Data Access

Gregory Touretsky, Spring 2015
Information provided in these slides is for educational purposes only
Outline

• Introduction
  • Main challenge – avoiding DDoS

• Continuous Integration (CI) use-case
  • Definition, workflow, three main steps

• Challenges and solutions
  1. Handling multiple concurrent writes (software build step)
  2. Handling mass of reads for multiple datasets (small-scale regression step)
  3. Enabling remote data access (full-scale regression step)
Introduction
Sample use-cases

High Performance Computing: use of many computing resources over periods of time to accomplish a computational task
Main challenge: Avoiding DDoS

- 5,000 clients accessing data
- Every client has 1Gbps network
- Is DoS inevitable?
Approach to solution?
Know your workload....

- Permanent access vs. sporadic?
- Read only vs. read/write?
- Sequential vs. random?
- Data layout
  - Small files / large files
  - Deep / flat directory structure
File Access Slowness

- Black box testing by several clients
- Longer response time triggers automatic data collection, analysis and action
- Minimize Cost of Monitoring
Continuous Integration (CI) use cases
Definition

• Continuous integration (CI) is the practice, in software engineering, of merging all developer working copies with a shared mainline several times a day (Wikipedia)
Workflow

Central code repository

Continuous Integration system

Build system

DorA Test sub-system

Test system

1. Check-out

2. Check-in

3. Reject

4. Test

5. Accept

3. Build

6. Once a week – large scale regression

Goal: ensure always alive code. Individual changes don’t break compilation or basic tests

Many tests? Remember the scheduler...

Challenges in Modern Data Centers Management, Spring 2015
CI: Three main steps

1. Software build (read/write, 100’s jobs)
   • Run in a single data center

2. Multiple small-scale regression (read-mostly, 1000’s of jobs each)
   • Run in single data center
   • Multiple datasets – one per-user (lot’s of data)

3. Full-scale regression (read-mostly, 100K+ jobs)
   • Runs in multiple data centers (lots of jobs...)
   • Single dataset (data not too large)

• In the coming slides we cover each step, data-access challenges it introduces, and possible solutions
Challenges and solutions
Step #1: Software build

• Compile modified sources in the user’s private NFS work area
• Numerous source files
• Translates to 100s of compilation jobs
• Mix of small reads, **multiple writes** and metadata calls
  • Non-shared (private) compilation logs
  • Shared objects
Step #1: Software build

- Compile modified sources in the user’s private NFS work area
- Numerous source files
- Translates to 100s of compilation jobs
- Mix of small reads, multiple writes and metadata calls
- Non-shared (private) compilation logs

Read/Write, many files
Multiply by dozens of users
Challenge #1: Handling multiple concurrent writes
Solutions

1. Write logs on local disks
2. Write shared objects to Scale-out storage
Scale-out storage

Scale-up (traditional):

Scale-out:

Storage Controller

Storage Controller

Storage Controller

Scale Performance (IOPS, Bandwidth)
Scale Capacity
Single namespace
Characteristics

• Hardware & software vs. software-only
• Split data/meta-data nodes vs. distributed data/meta-data
• Client-side SW vs. standard protocols
• Strong vs Eventual consistency
Sample implementation: Isilon

Source: http://www.emc.com

Challenges in Modern Data Centers Management, Spring 2015

Page 22
Sample implementation: Lustre

Source: http://wiki.lustre.org/
Parallel NFS (NFS 4.1)

- Client reads/writes a file
- Server grants permission
- File layout (stripe map) is given to the client
- Client parallel R/W directly to data server

- No single storage node is a bottleneck
- Improves large file performance
- Single namespace

Source: [http://www.esg-global.com/blogs/thoughts-on-pnfs/](http://www.esg-global.com/blogs/thoughts-on-pnfs/)
Handling multiple concurrent writes: Summary

• Know your workload
• Behavior monitoring and automatic response
• Multiple solutions

<table>
<thead>
<tr>
<th>Solution</th>
<th>Preferred access pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local disks</td>
<td>• Applicable for temporary data</td>
</tr>
<tr>
<td>Scale-out storage</td>
<td>• Read/Write, mixed file size</td>
</tr>
<tr>
<td></td>
<td>• May vary per implementation</td>
</tr>
<tr>
<td></td>
<td>• Single Point Of Failure risk (SW-wise)</td>
</tr>
</tbody>
</table>
Step #2: Multiple small-scale regressions

- Run **limited amount** of tests against newly compiled code in the user’s private NFS area
  - Many users in parallel (many datasets – total size of data is very large)
- Few binaries to read
- Translates to 1000s of test jobs
  - Read compiled binary
  - Execute test
- Write short log file - care about log **only** if test has failed
Step #2: Multiple small-scale regressions

- Run limited amount of tests against new compiled code in the user’s private NFS work area prior to final acceptance into central repository
- Few binaries to read
- Translates to 1000s of test jobs
  - Read compiled binary
  - Execute test
  - Write short log file (care about log only if test has failed)
- Many users (datasets) in parallel
- So total size of data is very large

Black: Bytes to NFS
Red: Bytes from NFS

Challenges in Modern Data Centers Management, Spring 2015

ONC-RPC Service Response Time statistics for NFS version 3: inf091_run_smart 1.16 trc

<table>
<thead>
<tr>
<th>Index</th>
<th>Procedure</th>
<th>Cells</th>
<th>Min SRT</th>
<th>Max SRT</th>
<th>Avg SRT</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>READ</td>
<td>748</td>
<td>0.000018</td>
<td>0.01702</td>
<td>0.000450</td>
</tr>
<tr>
<td>3</td>
<td>LOOKUP</td>
<td>359</td>
<td>0.000094</td>
<td>0.002431</td>
<td>0.000205</td>
</tr>
<tr>
<td>4</td>
<td>ACCESS</td>
<td>144</td>
<td>0.000106</td>
<td>0.000498</td>
<td>0.000419</td>
</tr>
<tr>
<td>5</td>
<td>READLINK</td>
<td>86</td>
<td>0.0000224</td>
<td>0.000537</td>
<td>0.000315</td>
</tr>
<tr>
<td>1</td>
<td>GETATTR</td>
<td>1</td>
<td>0.0000160</td>
<td>0.000160</td>
<td>0.000150</td>
</tr>
</tbody>
</table>
Challenge #2: Handling mass of reads
Solutions

1. Logs on local disks – (as before, to handle the writes)
2. Scale-out storage – (as before, to handle the writes)
3. Client-side caching (CacheFS, CaMA) – new
Client-side caching: cacheFS

- mount –t nfs –o fsc,rsize=32768,... filer:/filesystem /mnt/dir
- Cache miss = data transfer from the file server
Client-side caching: Performance trade-offs

• File lookup time
  • Increased – check cache first
  • Decreased – metadata may be cached locally

• File read time
  • Increased – check cache first
  • Increased – copy network data to cache
  • Decreased – cache quicker to read

• File write time
  • Decreased – lazy writeback; cache faster
  • Neutral – can write to disk and network simultaneously

• File replacement time
  • Increased – have to delete from cache
Client-side caching: CaMa

- API
  - Register a dataset
  - Query RLS for location
- Peer-to-peer distribution
- Read-only
- Global access statistics
- Cache management – ttl/disk space/etc
- Integration with batch scheduler
Connecting CaMA with the job scheduler

• Concurrent starts vs. starts over time → impact on storage, with/without Cama
• Concurrent copies limits from CS/FS
• Caching levels
• Prefer pre-populated cache
CaMa in action

Fileserver read latency

Hot Filer threshold
Handling mass of reads: Summary

- Know your workload
- Behavior monitoring and automatic response
- Multiple solutions

<table>
<thead>
<tr>
<th>Solution</th>
<th>Preferred access pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>CacheFS</td>
<td><strong>Read-mostly</strong>, many accesses from <strong>few</strong> clients</td>
</tr>
<tr>
<td>CaMa</td>
<td><strong>Read-only</strong>, many accesses from <strong>many</strong> clients</td>
</tr>
</tbody>
</table>
Step #3: Full-scale regression

- Run an “unlimited” amount of tests (jobs) against a chosen dataset
  - Get the widest test coverage possible
  - Single dataset – total size of data is not too large
- Few binaries to read
- Translates to 100,000s of test jobs (needs multiple data centers to execute)
  - read compiled binary
  - execute test
- write short log file - care about log only if test has failed
**Step #3: Full-scale regression**

- Run an "unlimited" amount of tests (jobs) against a chosen dataset.
- Get the widest test coverage possible.
- Single dataset — total size of data is not too large.
- Few binaries to read.
- Translates to 100,000s of test jobs (needs multiple data centers to execute).

**Challenges in Modern Data Centers Management, Spring 2015**

<table>
<thead>
<tr>
<th>ONC-RPC Service Response Time statistics for NFS version 3: infs091_run_smb4.1.16.trc</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
</tr>
<tr>
<td>-------</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

Black: Bytes to NFS
Red: Bytes from NFS
Challenge #3: Enabling remote data access

- Can we just use remote compute?
- What challenges would you expect?

Data Center 1

Wide Area Network

Data Center 2

- Additional 5,000 clients
- 5,000 miles away

Remember the [global] scheduler...
Solutions

1. Logs on local disks – (as before, to handle the writes)
2. Scale-out storage – (as before, to handle the writes)
3. Client-side caching (CacheFS, CaMA) – (as before, to handle the reads)
4. Leverage global batch capacity (RM lecture part III)
   • Remote execution – replication and site-level caching
Bandwidth

- Network bandwidth: Throughput of a logical or physical communication path in a digital communication system

<table>
<thead>
<tr>
<th>Sample connection bandwidth</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>56 kbit/sec</td>
<td>Modem/Dialup</td>
</tr>
<tr>
<td>1.5 Mbit/sec</td>
<td>ADSL</td>
</tr>
<tr>
<td>2.048 Mbit/sec</td>
<td>E1</td>
</tr>
<tr>
<td>10 Mbit/sec</td>
<td>Ethernet</td>
</tr>
<tr>
<td>11 Mbit/sec</td>
<td>Wireless 802.11b</td>
</tr>
<tr>
<td>54 Mbit/sec</td>
<td>Wireless 802.11g</td>
</tr>
<tr>
<td>100 Mbit/sec</td>
<td>Fast Ethernet</td>
</tr>
<tr>
<td>600 Mbit/sec</td>
<td>Wireless 802.11n</td>
</tr>
<tr>
<td>1 Gbit/sec</td>
<td>Gigabit Ethernet</td>
</tr>
<tr>
<td>10 Gbit/sec</td>
<td>10 Gigabit Ethernet</td>
</tr>
</tbody>
</table>
Latency

• **Round-trip time (RTT)** is the length of time it takes for a signal to be sent plus the length of time it takes for an acknowledgment of that signal to be received.

```sh
# ping local-host
64 bytes from local-host: icmp_seq=1 ttl=64 time=4.30 ms

# ping remote-host
64 bytes from remote-host: icmp_seq=1 ttl=52 time=196 ms
```
Latency impact

LAN: RTT = 0.15 ms
Rsize=32k

WAN: RTT = 211 ms
Rsize=32k

# time
cp /mnt/A/100MBfile /dev/null
0:46.58

# time
cp /mnt/A/100MBfile /dev/null
0:00.99
Protocol optimization for data transfers

• Underlying transport protocol: UDP vs. TCP
• Multi-stream transfer
• WAN optimization
TCP: Transmission Control Protocol

• TCP: reliable data transfer protocol
  • SEND – ACK

• Congestion window: max # of unacknowledged packets a source can send
  • TCP buffers

Challenges in Modern Data Centers Management, Spring 2015
TCP window tuning

• Goal: eliminate “idle time” waiting for ACK

• Optimal TCP window size = Bandwidth-in-bits-per-second * Round-trip-latency-in-seconds
  • The bigger the window, the faster?

• Bigger window:
  • More buffer memory on the server
  • Longer retransmit time
Example

• Haifa – Amsterdam link: 10 Mbps; 60ms RTT
• What is the optimal TCP window size to get maximum possible throughput?

\[
10 \text{ Mbps} \times 0.06 \text{ s} = 0.6 \text{ Mbit} = 76.8 \text{ KB} \rightarrow 64\text{KB}
\]

\[
\frac{64 \text{ KB}}{0.06 \text{ s}} = 1.04 \text{ MB/sec} = 8.3\text{Mb/s} \rightarrow 83\% \text{ of WAN BW}
\]
UDP: User Datagram Protocol

- Connectionless protocol
  - Unreliable
  - Not ordered
  - Lightweight
  - Datagrams
  - No congestion control
WAN optimization

- Deduplication
- Compression
- Latency optimization
Throughput comparison

• WAN link: Israel – California (RTT = 200 ms)
• File size: 137 MB
Remote data access: replication

- All data is replicated = equal storage capacity needed
- Synchronous replication
  - WAN is a bottleneck
  - No data consistency for DR
- Asynchronous replication
  - Update frequency
  - Inband / out-of band updates
Remote data access: client-side caching

• Similar to LAN
• Many clients = many transfers
Remote data access: site-level proxy caching

- Data transferred on-demand = reducing demand for WAN, storage
Caching challenges: data consistency vs. performance impact

- Attribute caching timeout
- Delegations
Write through vs. Write back

Source Filer

Create file1

Modify file1

Cache Filer

Modify file1 (write through)

ACK

Modify file1

Modify file1 (write back)

ACK

Slow write, no data corruption

Fast write, potential data corruption

Maximum write-back delay
WAN topology

- Distribution optimization
Enabling remote data access: Summary

- Know your workload, and network
- Optimize your transport protocol
- Multiple solutions

<table>
<thead>
<tr>
<th>Solution</th>
<th>Preferred usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synchronous replication</td>
<td>Disaster recovery for critical applications</td>
</tr>
<tr>
<td></td>
<td>Active-Active. High demand for WAN</td>
</tr>
<tr>
<td>Asynchronous replication</td>
<td>Disaster recovery; remote usage – all data is required</td>
</tr>
<tr>
<td>Client-side caching</td>
<td>Optimize reads; support few remote clients per location</td>
</tr>
<tr>
<td>Site-level cache</td>
<td>Optimize reads and some writes; consistency vs. performance; subset of data is</td>
</tr>
<tr>
<td></td>
<td>required remotely</td>
</tr>
</tbody>
</table>