS Service: rpcsec_gss_svc_none (1)

- GSS Context: <DATA>

- Verifier
  - Network File System, Ops(1): EXCHANGE_ID
    - [Program Version: 4]
      - [V4 Procedure: COMPOUND (1)]
    - Tag: <EMPTY>
    - mlnversion: 1
    - Operations (count: 1)
QOP integrity

Frame 17: 394 bytes on wire (3152 bits), 394 bytes captured (3152 bits)
Ethernet II, Src: IntelCor_a0:ca:f4 (00:1c:c0:a0:ca:f4), Dst: Cisco_9f:f0:b9
Internet Protocol Version 4, Src: 131.169.185.68 (131.169.185.68), Dst: 131.1
Transmission Control Protocol, Src Port: ggf-ncp (678), Dst Port: nfs (2049),
Remote Procedure Call, Type: Call XID:0x34e5da9f

Fragment header: Last fragment, 324 bytes
XID: 6x34e5da9f (887478943)
Message Type: Call [0]
RPC Version: 2
Program: NFS (100003)
Program Version: 4
Procedure: COMPOUND (1)
[The reply to this request is in frame 21]

Credentials
  Flavor: RPCSEC_GSS (6)
  Length: 36
  GSS Version: 1
  GSS Procedure: RPCSEC_GSS_DATA (0)
  GSS Sequence Number: 1
  GSS Service: rpcsec_gss_svc_integrity (2)
  GSS Context: <DATA>
  Verifier

Network File System
[Program Version: 4]
[V4 Procedure: COMPOUND (1)]
  GSS Data, Ops(1): EXCHANGE_ID
  GSS Checksum: 0000001c04000000f000000003991c9a0af5569f...
QOP privacy

Frame 17: 422 bytes on wire (3376 bits), 422 bytes captured (3376 bits)

Ethernet II, Src: IntelCor_a0:ca:f4 (00:1c:00:a0:ca:f4), Dst: Cisco_9f:

Internet Protocol Version 4, Src: 131.169.185.68 (131.169.185.68), Dst:

Transmission Control Protocol, Src Port: 1016 (1018), Dst Port: nfs (2049)

Remote Procedure Call, Type:Call XID:0x9100b1e2

▷ Fragment header: Last fragment, 352 bytes

  XID: 0x9100b1e2 (2438033314)
  Message Type: Call (0)
  RPC Version: 2
  Program: NFS (100009)
  Program Version: 4
  Procedure: COMPOUND (1)

  [The reply to this request is in frame 21]

▷ Credentials

  Flavor: RPCSEC_GSS (6)
  Length: 36
  GSS Version: 1
  GSS Procedure: RPCSEC_GSS_DATA (6)
  GSS Sequence Number: 1
  GSS Service: rpcsec_gss_svc_privacy (3)

  [GSS Context: <DATA>]

▷ Verifier

Network File System

  [Program Version: 4]
  [V4 Procedure: COMPOUND (1)]

  GSS Data: <DATA>
SUMMARY

- High performance RPC library
- Compatible with existing standards
- Meets today's requirements
  - IPv6, AES256
- In production since 2009 (dCache-1.9.5)
Ready to use by others

- Spitted into an independent library
- Licensed with LGPLv2
- Hosted on http://code.google.com/p/nio-jrpc/
- Maven repo.
- Already used in third party products
  - BACnet
  - One of the Swiss banks