Effectively using Amazon Web Services

Hobin Yoon, Jim Donahue,
Ada Gavrilovska, Karsten Schwan

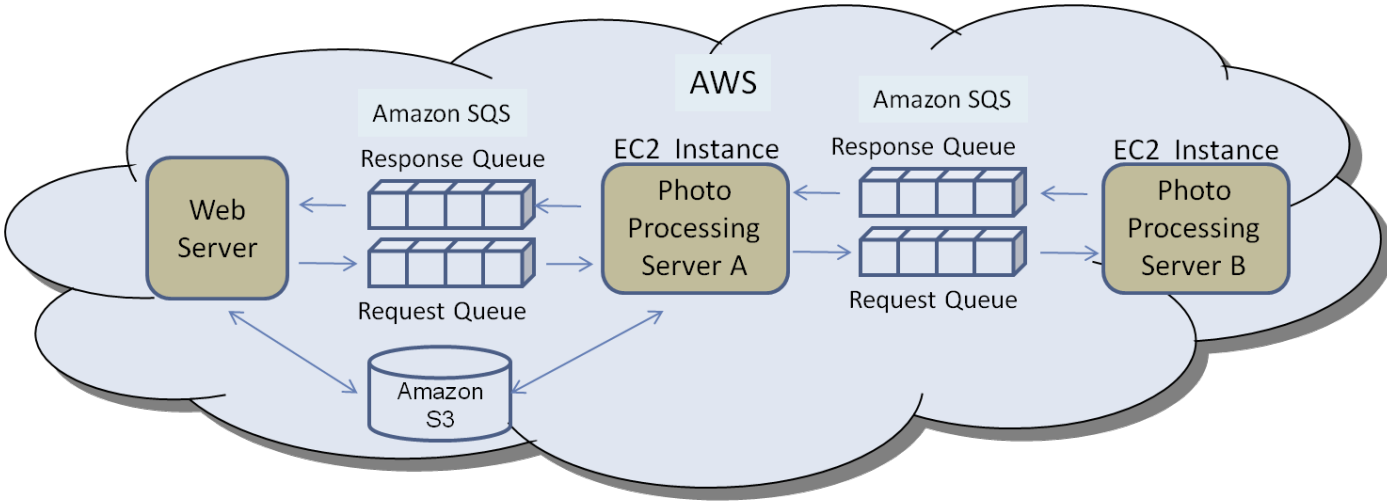
CERCS, Georgia Tech
ATL, Adobe Systems

Message latency of SQS (Simple Queue Service)

Optimizing upload performance to S3

Simple Queue Service

- Highly reliable and scalable distributed queue system for web service applications.
- A buffer between the component sending data and the component receiving the data for processing.



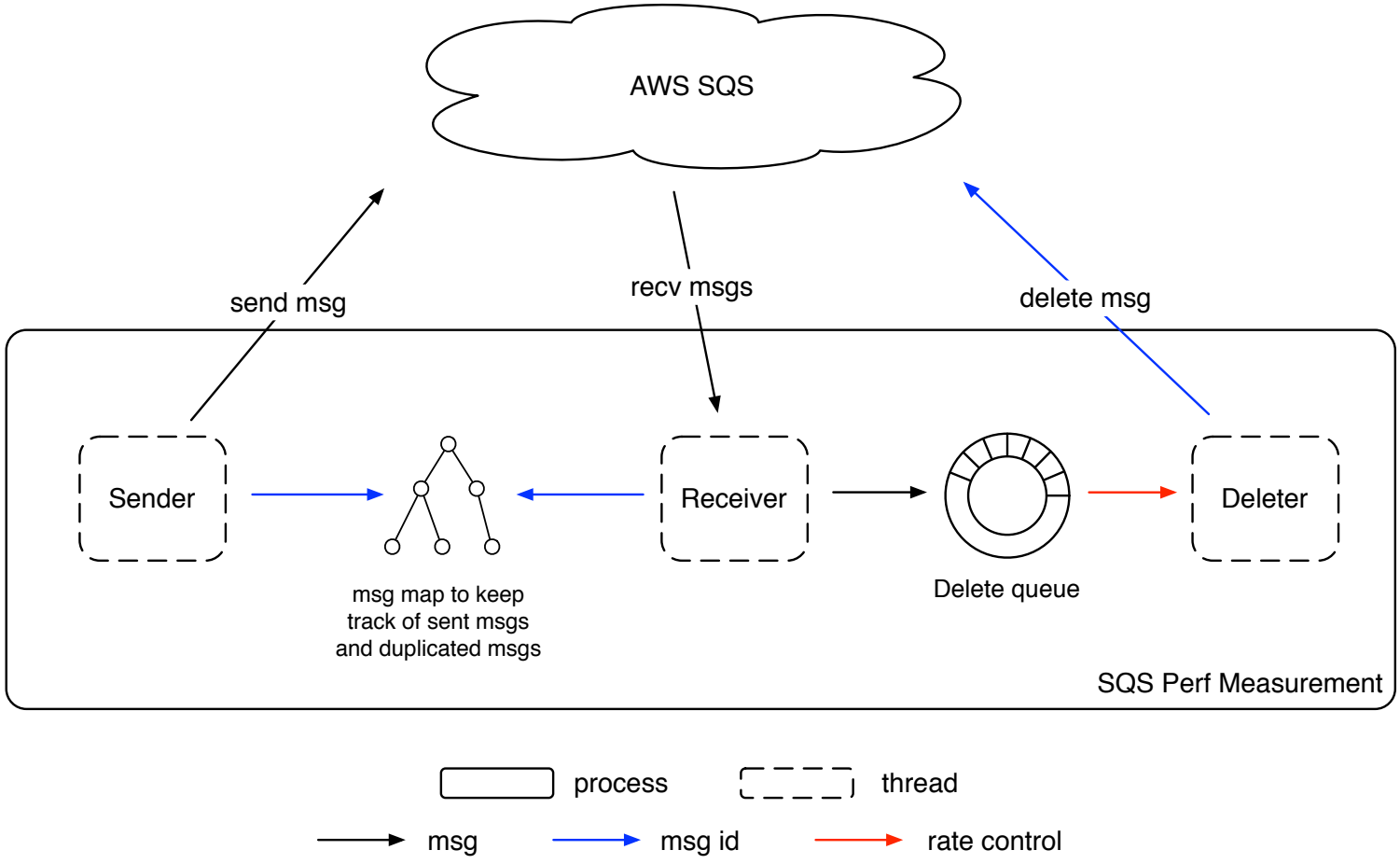
From "Building Scalable, Reliable Amazon EC2 Applications with Amazon SQS."

Good for interactive applications?

- Vague fears about potential very high send-to-receive latency.
  - Many concerns on Amazon SQS forum with latencies more than 60 seconds.
- Let's do some measurement!

First measurement - As Fast As Possible

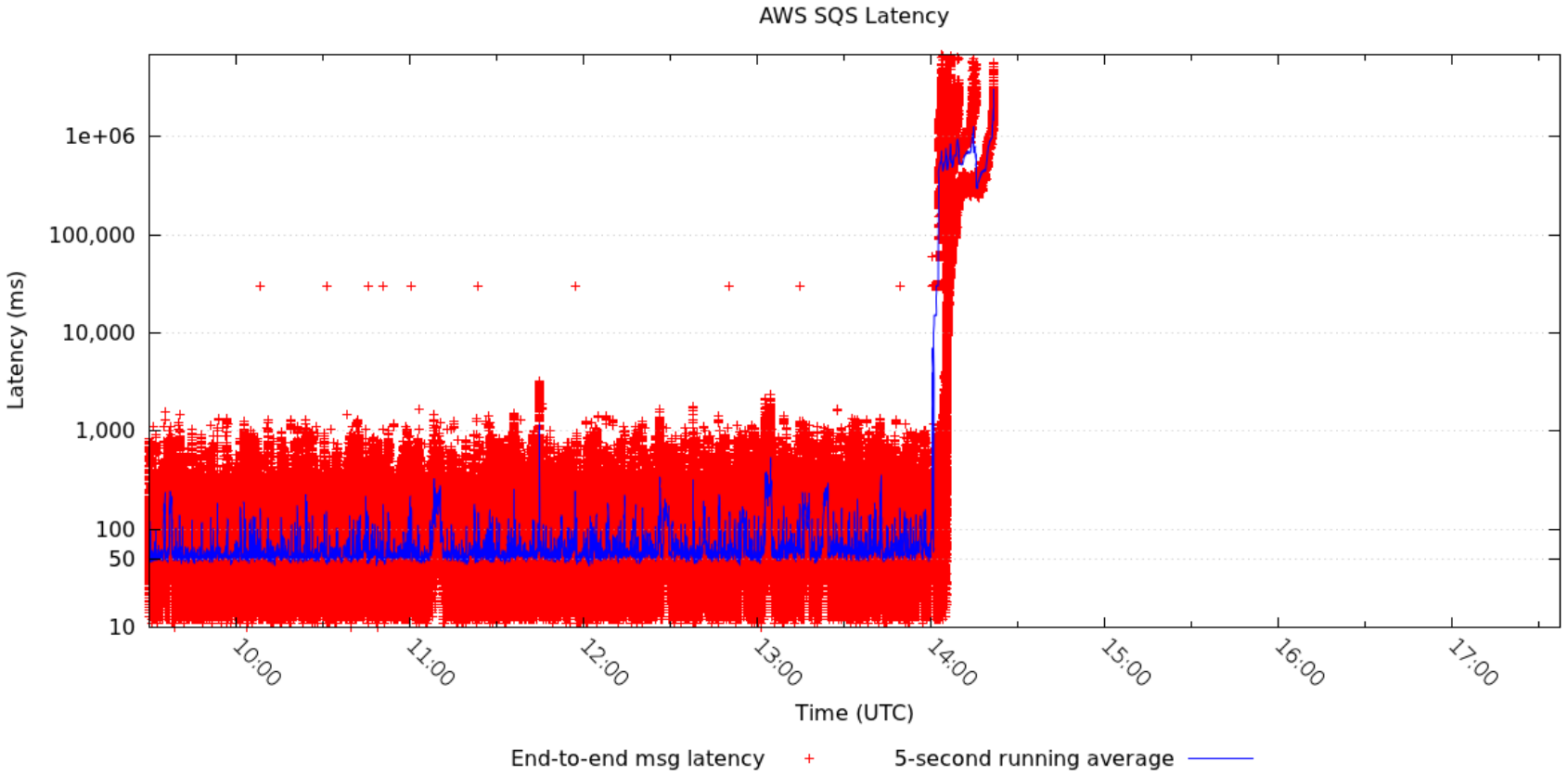
- Send/receive/delete messages as fast as possible.



SQS performance measurement architecture

Huge latencies… really?

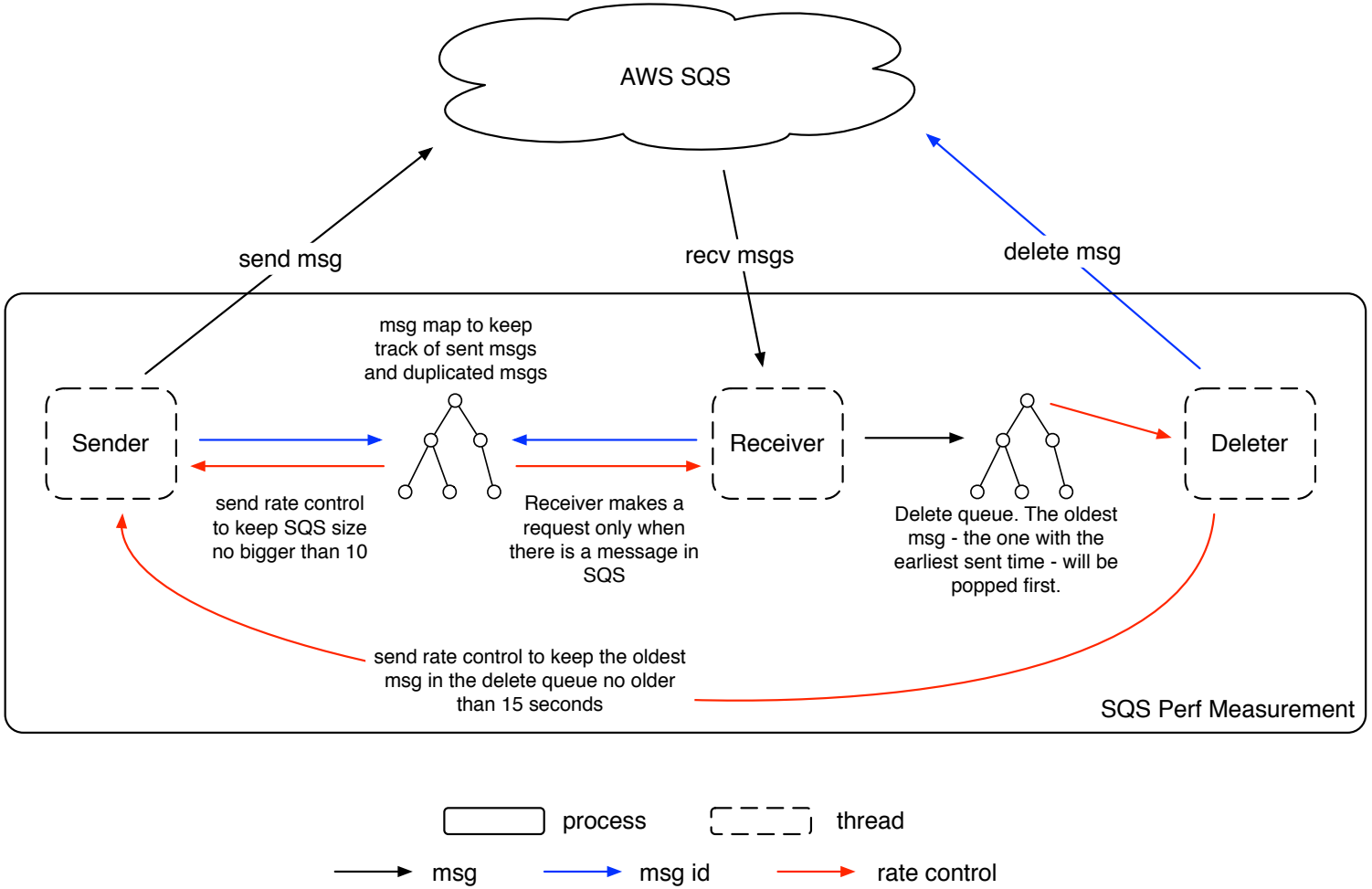
- 30 seconds of sporadic latencies
- After 5 hours, latencies increased sharply and the measurement client became unresponsive…



After inspecting logs, it turned out…

- Lots of duplicated messages.
  - Due to deleter being slower than receiver, messages kept reappearing after the 30-sec invisibility time window.
- Delete queue became too big and consumed all RAM.
- Need Rate control!

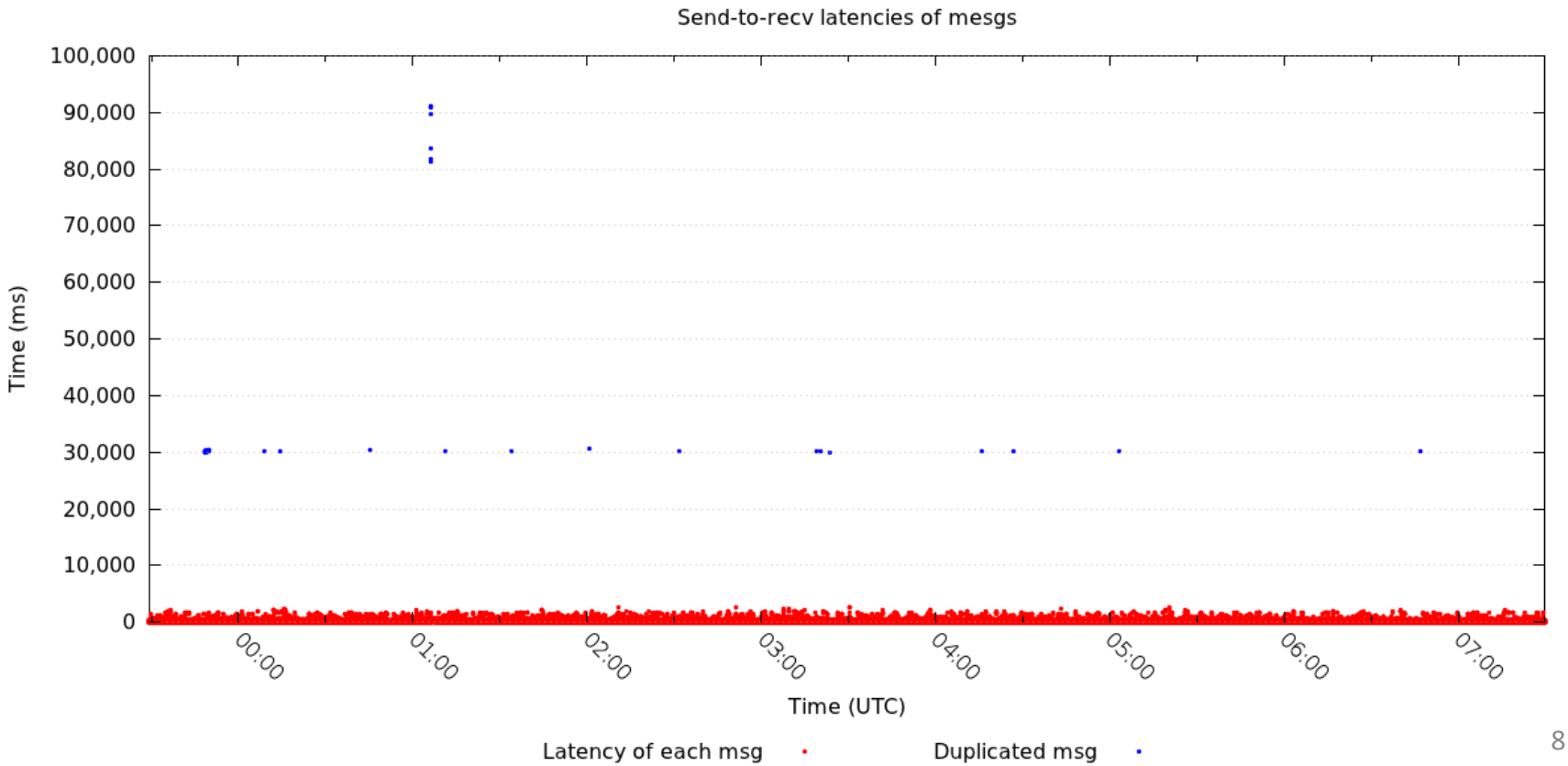
Rate-controlled measurement



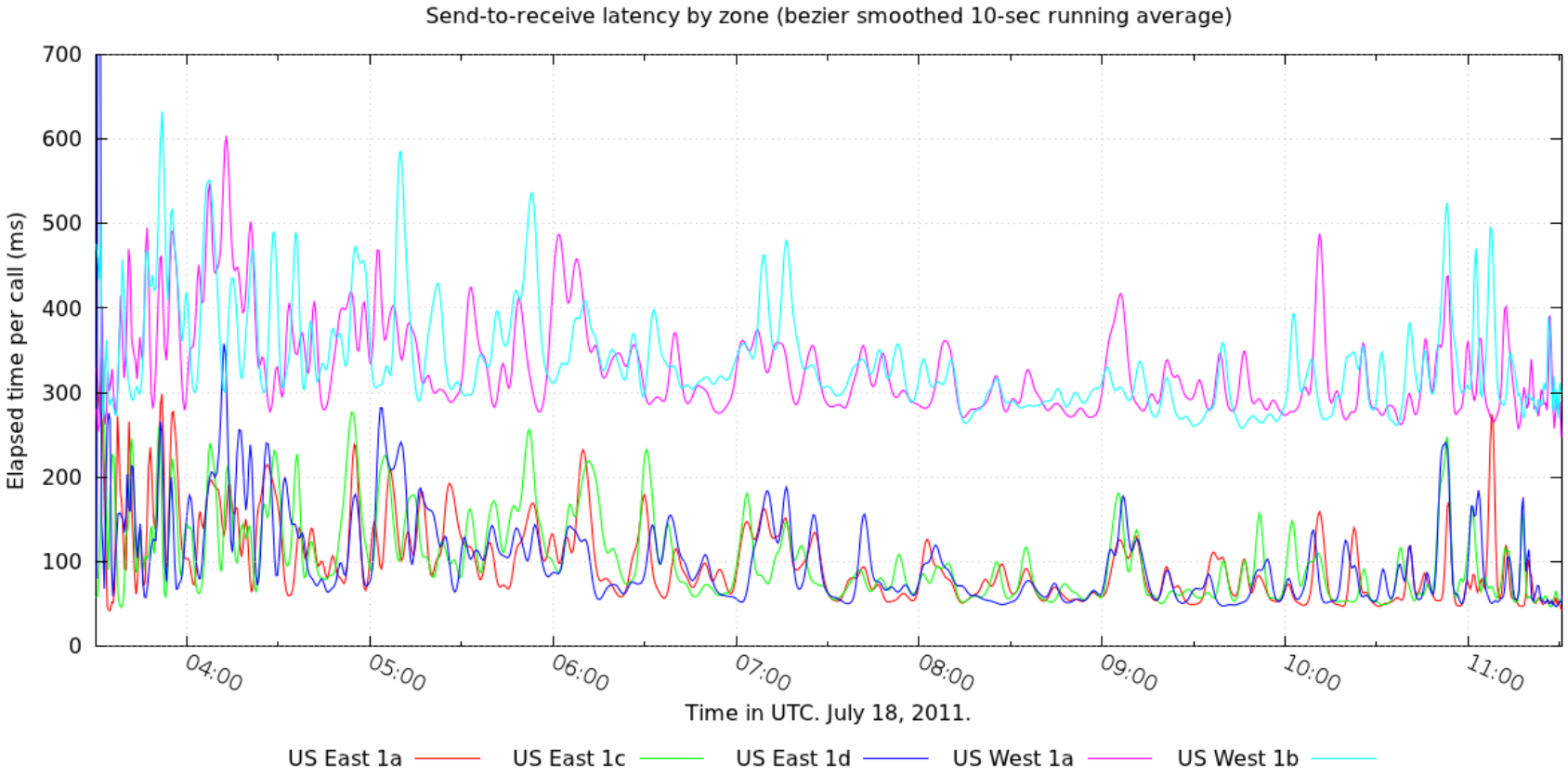
SQS performance measurement architecture

Huge latencies are gone!

- avg < 100 ms, max < 3 s
- Some big latencies turned out to be from duplicated messages. Message handling should be idempotent!



Performance by region/zone

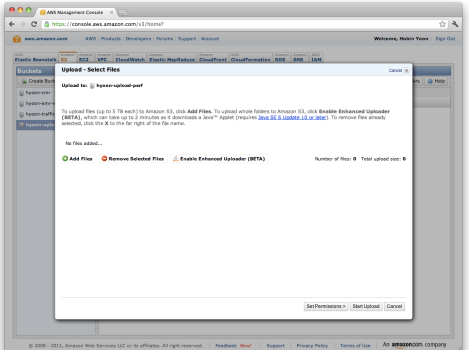


Message latency of SQS (Simple Queue Service)

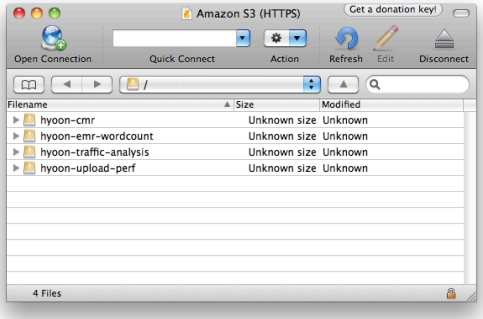
Optimizing upload performance to S3

There are various S3 upload tools…

- AWS web interface



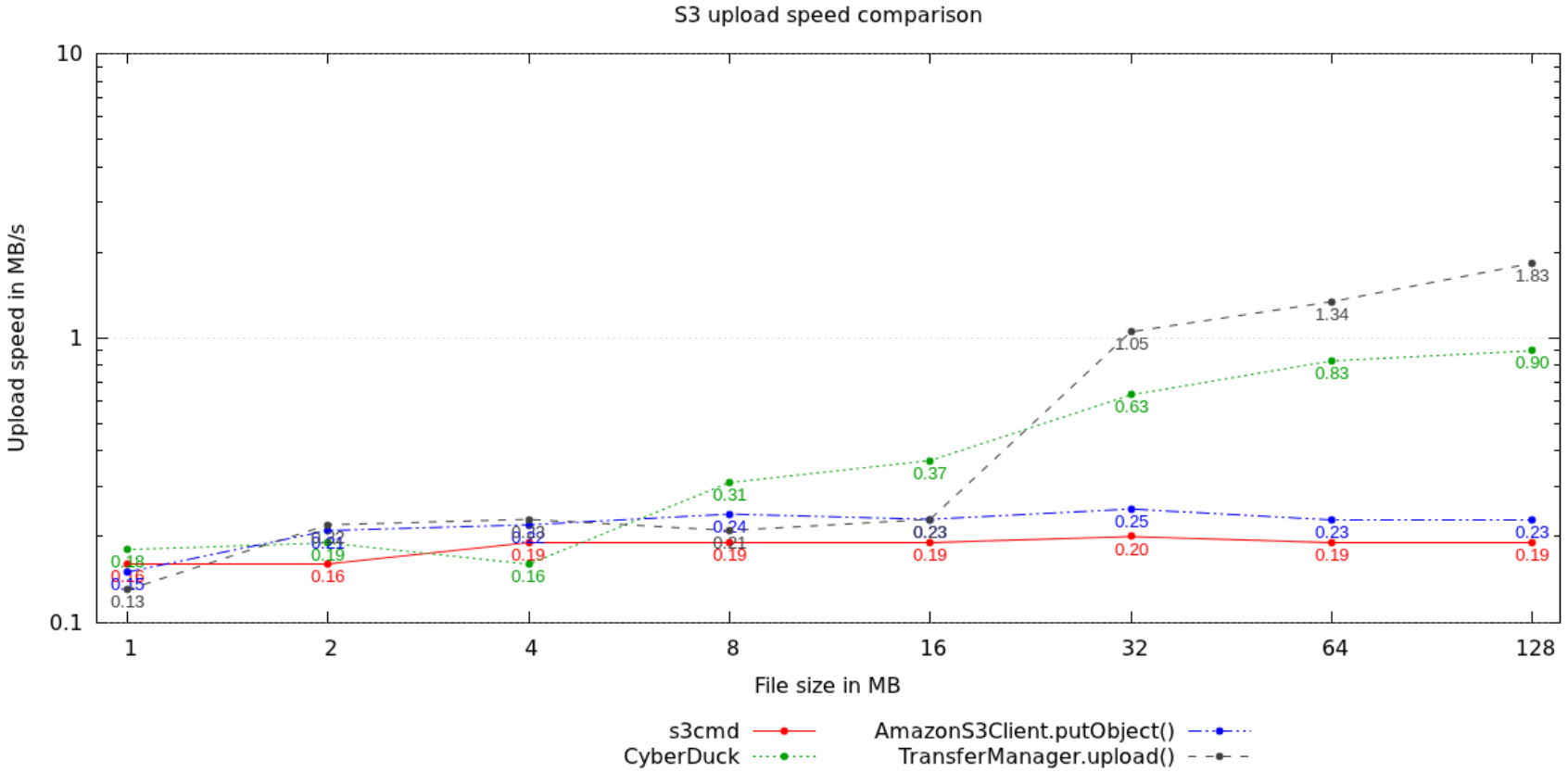
- CyberDuck



- s3cmd: command line S3 client

- Amazon Java APIs
  - Low level API: AmazonS3Client.putObject()
  - High level API: TransferManager.upload()

...with different performances



- # of connections: S3cmd 1, putObject 1, CyberDuck 5, and TransferManager 10.

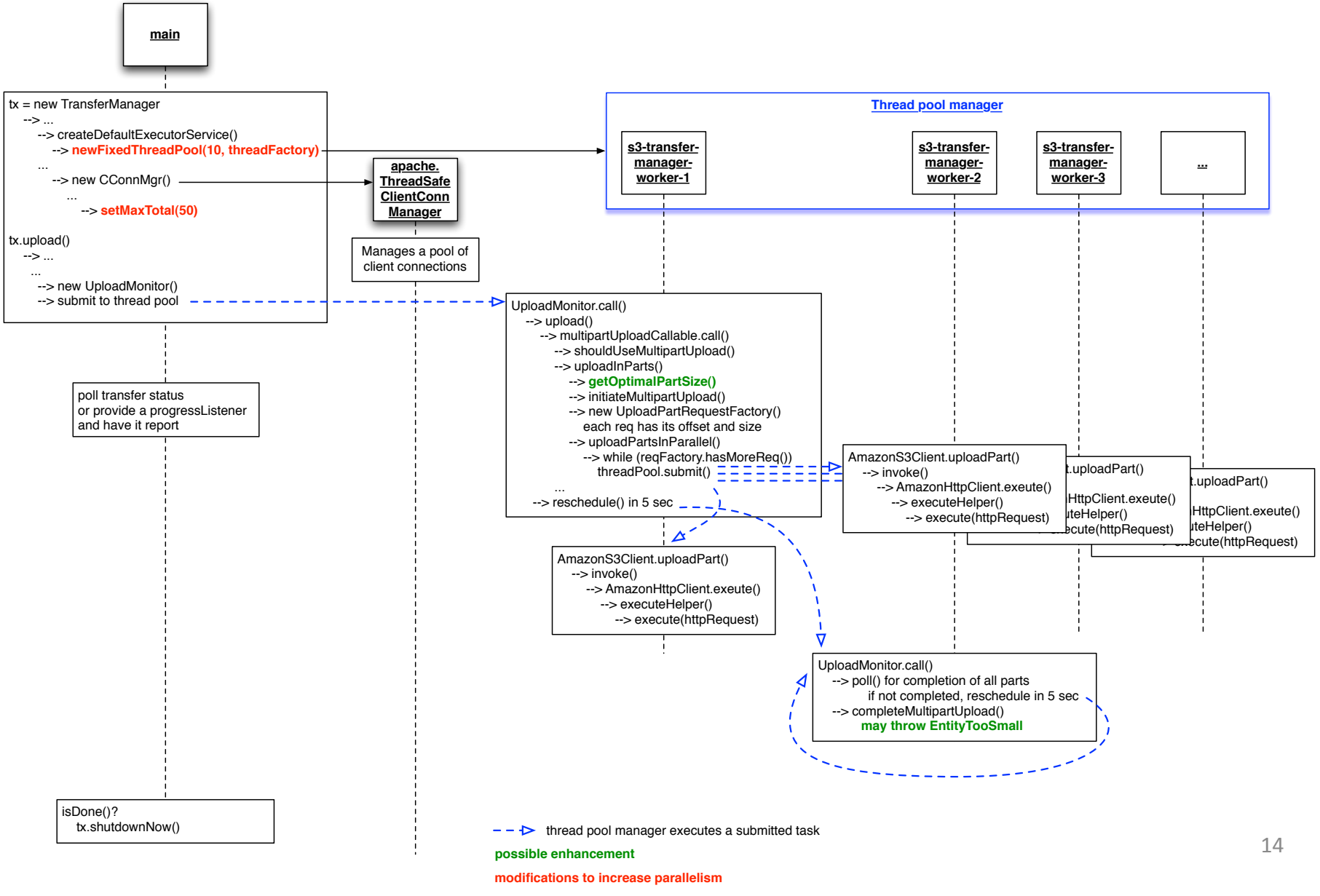
Multi-part upload of small files?

- TransferManager, with default configuration, does multi-part upload when file size > 16MB and chunk size >= 5MB

- S3 server does not allow it with smaller files…

  16:06:42.438 [s3-transfer-manager-worker-4] DEBUG com.amazonaws.services.s3.transfer.internal.UploadMonitor.call(UploadMonitor.java:159) - Status Code: 0, AWS Request ID: 3784CB129EBAEF41, AWS Error Code: EntityTooSmall, AWS Error Message: Your proposed upload is smaller than the minimum allowed size, S3 Extended Request ID: 7cymUI2eF7Ui9nTv3Tsl59JwPzWMdXl+K4R4Pb0rNmzJ/cC5Xzxo7UT6NMlCket6

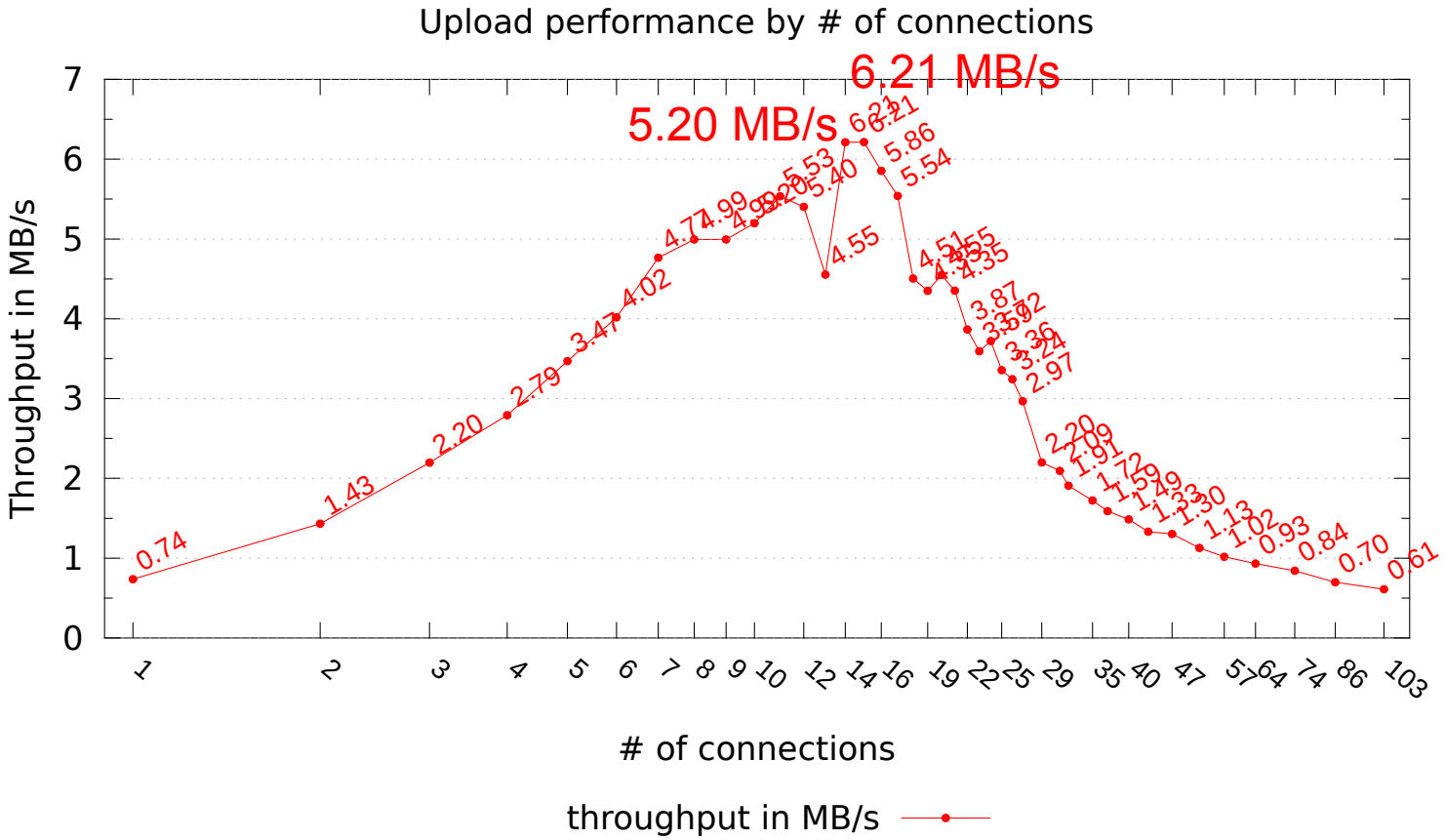
A closer look at TransferManager



With more threads and connections…

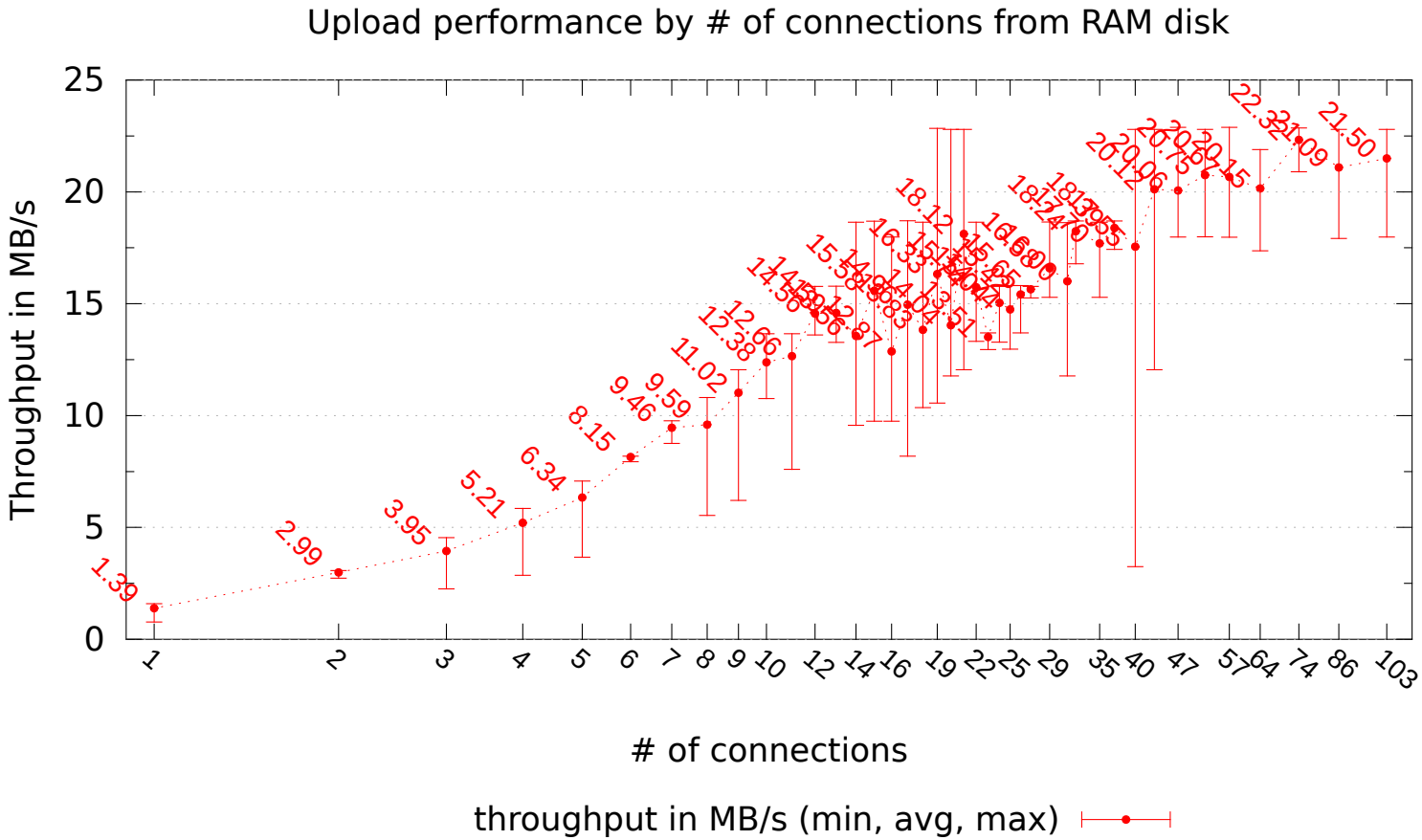
- Connection manager puts a limit on the # of connections - 50 connections.
  - to 200 connections.
- A pool of fixed # of worker threads - 10 threads.
  - with CachedThreadPool, which creates threads as needed and reuse ones when they are available.

… got 20% performance gain



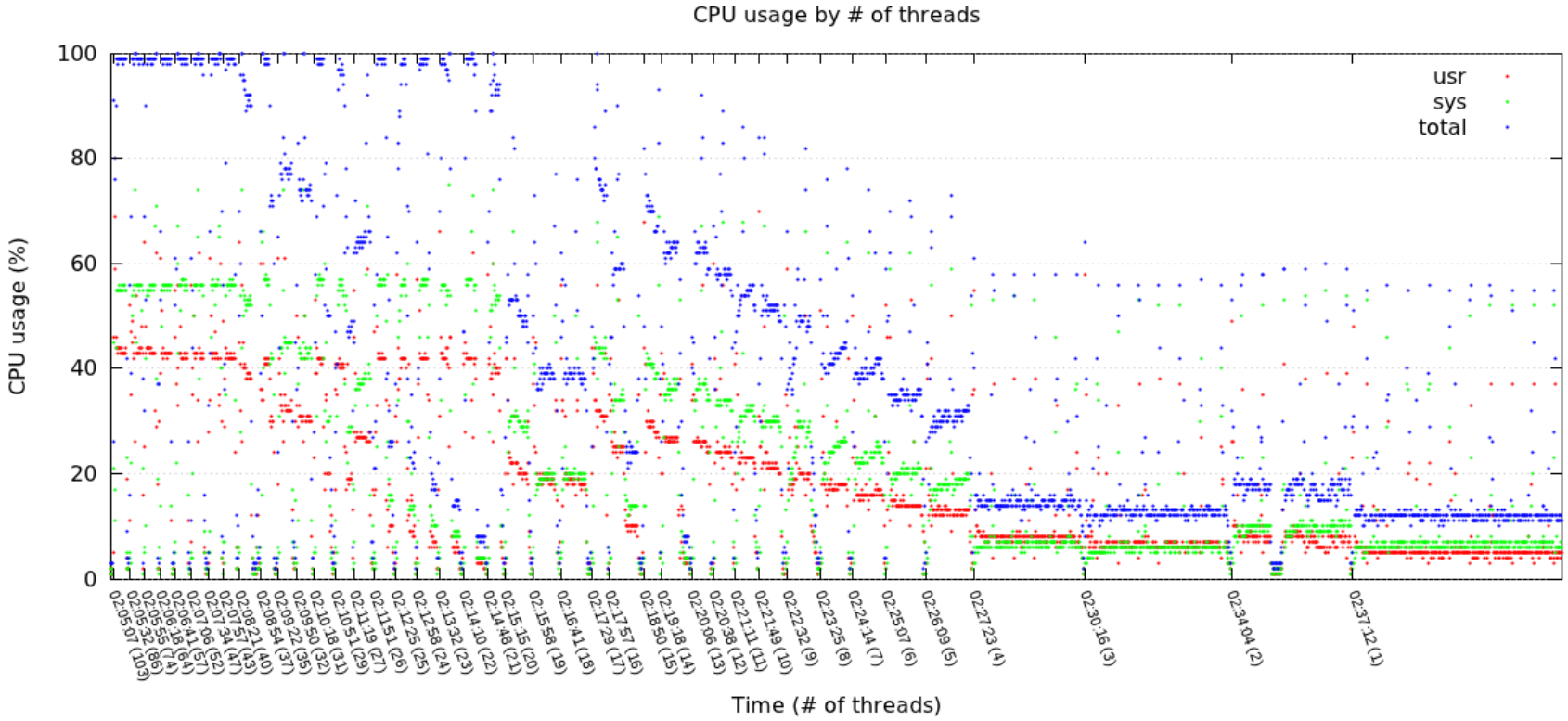
- from 5.20 MB/s with 10 connections, which is the max # of conn. With unmodified code,
- to 6.21 MB/s with 14 connections,
- but it start to decrease… because of HDD?

Data from RAM disk



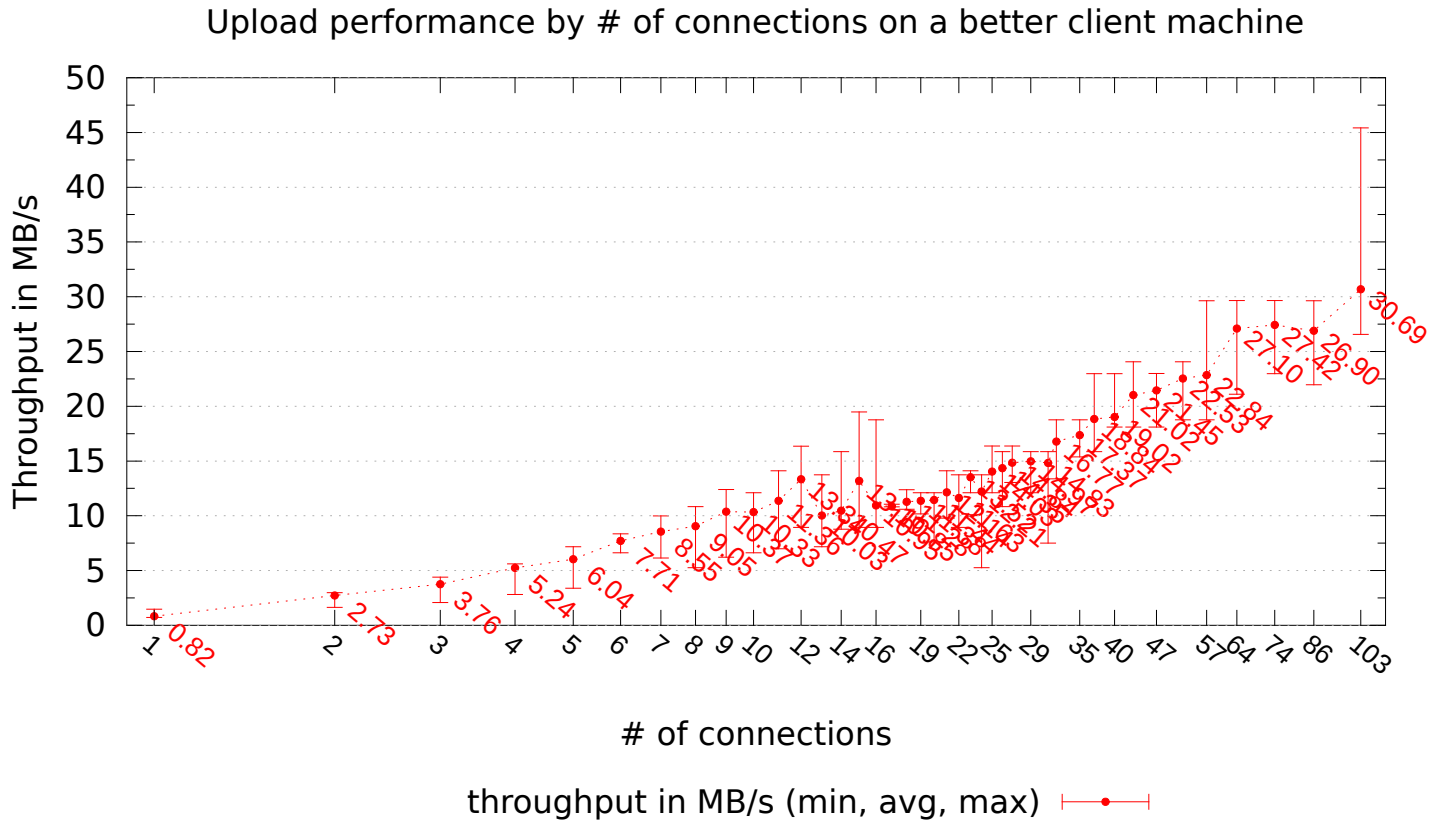
- Scaled up to 22.32 MB/s, which is 3.6 x more gain.
- Now the bottleneck is… CPU?

CPU usage



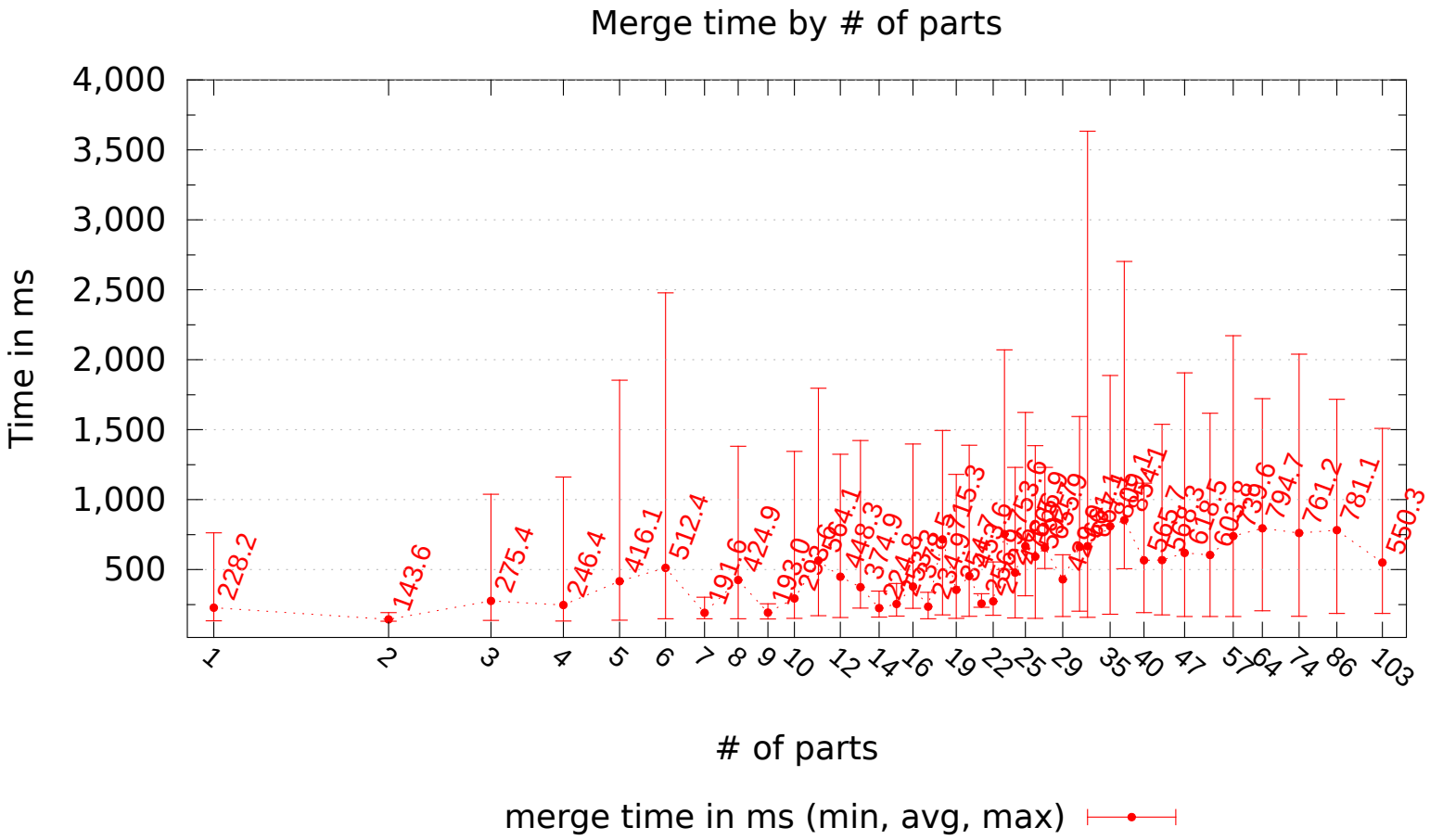
- CPU saturates around 20 threads.
  - On Macbook Pro 4,1. Core2 Duo 2.6GHz.
- On a better machine?

On a better machine



- 30.69 MB/s from 22.32 MB/s, which is 38% more gain.
- Switched to Xeon Quad Core 2.8 GHz, 12 GB of DDR3 RAM, 7200 rpm HDD
- from Core 2 Duo 2.6 GHz, 4 GB of DDR2 RAM, 5400 rpm HDD

Merging parts



- < 1 sec. Good scalability.

Lessons learned

- With minimal modifications to TransferManager and by detecting and eliminating bottlenecks one by one, high throughput can be achieved.
  - Unmodified code (5.20 MB/s)
  - More connections and threads (6.21 MB/s)
  - Data from RAM disk (22.32 MB/s)
  - With a better client (30.69 MB/s)

More throughput?

- Network may not be a bottleneck.
  - iperf from the client machine to a large EC2 instance showed 1 Gb/s with 120 connections.
- More throughput could have been achieved with either
  - more # of parts (with a bigger file) or
  - with multiple instances of TransferManagers.

Thank you!!

Comments or Questions?