JPGs and 3GPs and AMRs Oh My!

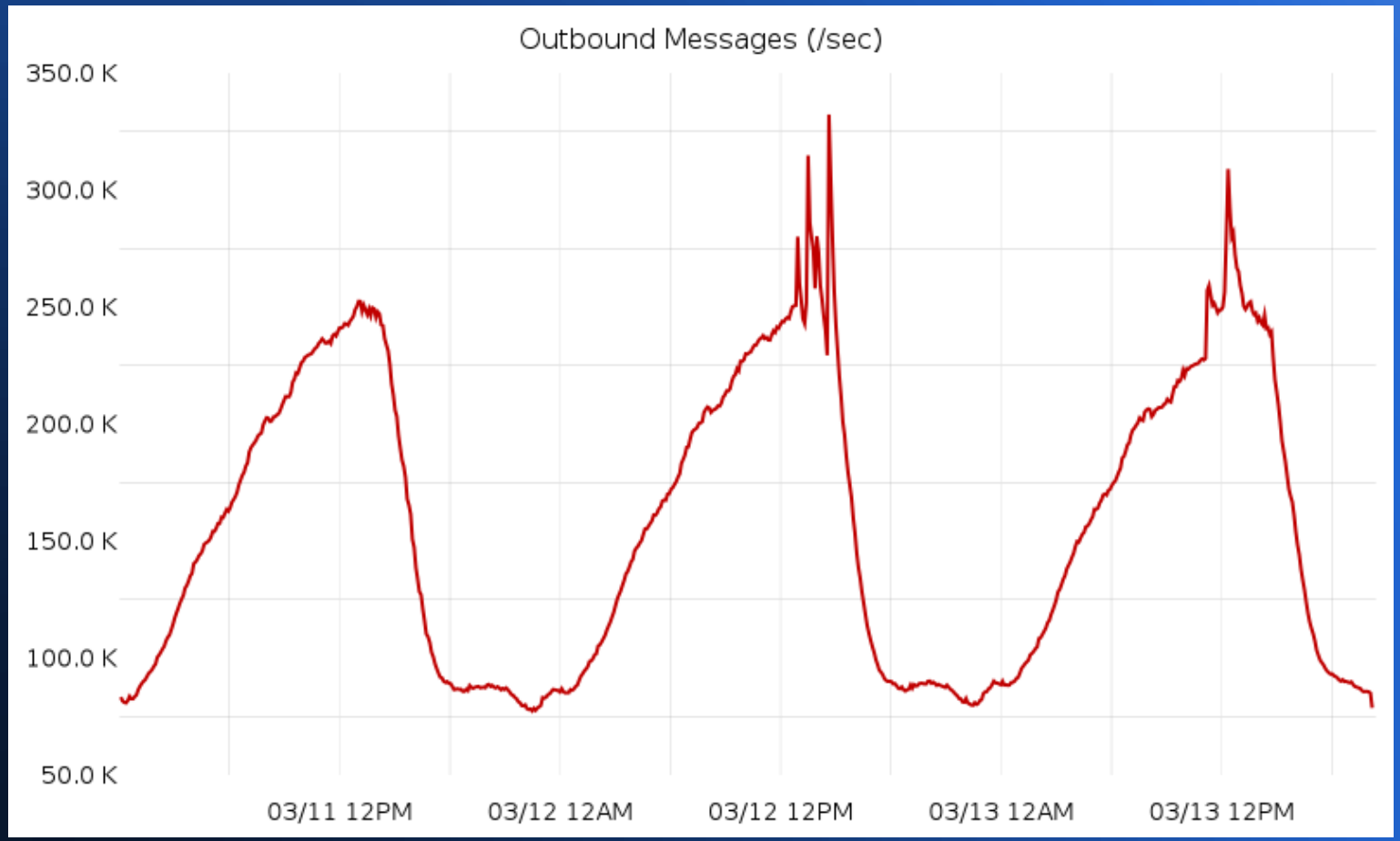
Rick Reed
WhatsApp

Erlang Factory SF
March 21, 2013

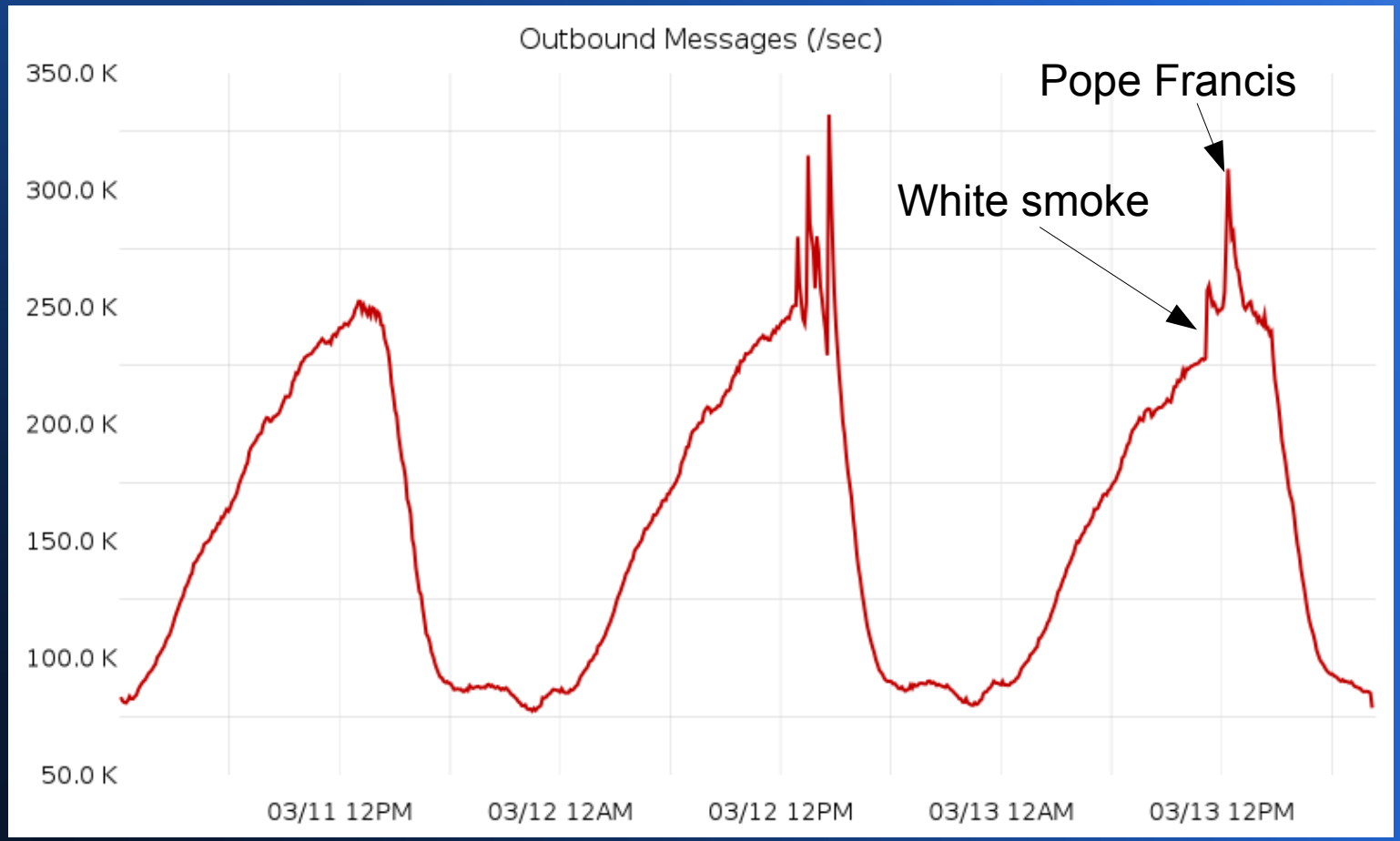
About me ...

- Joined server team at WhatsApp in 2011
- No prior Erlang experience
- Focus on systems scalability

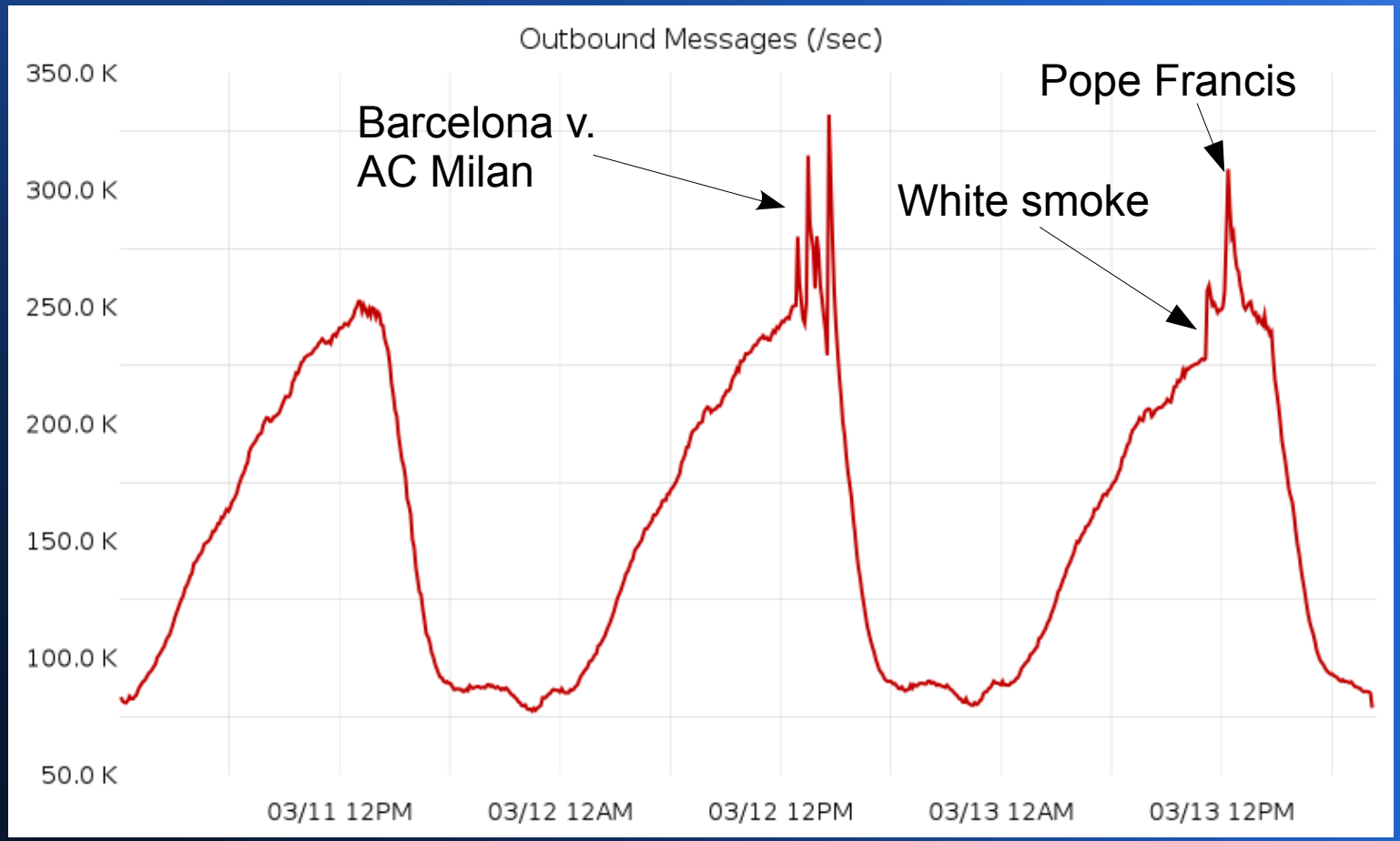
Religious Moments



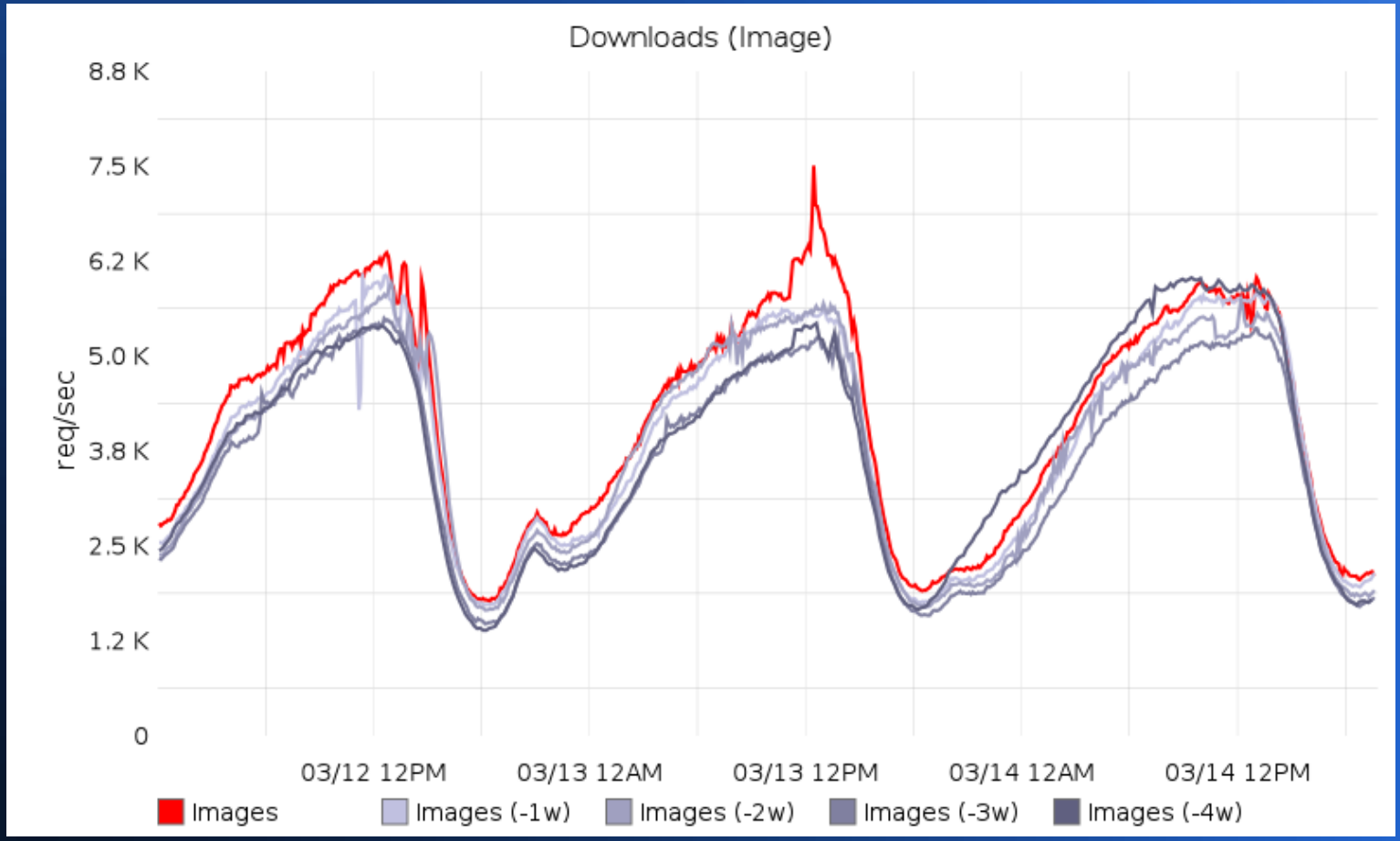
Religious Moments



Religious Moments



Pictures too ...



Overview

- The multimedia problem
- Legacy implementation
- New architecture
- Challenges and workarounds
- Results and conclusions

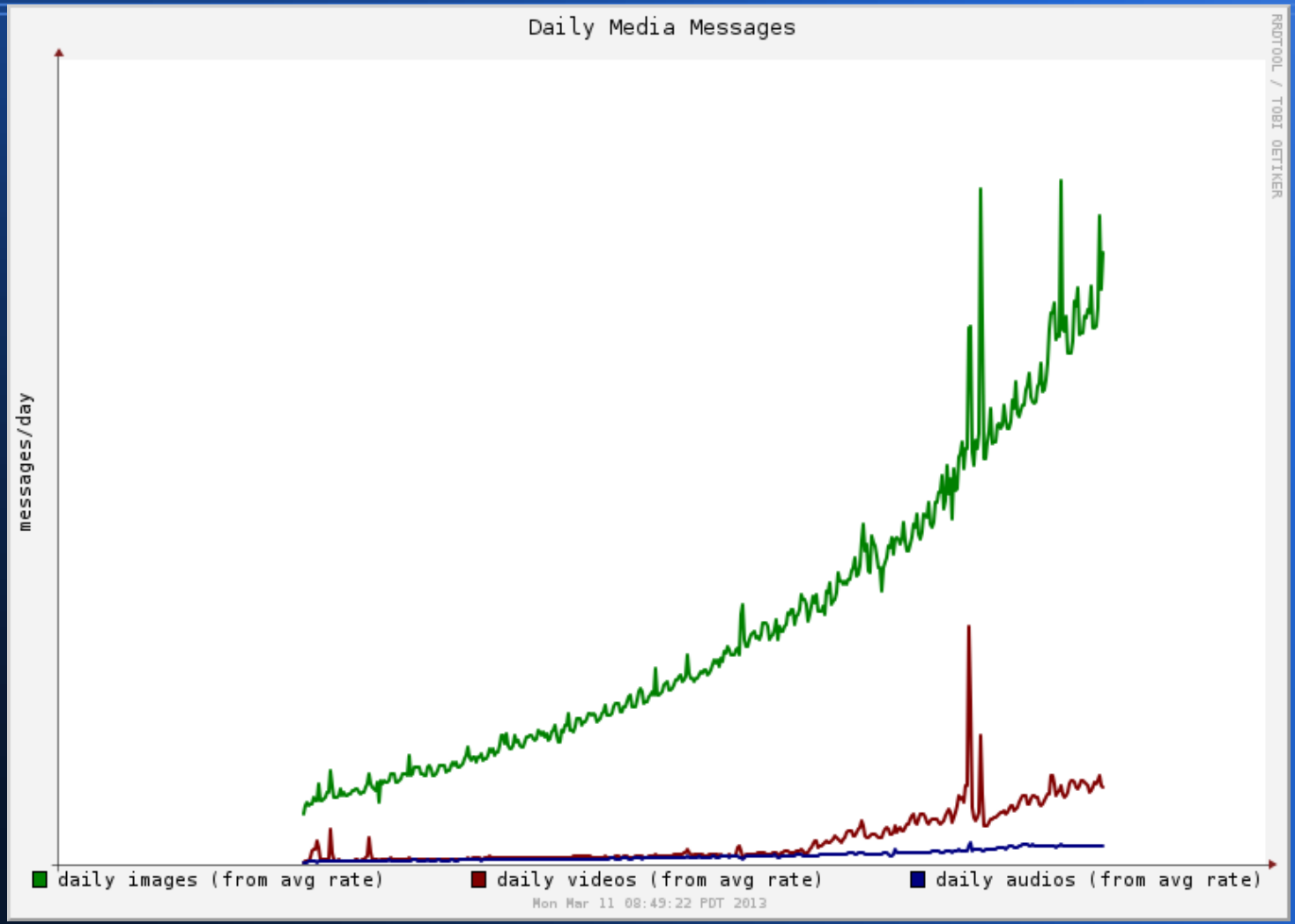
The Problem

- Multimedia messages (MMS)
  - Image
  - Video
  - Audio
- Some recorded on sender's phone, many not
- Group messaging
- Multi-platform support (transcoding)
- Store-and-forward, no archiving

The Problem

- A good (and fun) problem to have:
  - More users
  - More usage per user
  - More multimedia as a % of usage per user

The Problem


Daily Media Messages
messages/day
RRDTOOL / TOBI OETIKER
daily images (from avg rate)
daily videos (from avg rate)
daily audios (from avg rate)
Mon Mar 11 08:49:22 PDT 2013

The Problem

- Legacy system issues
  - Scalability
  - Ad-hoc transcoding
  - Not Erlang

Goals

- Scalability
- Reliability
- Improved user experience

Legacy MMS Implementation

- Lighttpd + PHP
- Dual hexcore with 12 x SATA JBOD
- DNS round-robin
- No reference counting
- Time-based media expiration
- Client-initiated (server-hosted) transcoding

New MMS Architecture

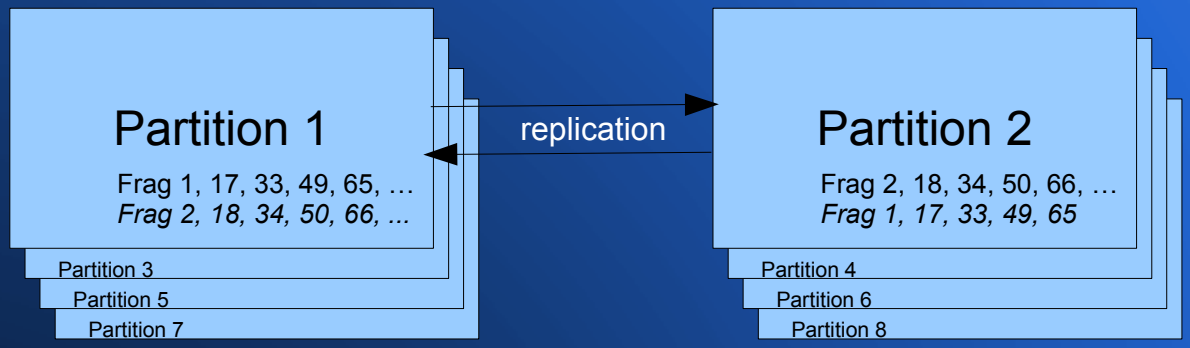
- New features
  - Resumable uploads and downloads
  - Reference counting
  - Upload de-dup
  - Server-controlled transcoding
  - Server-side “trimming”

New MMS Architecture

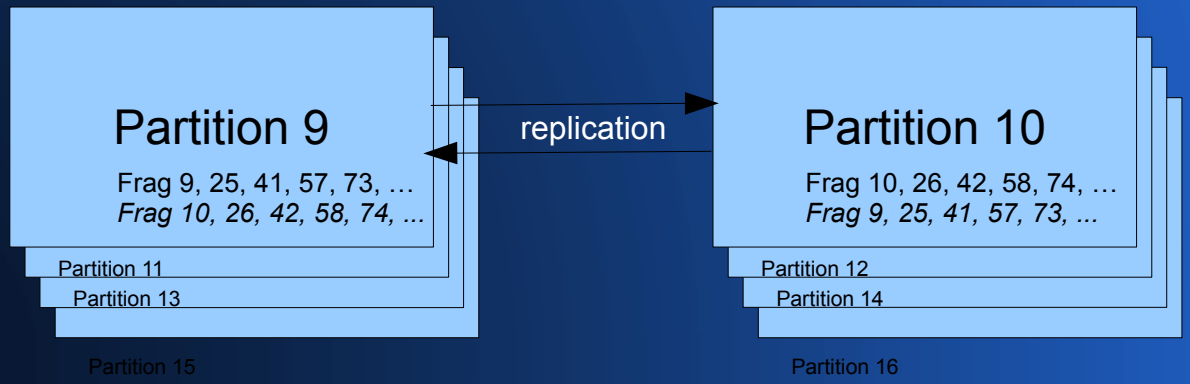
- New database
  - Objects, References, Transcodings
- mnesia
  - disc_copies tables
  - Partitioned islands and fragmented tables
  - All operations run async_dirty
  - Use key hashing to collapse all ops per key to a single process

New MMS Architecture

Mnesia Island 1



Mnesia Island 2



New MMS Architecture

- Integration with Erlang messaging cluster
  - Upload de-dup
  - Upload load balancing
  - Reference management (create, ack)

New MMS Architecture

- HTTP upload/download service
  - Preserves some commonality w/ legacy system
  - Good protocol support
    - Content-type
    - Ranges
  - Some drawbacks
    - SSL negotiation delay
    - Support lacking on some client platforms

New MMS Architecture

- Web server: YAWS
  - Nice balance of support for
    - Serving media files
    - Programmability
  - Only handful of patches for our environment
  - Runs embedded alongside other server procs

New MMS Architecture

- Object storage
  - Simple file-per-object (FreeBSD UFS2)
    - JBOD (directly attached to motherboard)
      - Image: 6xSSD
      - Audio/Video: 6xSATA
    - Hashed directory tree (< 1k files in leaf dir)
    - 16k blocksize
  - Same storage used for transient message store
    - Lots of experience (~4B cycles/day)
    - Long-term predictability

New MMS Architecture

- Media identification
  - erl_img triage (NIF)
  - MediaInfo (fork)
  - ffprobe (fork)
- Transcoding
  - ffmpeg (fork)
- Custom clone of os:cmd to bypass shell

New MMS Architecture

- Proxy misdirected requests
  - We hand out specific hostnames
  - Objects may get archived or moved
  - Acts as reverse proxy to host with content
  - yaws_revproxy
    - Somewhat difficult to set args correctly
    - Otherwise, works great

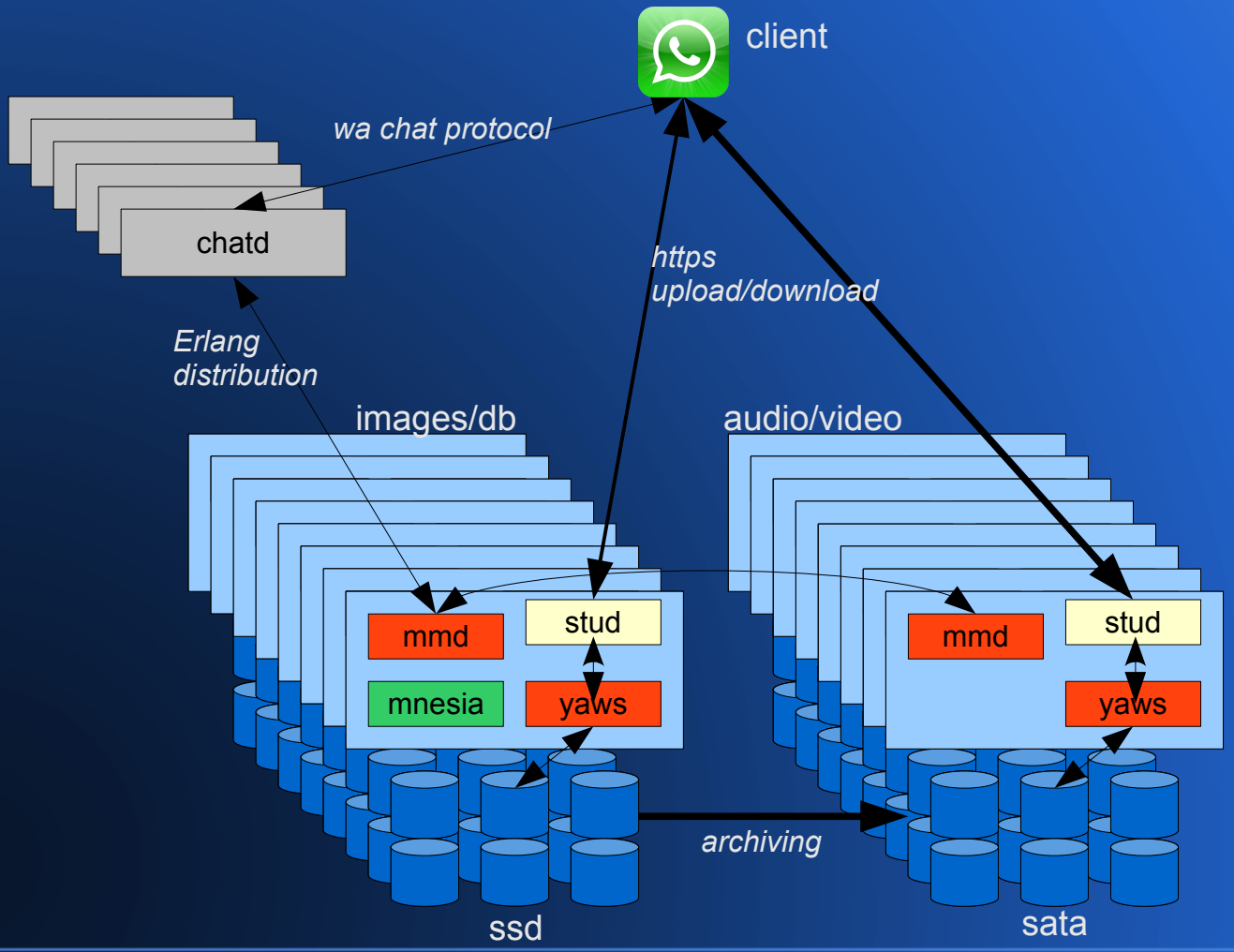
New MMS Architecture

- Maintenance processes
  - Reaper (drop ack'ed and old references)
  - Reclaim (drop unreferenced objects w/ delay)
  - Archiver (move old images to slow storage)
  - Clean (remove stranded files)

New MMS Architecture

- Hardware Specs
  - Dual octo-core E5-2690 (32 logical CPUs)
  - 256GB RAM (128GB for A/V hosts)
  - 6 x 800GB SSD (4TB SATA for A/V hosts)
  - 2 x dual link-agg gig-E (public, private)

New MMS Architecture



Challenges and Workarounds

- SSL
  - Connection bottleneck (throughput < RSA rate)
  - Offloaded SSL termination to stud
  - Patched YAWS to accept HaProxy-style header
  - Request rate now limited by RSA rate
  - Need multiple loopback aliases for >64k ports

Challenges and Workarounds

- sendfile
  - sendfile & async threads don't mix
    - On FreeBSD, at least
    - Long BEAM stalls
    - Tried both file:sendfile and YAWS driver
  - Disabled sendfile in YAWS
  - Run +A 1024
  - Plenty of memory bandwidth on our hosts

Challenges and Workarounds

- Mnesia table sizes
  - ~250M objects, ~750M references per host
  - Limits to how much RAM we can stuff in a host
  - Moved from naïve/native to packed format
    - Packed record fields into 60-bit integers
    - Packed {bin1, bin2} key into <<bin1, bin2>>
    - encrypt(filehash) <=> id instead of hash() => id
  - Record storage size reduced ~45%

Challenges and Workarounds

- Each host:

Active Tables	Local Copy Type	Records	Bytes
mmd_obj2(128)	unknown	242,325,152	46,859,527,336
mmd_reclaim	disc_copies	6,301,921	930,213,336
mmd_ref3(128)	unknown	759,076,825	138,261,758,224
mmd_upload	disc_copies	2,069,134	413,827,448
mmd_xcode2(128)	unknown	8,477,025	2,648,887,144
schema	disc_copies	387	444,384
Total		1,018,250,444	189,114,657,872

Challenges and Workarounds

- Slow reference checking
  - Ordered set keyed by Id & Ref concatenated
  - Naïve: length(get_refs(Id)) == 0
  - Fast:

```
case mnesia:next(Tab, Id) of
      <<Id:?ID_LEN/binary, _RefId:?REF_LEN/binary>> → true
      _ → false
end
```

Challenges and Workarounds

- mnesia:select by background procs
  - Returned lists of >>1M entries
  - Originally put in scratch ets table for iteration
  - Select with continuation much better

Challenges and Workarounds

- Db migrations
  - Converting record formats while online
  - Lazy migration:
    - Read: read(new), if missing read(old)
    - Write: write(new), delete(old)
    - Delete: delete(new), delete(old)
  - No transactions
    - Hash key to specific process on specific host
  - Final batch conversion, then normal behavior

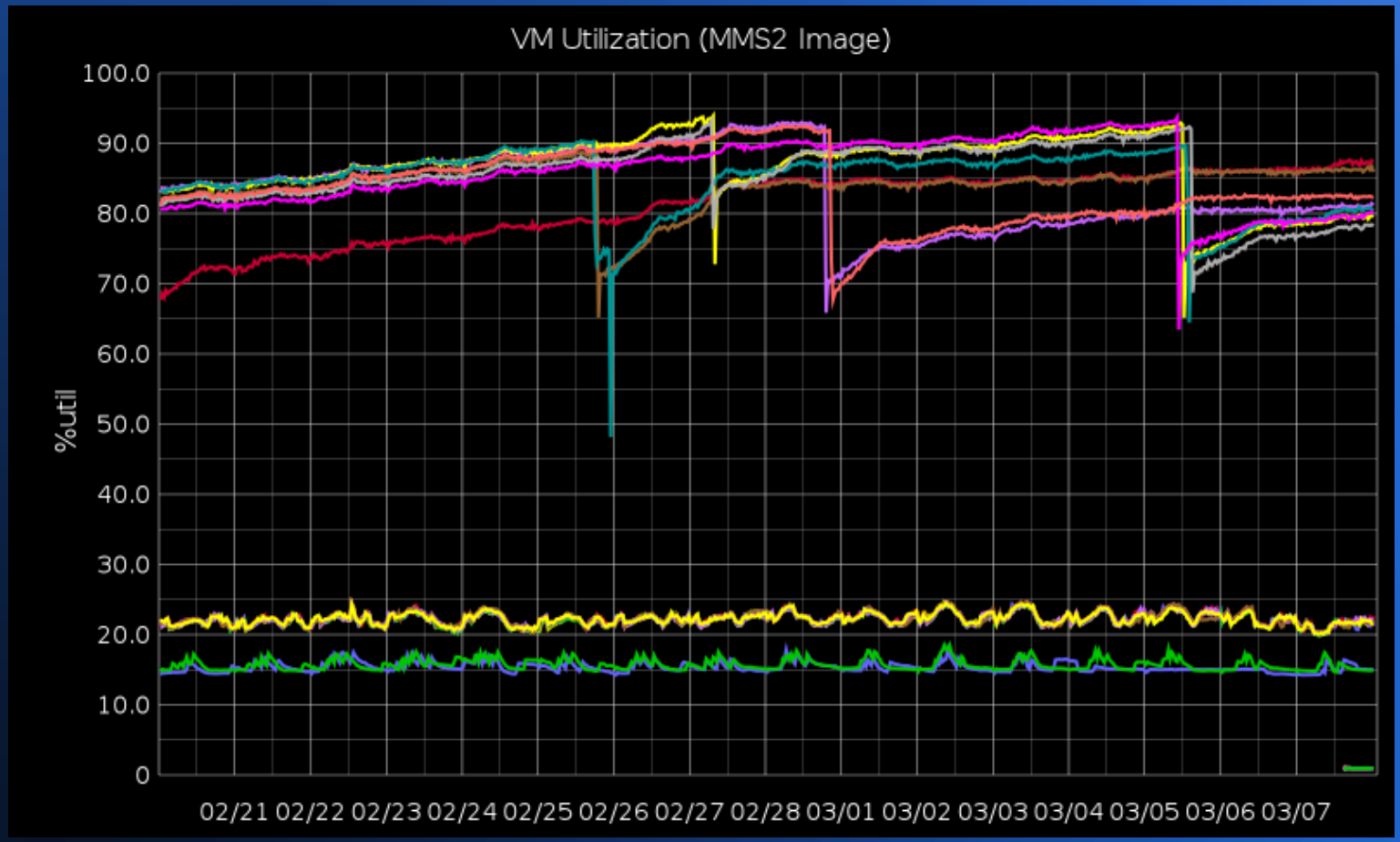
Challenges and Workarounds

- Bandwidth
  - Users hungry to consume and share media
  - Single host easily fills 2x1g aggregated uplink
  - Dealing with soaring bandwidth bill
    - Cap and manage various media parameters (video dimensions/bitrate/framerate, audio sampling rate/bitrate, etc.)
    - Count downloads and force down-conversion

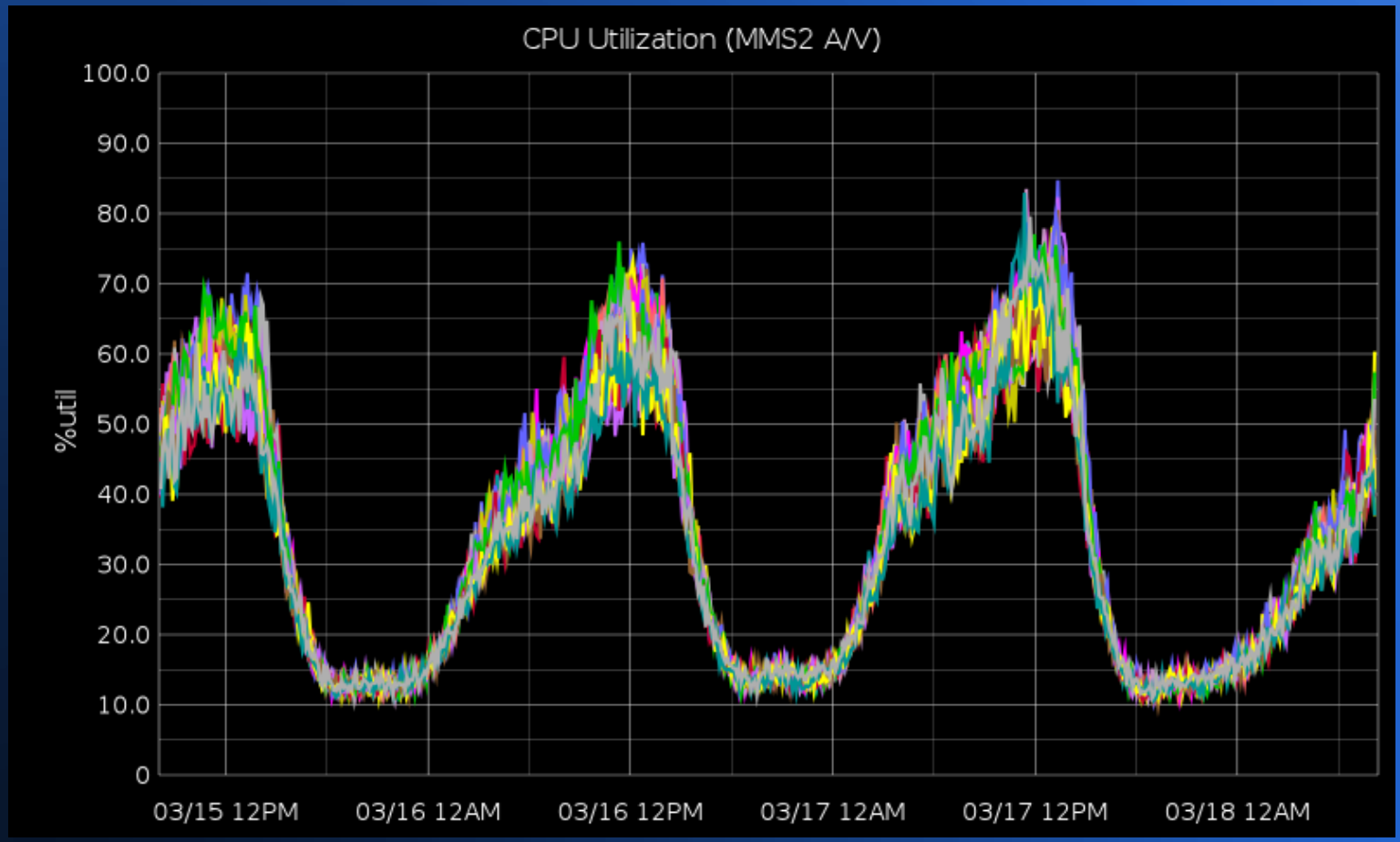
Challenges and Workarounds

- Lingering issues:
  - Storage redundancy
  - Memory leakage on db hosts (fragmentation?)
  - mnesia schema ops under load
  - Tweaking transcodings for playback issues
  - Managing transcode CPU
  - Capacity planning/management

Challenges and Workarounds



Challenges and Workarounds

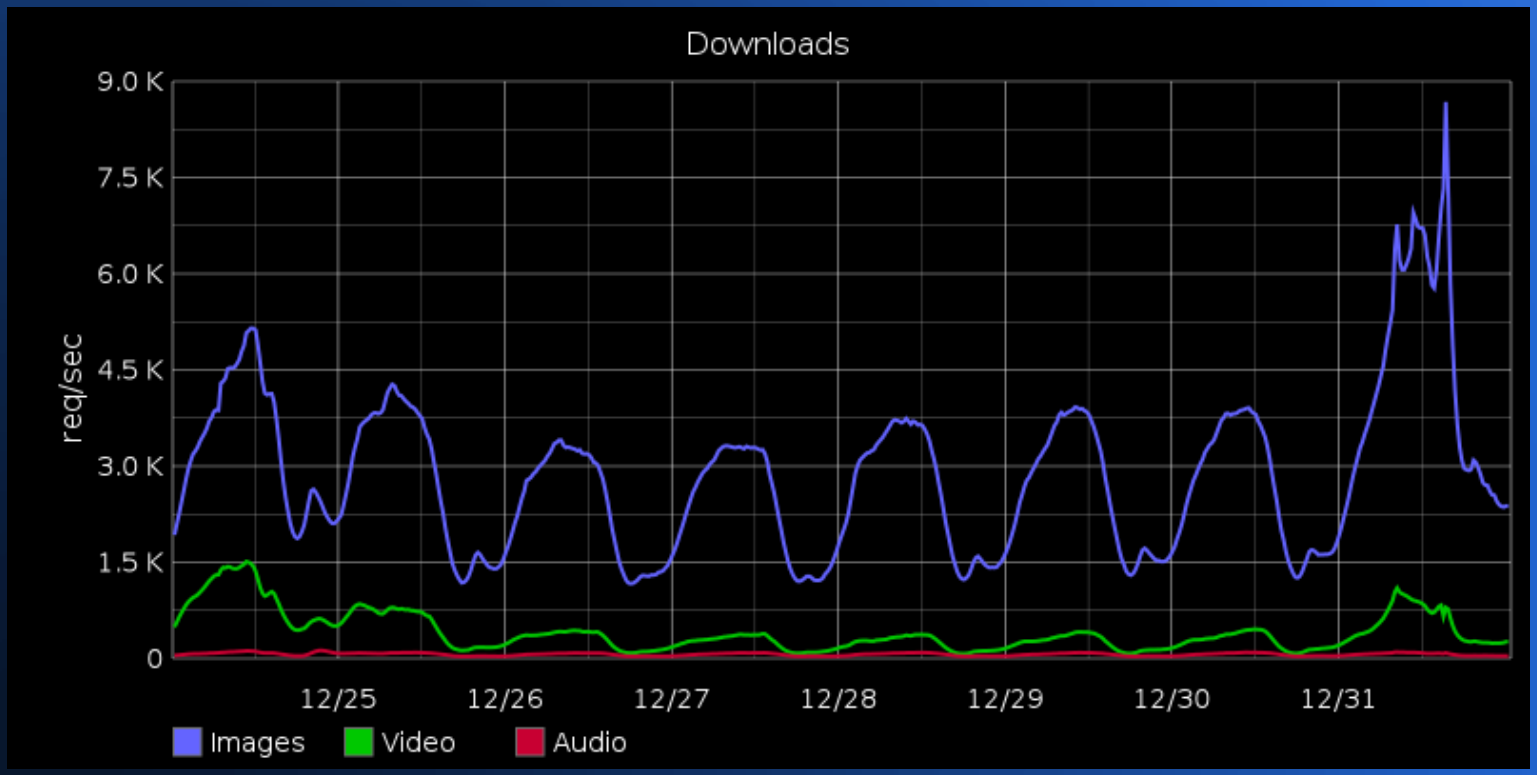


Results

- Peak Scalability
  - 214M images in a day
  - 8.8K images/sec downloaded
  - 29 Gb/sec output bandwidth

Results

- Holiday week



Results

- Erlang just fine for pushing (lots of) bytes
- But not good at everything

Results

- Example: Great for transcode configs
  - Various transcoding tweaks require code
  - Ability to deploy changes quickly
  - Raises value of server-initiated transcodes

Results

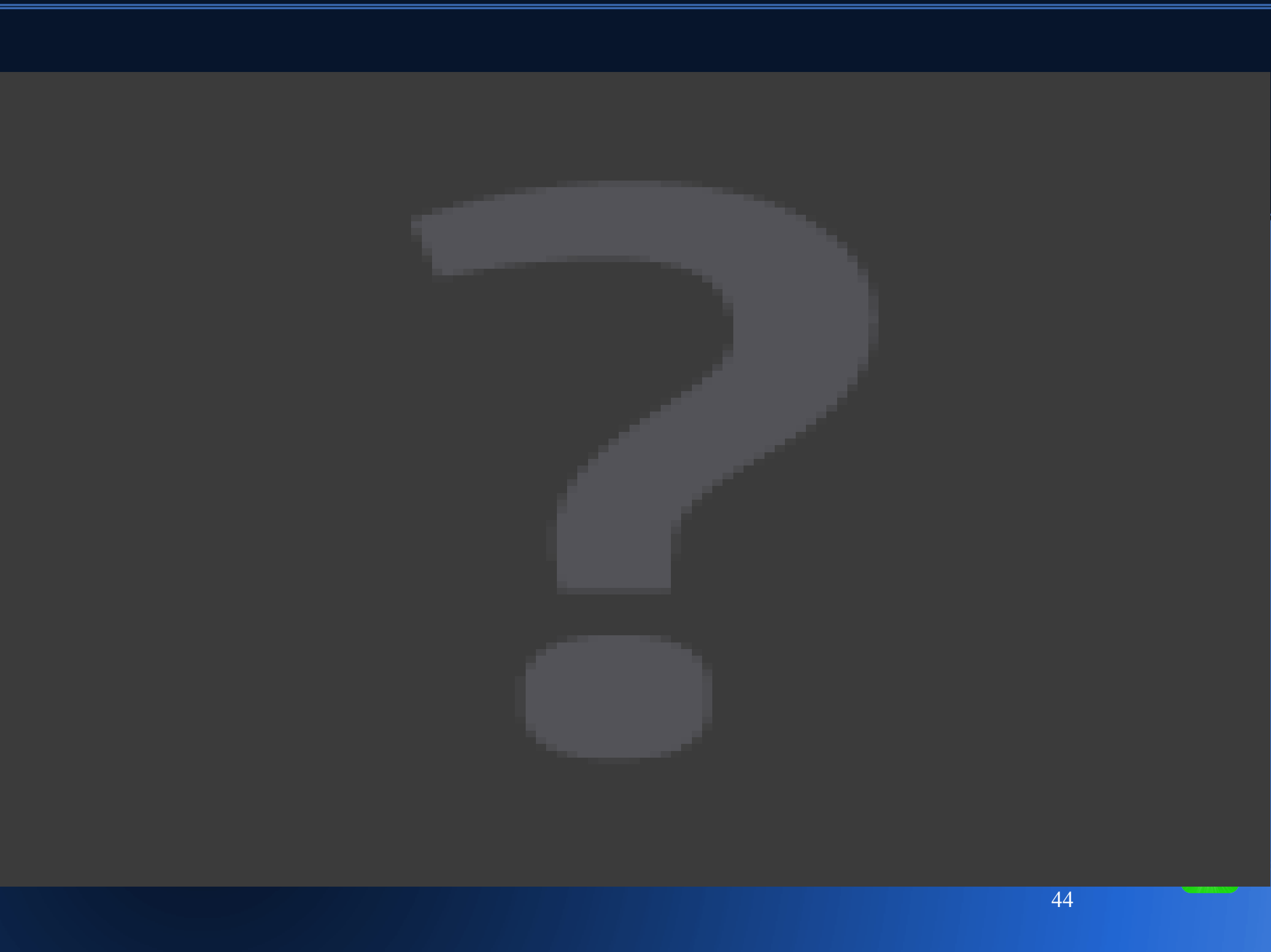
```
get_download_defaults (#mmd_client{os = android}, Type, video) →
    #mmd_parms{type=opt_default(Type, [video_mp4, video_3gp]),
               vcodec=[h264, mpeg4, h263],
               width=720, height=480,
               vbitrate=?VBITRATE_MAX,
               acodec=[aac, amrnb]};

get_download_defaults (#mmd_client{os = iphone, device = Device}, Type, video) →
    Parms = #mmd_parms{type=opt_default(Type, [video_mp4, video_quicktime]),
                       vcodec=[h264, mpeg4],
                       width=480, height=360, fps=15,
                       vbitrate=?VBITRATE_MAX,
                       acodec=aac},
    if Device =:= "iPhone_5";
       Device =:= "iPhone_4S";
       Device =:= "iPhone_4";
       Device =:= "iPhone_4_VZW" →
           Parms#mmd_parms{width=1280, height=720, fps=30};
       true →
           Parms
    end;
```

Results

```
  THR PRI NICE    SIZE    RES STATE   C    TIME   WCPU COMMAND
 1061  52    0  4314M  2953M uwait   3  77.9H 120.21% [beam.smp]
    1  22    0   140M 62820K kqread 14  83.7H  4.20% /usr/local/bin/stud -u
    1  22    0   136M 59184K kqread  6  77.0H  3.66% /usr/local/bin/stud -u
    1  23    0   116M 57704K CPU5    5  76.8H  6.49% /usr/local/bin/stud -u
    1  23    0   112M 57080K CPU14   5  82.9H  5.57% /usr/local/bin/stud -u
    1  22    0   120M 55512K kqread  3  78.8H  4.69% /usr/local/bin/stud -u
    1  22    0   132M 54524K kqread  1  78.0H  4.69% /usr/local/bin/stud -u
    1  22    0   128M 54444K kqread  2  69.4H  5.47% /usr/local/bin/stud -u
    1  22    0   116M 53472K kqread  2  74.5H  4.98% /usr/local/bin/stud -u
    1  23    0   120M 52772K kqread  1  74.6H  4.79% /usr/local/bin/stud -u
    1  23    0   112M 50192K kqread 13  66.2H  4.69% /usr/local/bin/stud -u
    1  23    0   108M 49688K kqread  7  74.3H  5.47% /usr/local/bin/stud -u
    1  21    0   108M 47420K kqread  1  66.7H  3.47% /usr/local/bin/stud -u
    1  22    0   120M 47288K kqread  1  54.8H  3.37% /usr/local/bin/stud -u
    1  23    0   100M 45512K kqread  5  55.9H  5.18% /usr/local/bin/stud -u
    1  21    0   112M 44636K kqread  2  56.0H  2.49% /usr/local/bin/stud -u
    1  97   10   107M 41956K CPU18  18   0:06 47.27% [ffmpeg]
    1  96   10   107M 41884K CPU16  16   0:05 44.19% [ffmpeg]
    1  96   10   107M 41844K CPU29  10   0:06 48.58% [ffmpeg]
    1  95   10   107M 41836K CPU22  22   0:05 42.29% [ffmpeg]
    1  93   10   103M 41276K CPU13  12   0:02 37.35% [ffmpeg]
    1  20    0   144M 31496K kqread  6 208:44  0.00% /usr/local/bin/stud -u
    1  96   10 92884K 29688K CPU28  28   0:07 48.88% [ffmpeg]
    1  89   10 92884K 29496K CPU26  26   0:01 23.68% [ffmpeg]
    1  96   10 92884K 28484K CPU8    8   0:06 47.56% [ffmpeg]
    1  97   10 88788K 26048K CPU25  25   0:06 47.56% [ffmpeg]
    1  89   10 88788K 25000K CPU21  21   0:00  0.00% [ffmpeg]
    1  95   10 88788K 24540K CPU23  23   0:06 46.19% [ffmpeg]
    1  96   10 84692K 22996K CPU24  24   0:06 47.56% [ffmpeg]
    1  92   10 84692K 22020K CPU11  26   0:02 34.28% [ffmpeg]
    1  89   10 84692K 21980K CPU31  31   0:00  0.00% [ffmpeg]
    1  91   10 84692K 20580K CPU30  30   0:01 31.69% [ffmpeg]
    1  91   10 84692K 19880K CPU27  27   0:01 32.18% [ffmpeg]
```

Questions?

- rr@ whatsapp.com



Challenges and Workarounds