Byte Range Delegations

Trond Myklebust
Network Appliance

NetApp®

Outline

- **Motivation**
  - Why would we want to strengthen caching in NFS?
- **Proposal for a model for caching**
- **Random thoughts**

An example customer

From: Nick I <clusterbuilder@gmail.com>
To: nfs@lists.sourceforge.net
Subject: [NFS] backbone of a large-file streaming system
Date: Tue, 8 Nov 2005 10:29:15 -0700 (09:29 PST)

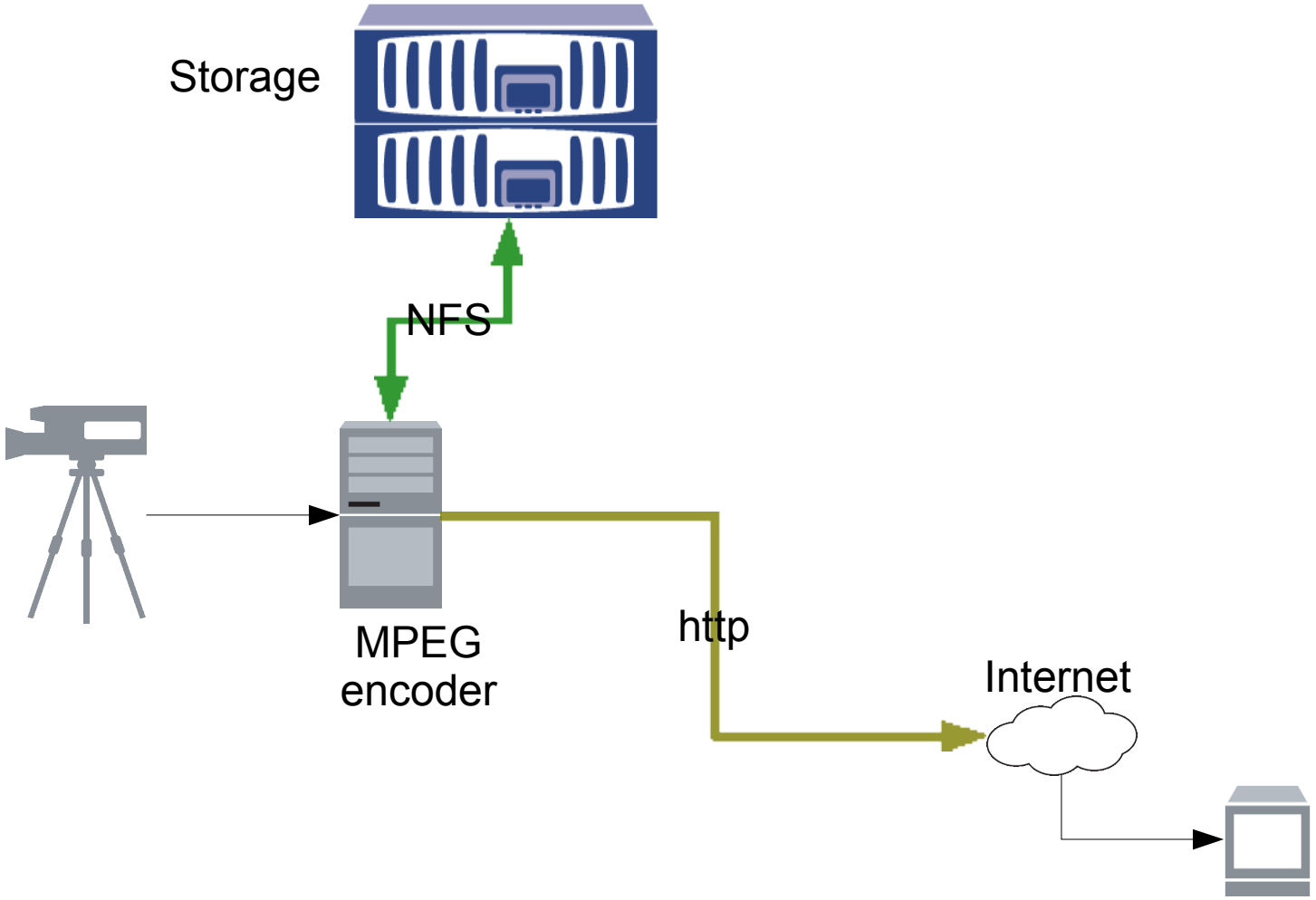
Hi,
I help maintain a Web site at www.clusterbuilder.org. You might have seen before that I have a section called Ask the Cluster Expert, where I am building a knowledgebase of cluster and grid information. When someone asks a question I am researching the answer to build this knowledgebase out. I received the following question:

"I am looking for a variety of solutions to be a backbone of a large-file streaming system providing thousands of concurrent download streams. Preferably commodity hardware and Linux, though I'm open to commercial solutions."

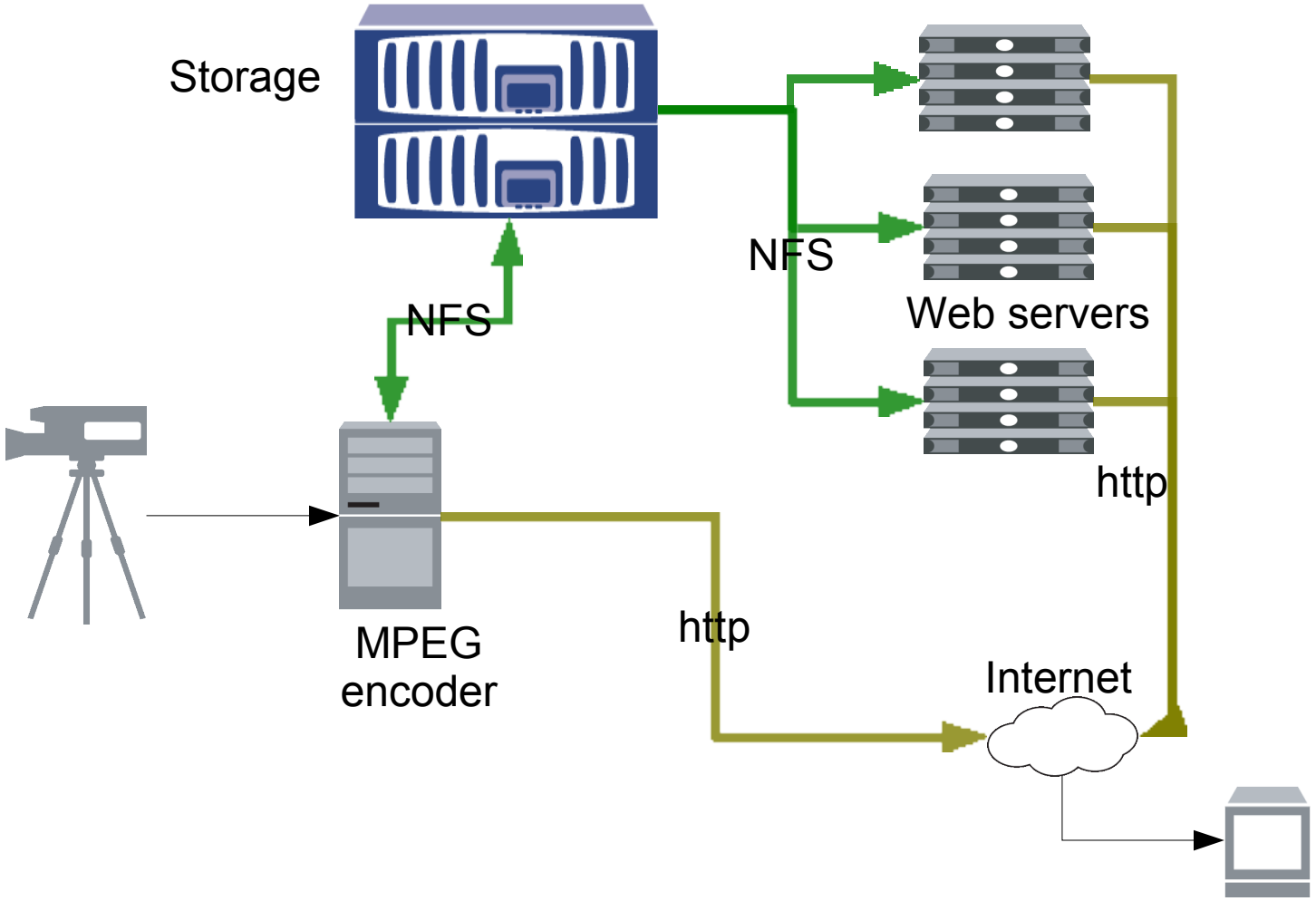
I am wondering if anyone here has suggestions of what applications will work best for this type of setup. You can respond to the question at www.clusterbuilder.org/FAQor respond in email.

Thanks,
Nick

An example: streaming multimedia



An example: streaming multimedia



Did this help?

Caching works fine for static files

For live multimedia

- **No...**

The problem

- **Traditional NFSv4 caching cannot work**
  - **Readers + writers means no delegations**
  - **Change attribute is global to the file**
    - **So append means it is constantly changing**

NetApp®

What about locking?

- Doesn't help either: read locks ensure that nothing changes while the locks are held, but once the lock is released, the guarantees are lost.
- Mandatory write locks can help protect against write ordering issues.

Other solutions

Write lots of little WORM files

- Yes, that would work
- Basically the route taken by applications like 'mh'

But

- Requires application support on both the writer + reader

An alternative...

Teach NFSv4 to cache partial files

An alternative...

Teach NFSv4 to cache partial files

Instead of full file delegations, allow the client to notify the server that it wishes to cache part of a file

An alternative...

- Teach NFSv4 to cache partial files
- Instead of full file delegations, allow the client to notify the server that it wishes to cache part of a file
- Allow the server to notify the client that the cache needs invalidating
  - Same thing we already do for file delegations

Byte range delegations

- Extension to NFSv4 that allows client to negotiate intent to cache with server
  - Can declare intent to cache a region of the file for reads
  - Can declare to cache writeback data for a region of the file
  - Described in http://www.ietf.org/internet-drafts/draft-myklebust-nfsv4-byte-range-delegations-00.txt
- Server can notify client when cache needs to be flushed by means of a callback
  - Like delegations, the callback initiates a limited time period during which client MUST flush cache, then declare it is done
- Allows for strong cache consistency

Some further enhancements

Instead of requiring client to flush writes, then clear cache, server has the option of allowing the client to downgrade its cache from a write cache to a read cache.

Support for readahead-type requests which do not trigger a recall of the write delegation.

Transparent to byte range locks

- but do enable caching of locks, like file delegations do.

Application to the streaming multimedia case

After opening the file, all clients issue a DELEG_OPEN request in order to obtain a byte range delegation stateid.

When the MPEG encoder has new data to write, it may choose to cache the write.

- If so, it should issue a DELEG_RANGE request for that byte range with the WRITEW_LT flag set.
- If granted, then any attempt to READ or WRITE in the area covered by the delegation will trigger a callback to the MPEG encoder, with a request to flush its cache.
- After flushing, the MPEG encoder is free to continue caching the data for reading.

What about the readers?

The http servers may either

- **call DELEG_RANGE with the READW_LT flag set for the byte range to cache**
- **or (better still) prepend any READ request with the DELEG_PUT_STATEID operation.**
- **If someone decides to write data to this area, the clients can expect to be called back and told to invalidate the cached data.**

Server responsibilities

- The server needs to track the byte range delegations held by each client
  - Not slavishly, though.
    - If circumstances permit it, the server may grant a larger range than the client requested. (Cut down on number of requests, and allows server to reduce "fragmentation").
- Server also helps resolve conflicts
  - Fair queueing
- Fencing of write requests that refer to old state.
- Enforcing "lease" aspect of delegation.

Random thoughts and speculation

- Eisler's point that for write caching, the lease timeout may be an obstacle (due to poor scheduling performance on client).
  - Address perhaps by converting “delegation” into “lock with notification” if the client has the file open for write. Leave it as a lease if the file is closed.
- Scalability improvements
  - Perhaps allow server to skip notifying those clients who have file closed, but are caching read data.
    - Requires a mechanism for revalidating delegation stateid after OPEN (just do DELEG_PUT_STATEID?)

Questions?

Storage Simplified