Investigating the Overhead of the REST Protocol to Reveal the Potential of Using Cloud Services for HPC Storage

Frank Gadban[1], Julian Kunkel[2], and Thomas Ludwig[3]

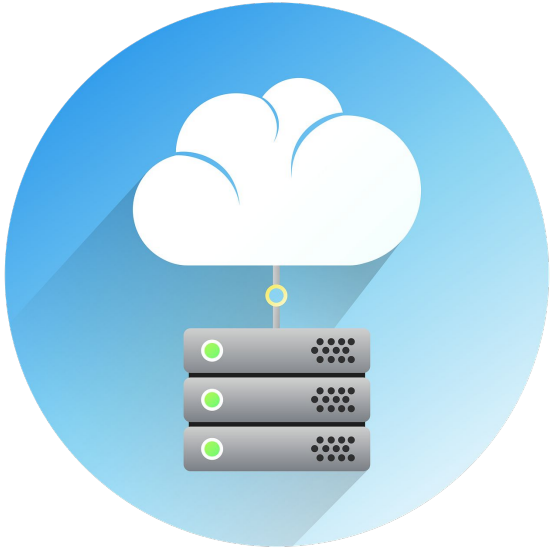
[1]University of Hamburg, 20146 Hamburg, Germany

[2] Reading University, Reading,UK

[3]DKRZ, 20146 Hamburg

Motivation: HPC & Cloud

The convergence between HPC and Cloud requires an optimal data sharing between HPC and cloud resources.



Object storage

- Organizes data into containers of flexible sizes referred to as objects
- Each object includes associated metadata and has a unique ID
- Usually, a simple hash of this ID determines where the object is physically located.
- HTTP is the defacto transfer protocol for Cloud storage

This paper

- Evaluates base performance for storing and retrieving objects
  - Compare with the native HPC transfer protocol (MPI)
- Assesses the performance improvements of HTTP2 and HTTP3 in the HPC context
- is available in the JHPS incubator

Motivation: Context to Storage

The research conducted here is embedded into my PhD topic

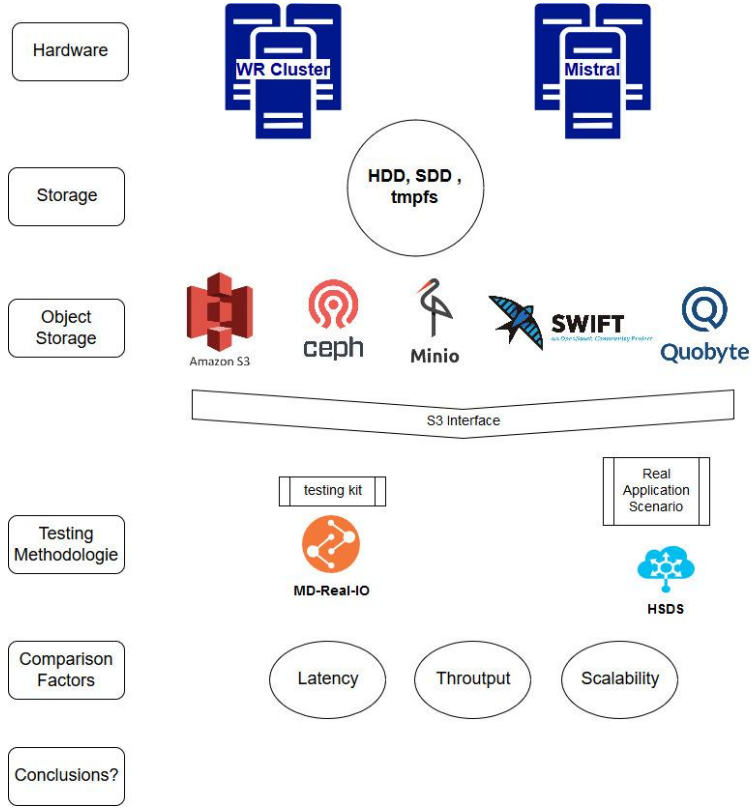
Initially we aimed to measure and discuss pro/cons of cloud solutions in the HPC environment

- In particular, understanding performance limitations

It turned out, that basic understanding of the REST API performance in the HPC context was not well researched

- HPC software aims to be performance optimized
- Cloud storage scales but doesn't aim to extract node performance

The core idea of this paper was to investigate REST limitations in the context of object storage



Methodology

- Assessment of performance
  - Aim: identify performance bottlenecks
  - Utilize performance counters to measure system performance usage
    - Measure CPU/memory utilization using Likwid on client/server
  - Assess performance relative to machine capabilities
  - Generalize measurements using a performance model
- Experiments
  - Execute benchmarks on two representative clusters
    - WR: Small research cluster, GBit Ethernet (Cloud environment)
    - Mistral (FDR Infiniband): DKRZ’s supercomputer for climate science
  - Define a REST Benchmark that is integrated with Likwid.
  - Integrate the OSU Micro-Benchmark with Likwid to test MPI.
  - Results Comparison.
  - Investigate the performance of HTTP1.1 vs. HTTP2 vs. HTTP3
- Model Validation: compare predicted results with experimentally observed values

Model

t(request) is the time to complete a request:

t(request) = t(client) + t(network) + t(server) (1)

t(client) = t(compute) + t(memory) + t(queued) (2)

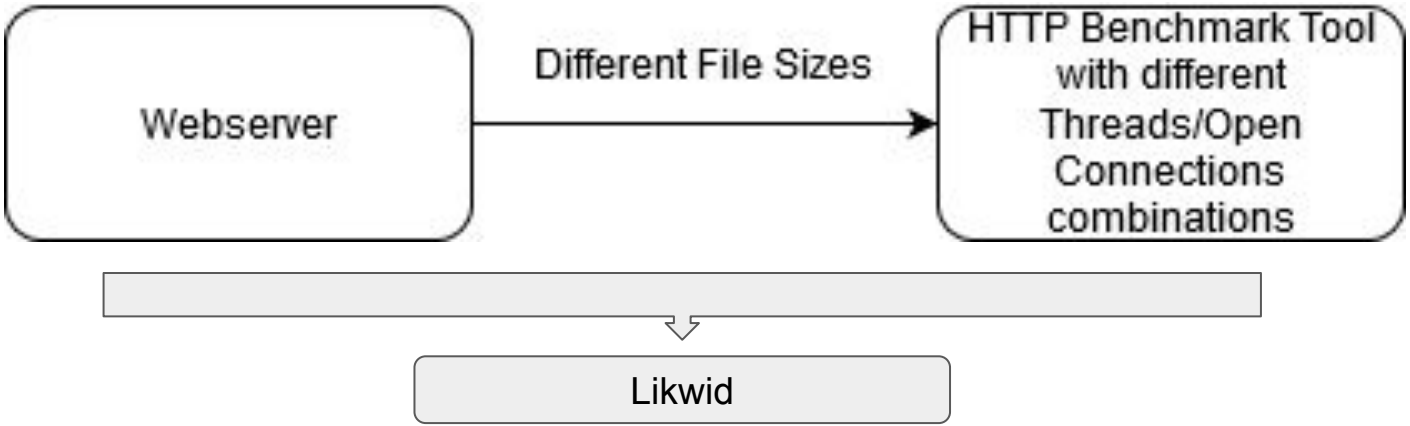
t(server) = t(compute) + t(memory) + t(queued) + t(pending) (3)

After some refinements we end up with:

$$t(request) = \alpha \cdot rtt + \beta_1 \cdot \frac{CUCs}{Rs} + \beta_2 \cdot \frac{L3EVs}{mem\_tp} + \beta_3 \cdot \frac{CUCc}{Rc} + \beta_4 \cdot \frac{L3EVc}{mem\_tp} + \beta_5 \cdot \frac{Obj\_size}{net\_tp}$$

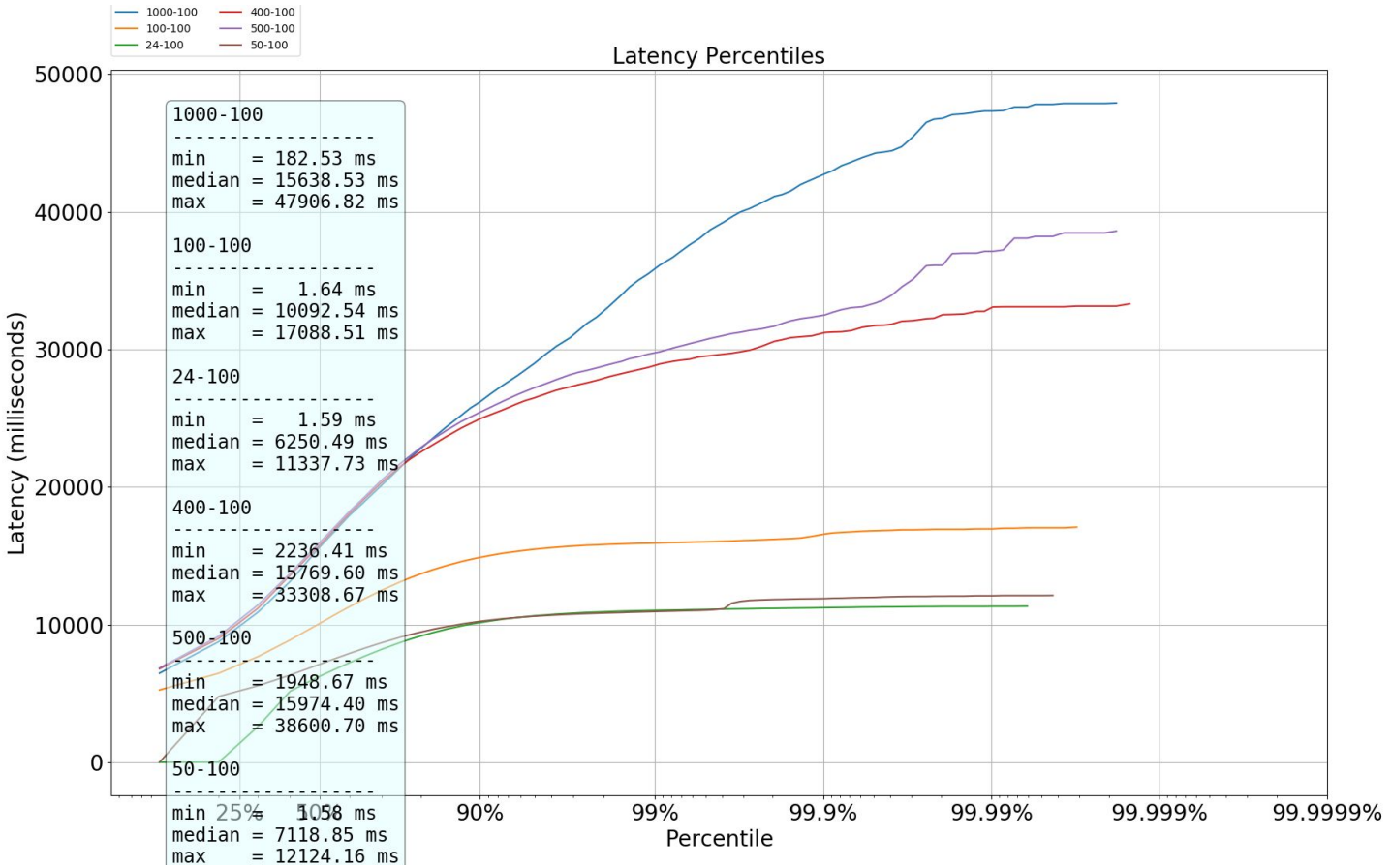
- α is a weighting factor ($0 \leq \alpha < 1$) [25], β_i are platform and protocol-dependent
- rtt round trip time.
- R: CPU clock rate in Hz.
- CUC: number of unhalted cycles on each core
- L3EV: data volume flowing through L3 cache. L3EVs and L3EVc represent server and client, respectively
- net_tp:effective network throughput

Benchmark Execution



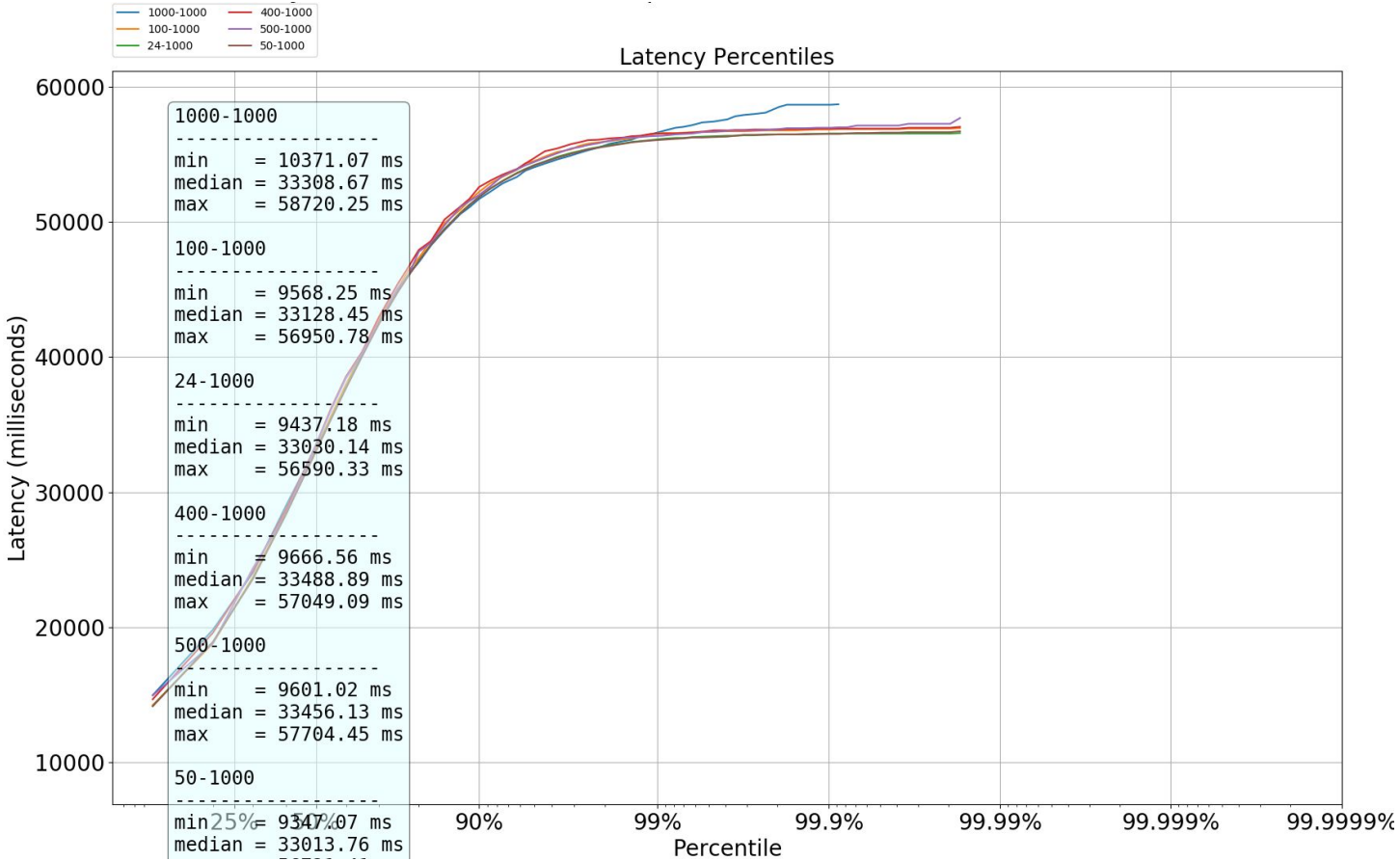
- Server: lighttpd web server
- Benchmark tool: wrk2 via. HTTP
- Data: Files containing randomly generated data
- Storage: Files are stored in tmpfs to minimize influence of storage media
- Likwid is recording the performance counters

Latency-results-wr2



Latency variation in relation to open connections for a file of size 100 KB

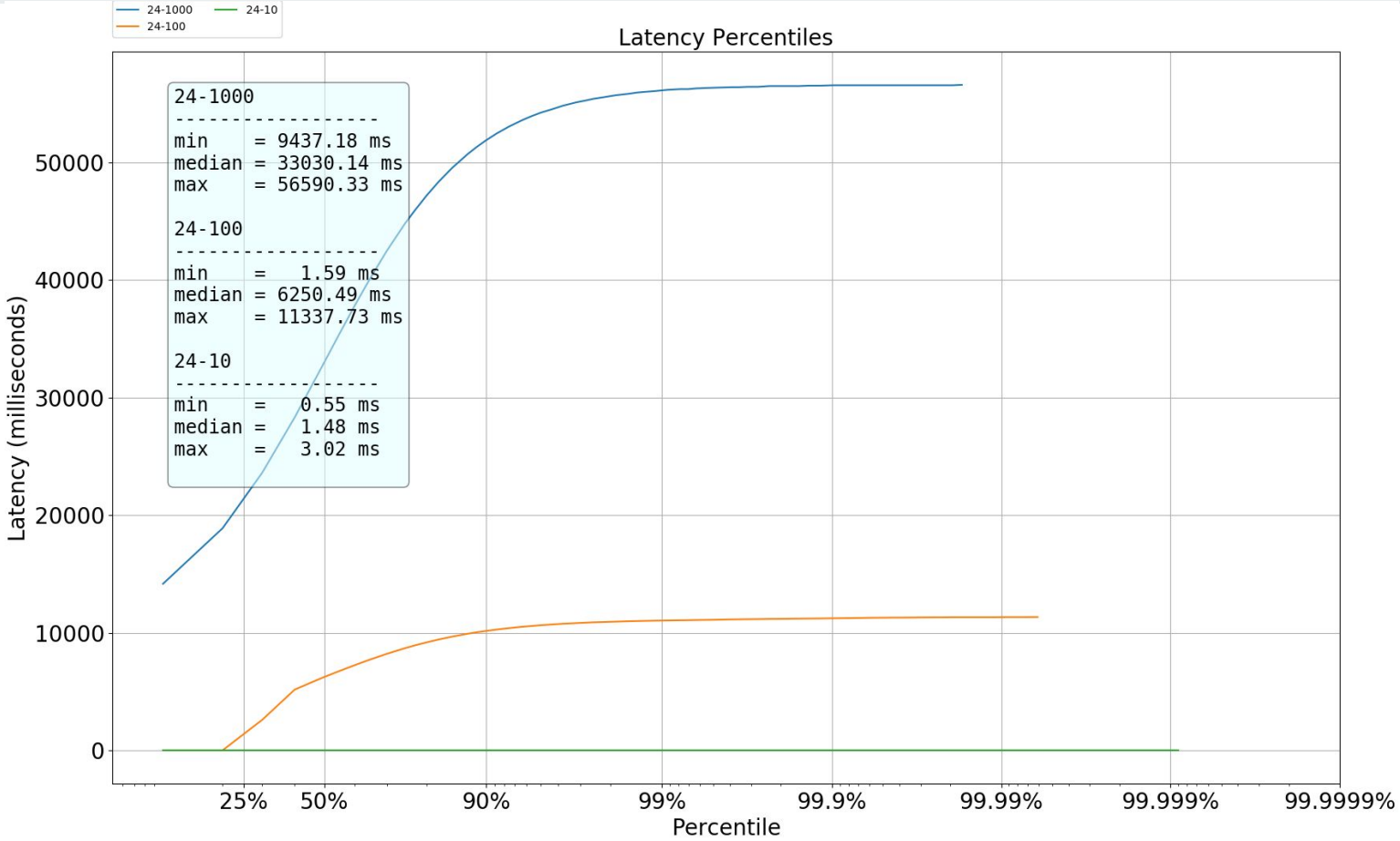
Latency-results-wr1



Latency variation in relation to open connections for a file of size 1000 KB

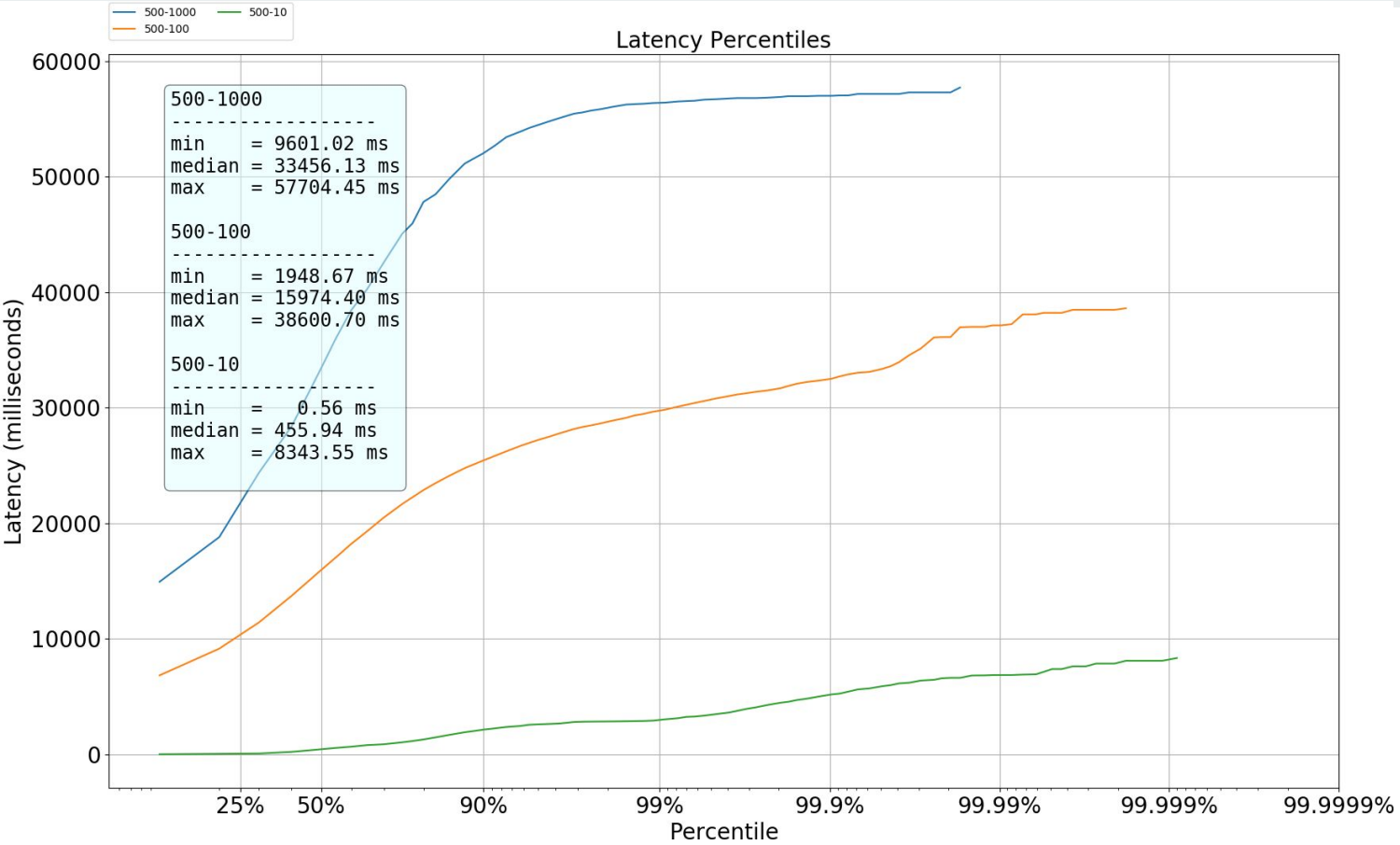
- Latency increases with the number of open connections especially for small file.
- When the file size grows beyond a certain limit, the number of connections will become irrelevant to the already high introduced latency.

Latency-results-wr3



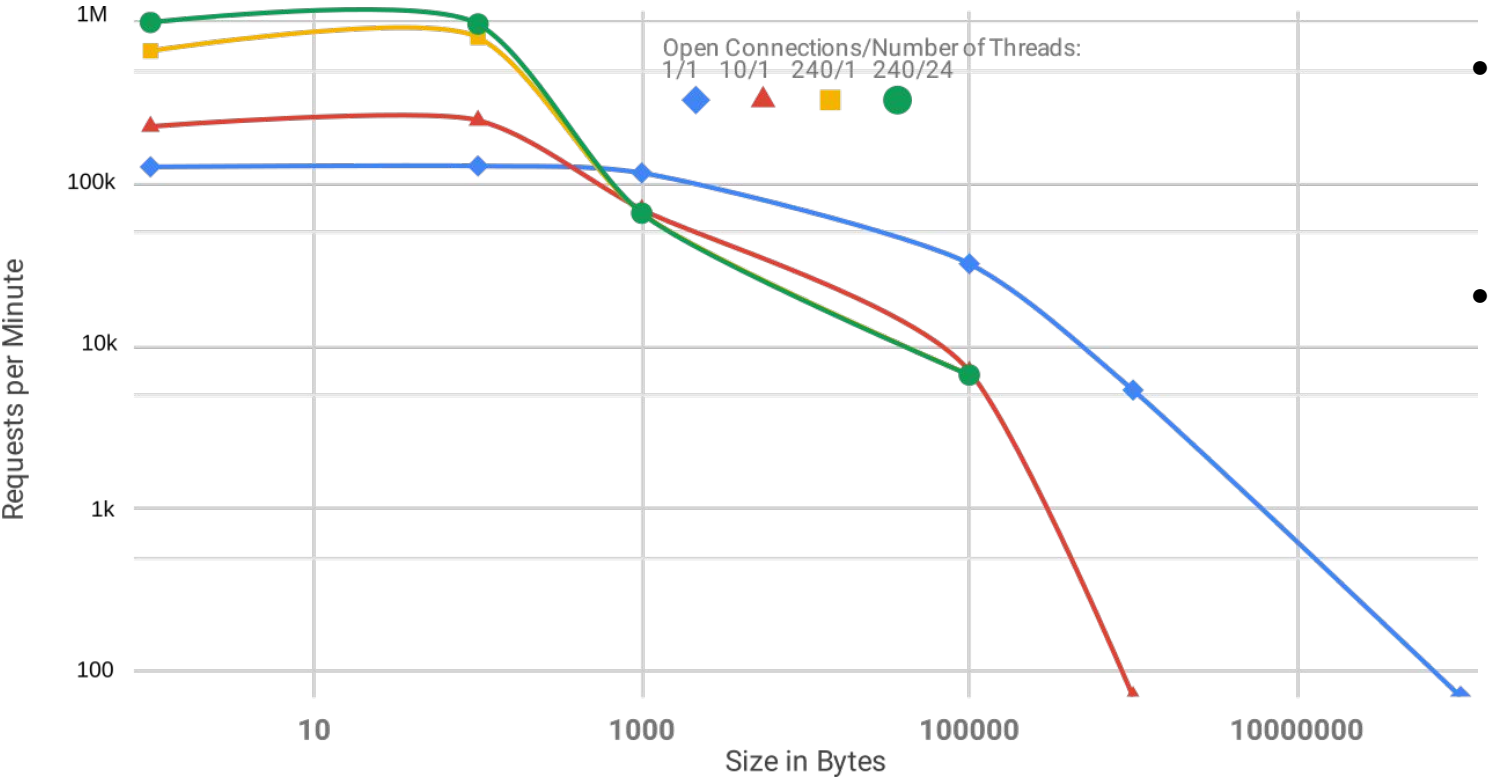
Latency Variation in relation to file size for 24 open connections

Latency-results-wr4



Latency Variation in relation to file size for 500 open connections

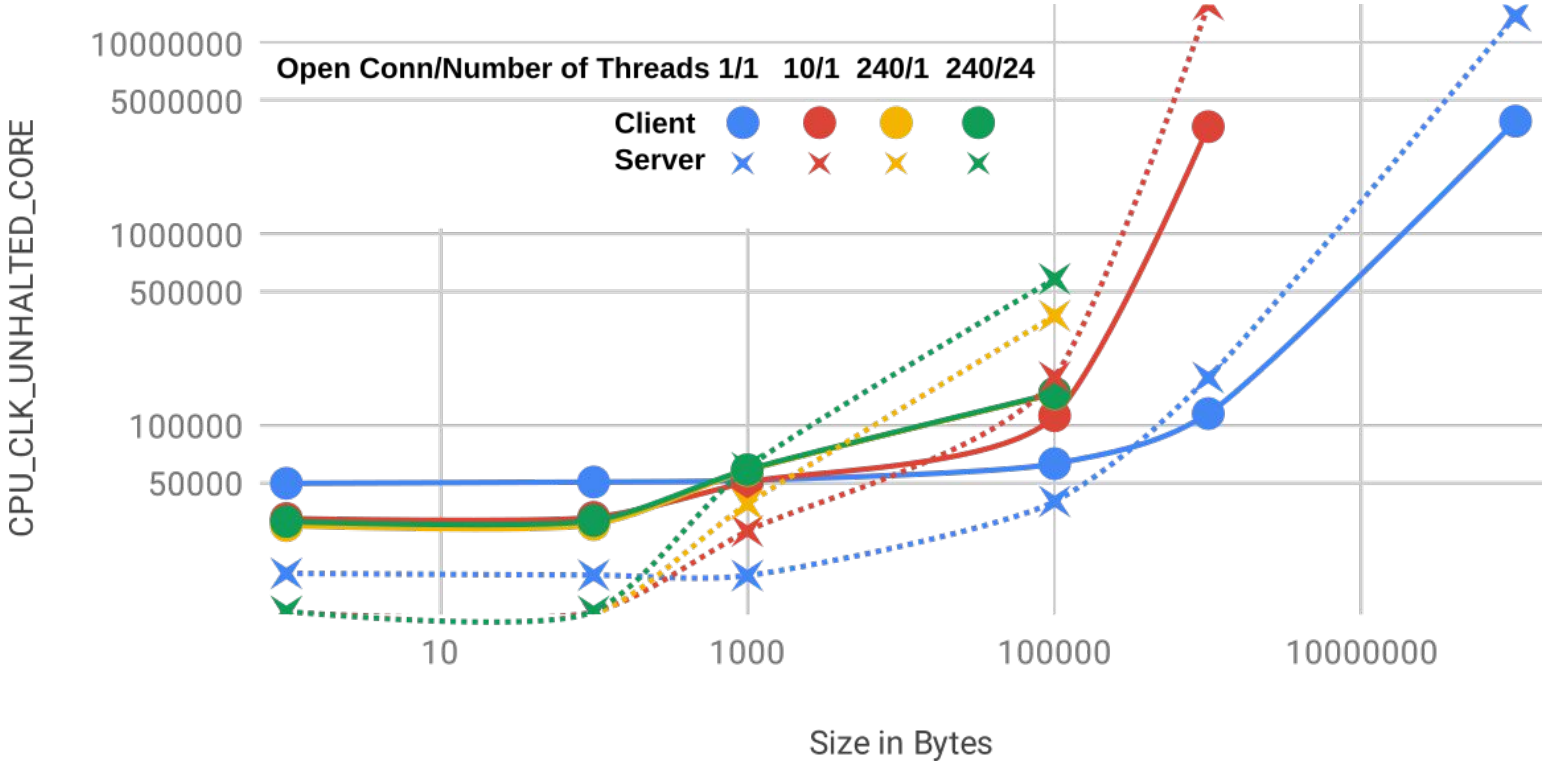
Throughput



- An increase in the number of Open Connections or in the number of threads will increase the throughput, for file sizes below 1 MB
- Optimize latency and throughput = use one or a relatively small number of open Connections and label the web requests accordingly= HTTP multiplexing

Throughput in requests per minute (logarithmic scale) related to object size for different combinations of Open Connections/Threads.

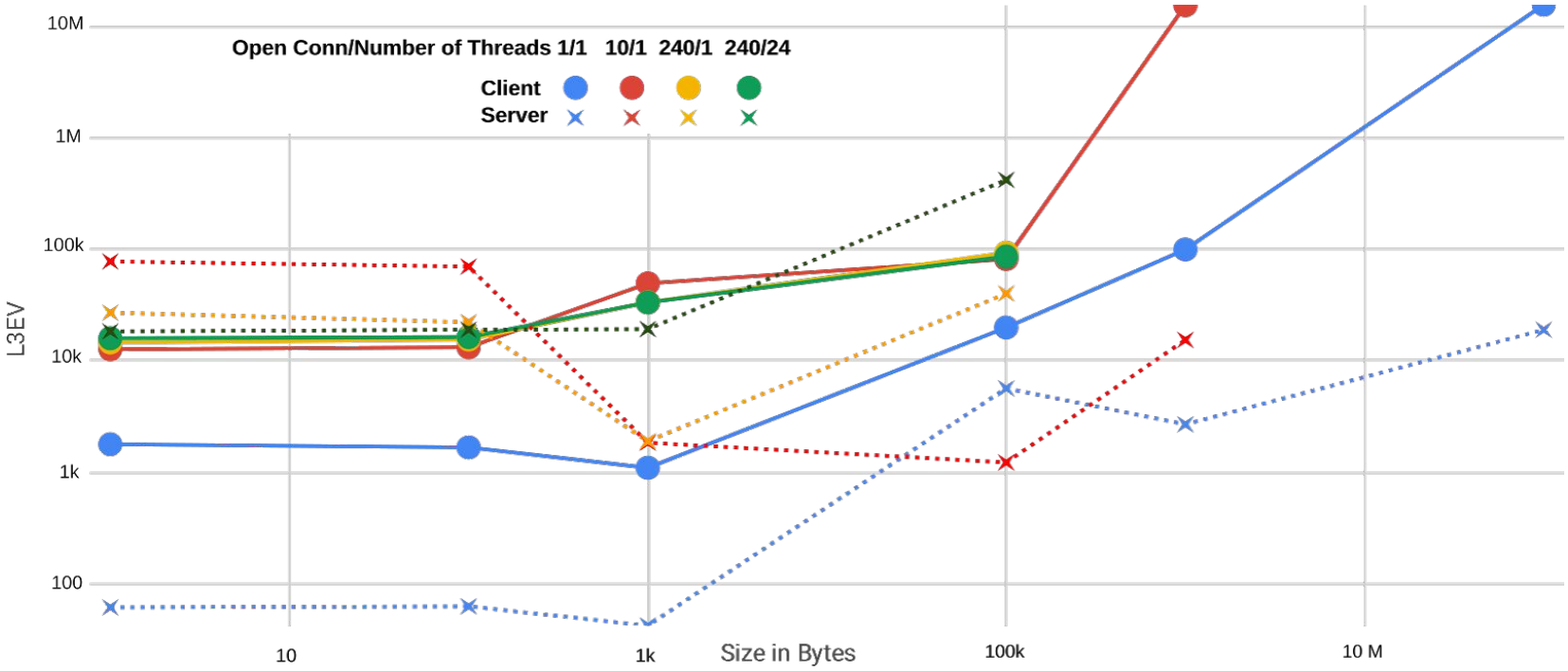
Resource Usage Measurements - CPU Cycles



CPU usage for the client and server related to size, for different Open Connections/Threads combinations

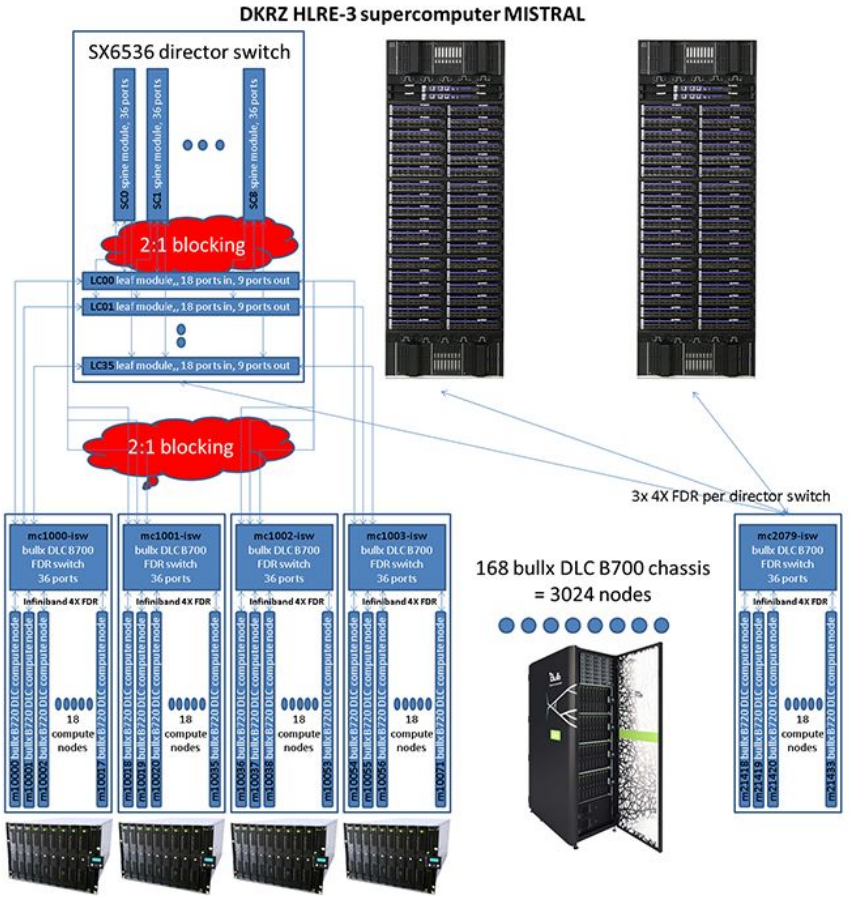
Resource Usage Measurements - Memory

Resource Usage Measurements



L3 memory evicted volume for the client and server related to size, for different Open Connections/Threads combinations

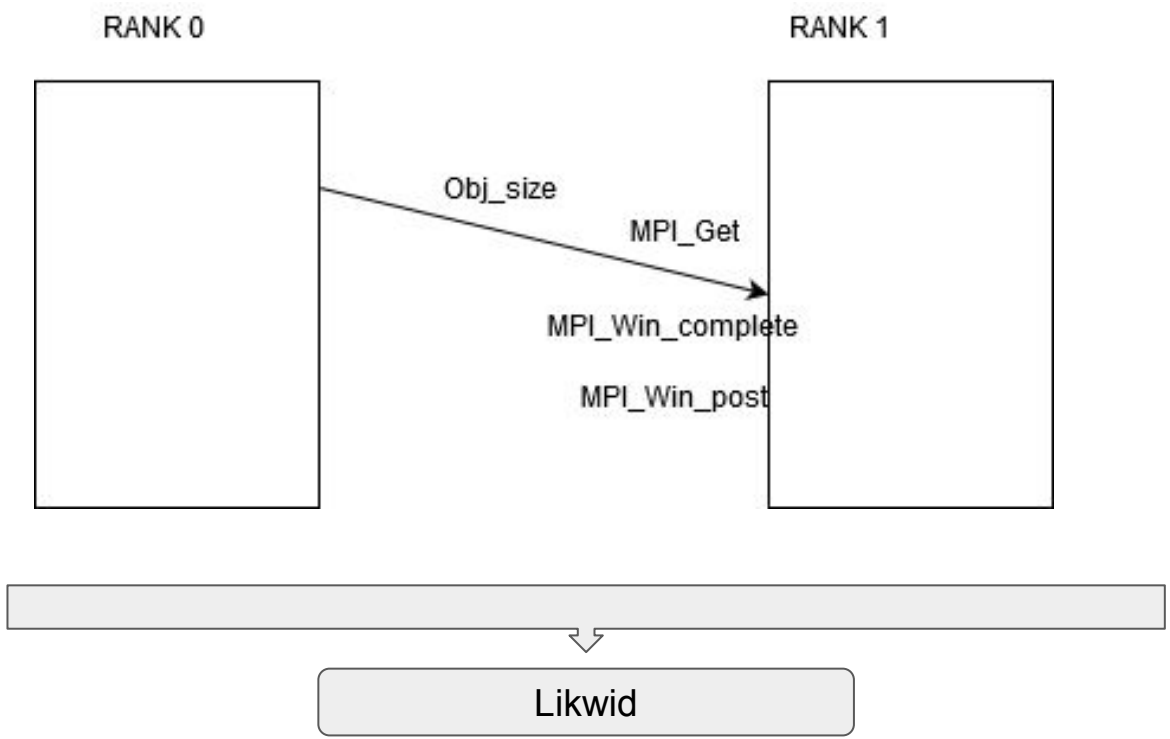
Benchmarks over IB - REST vs. MPI



Mistral, the High Performance Computing system for Earth system research at the German Klima research Center DKRZ.

Benchmarks over IB - REST vs. MPI- OSU Micro-Benchmark

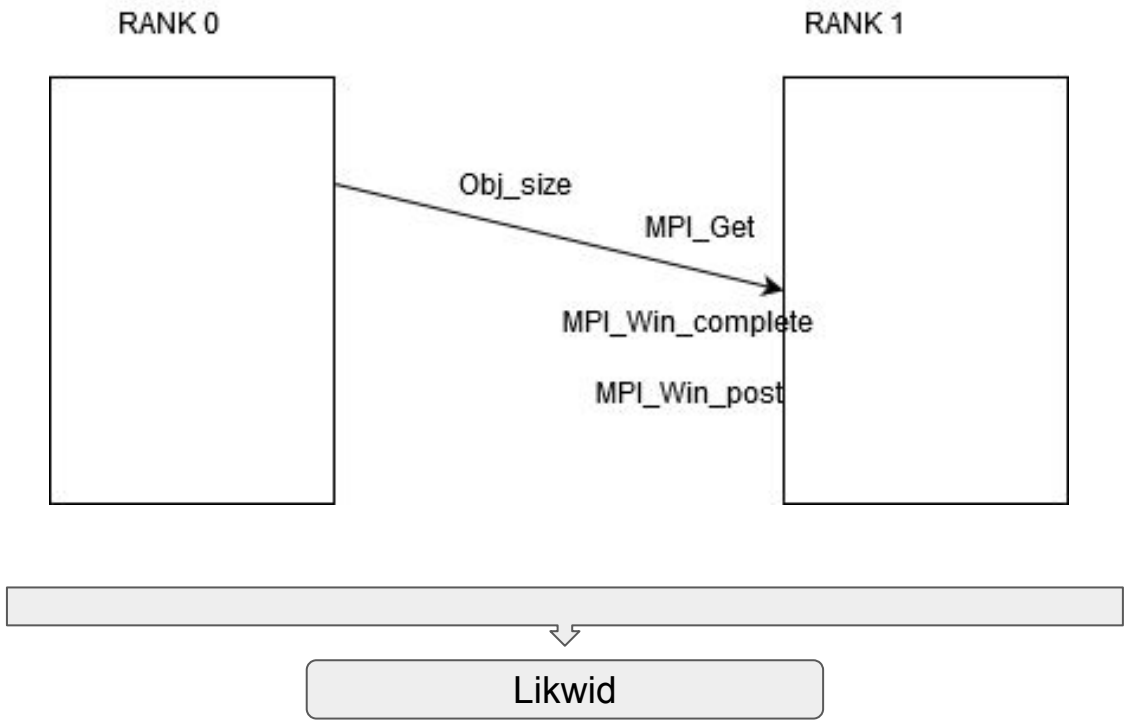
osu_get_latency



- RANK1 calls MPI_Get to directly fetch data of a certain size from the RANK 0 process's window into a local buffer.
- RANK1 waits on a synchronization call (MPI_Win_complete) for local completion of the Gets.
- After several iterations the average get latency numbers is reported.

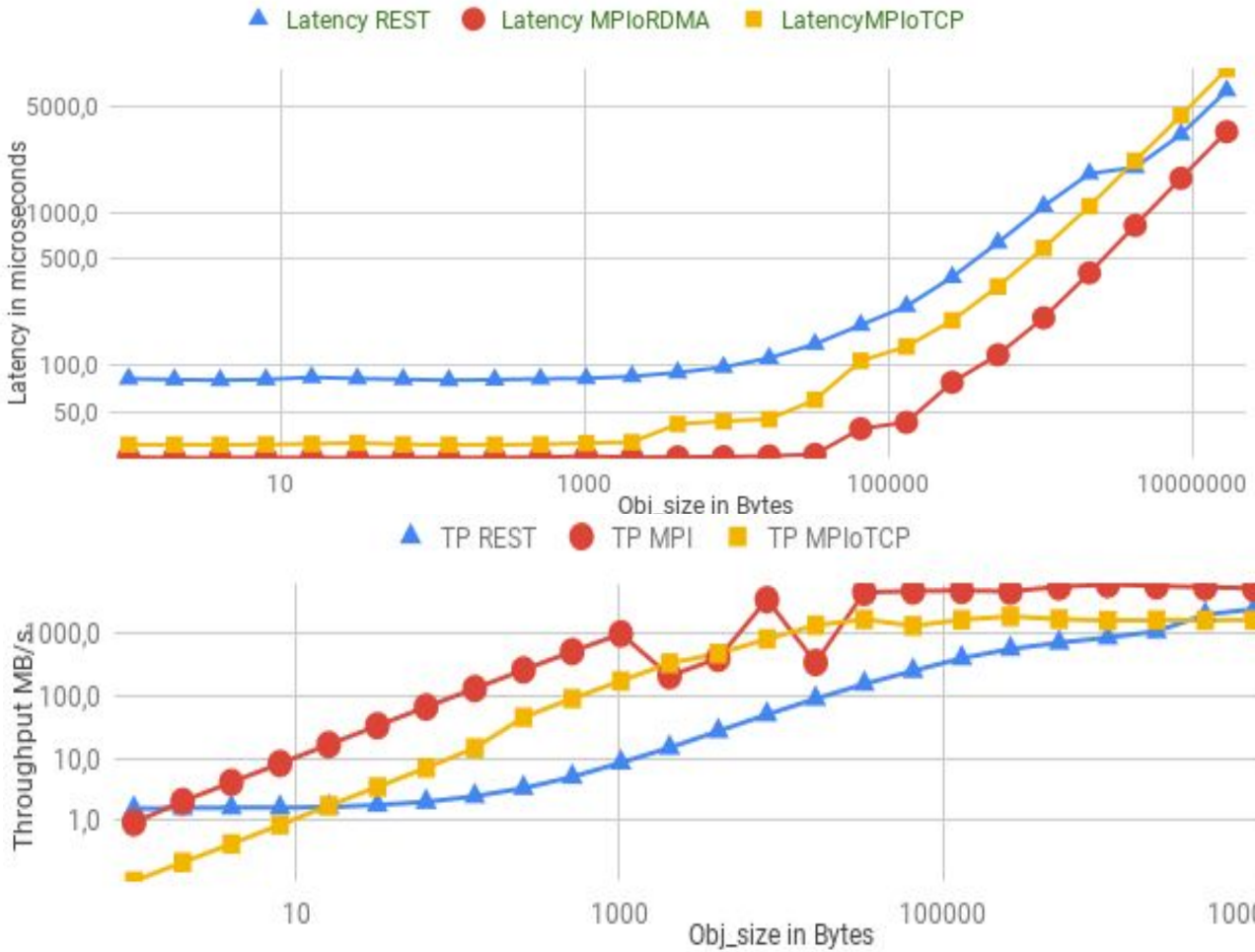
REST vs. MPI - OSU Micro-Benchmark

osu_get_bw



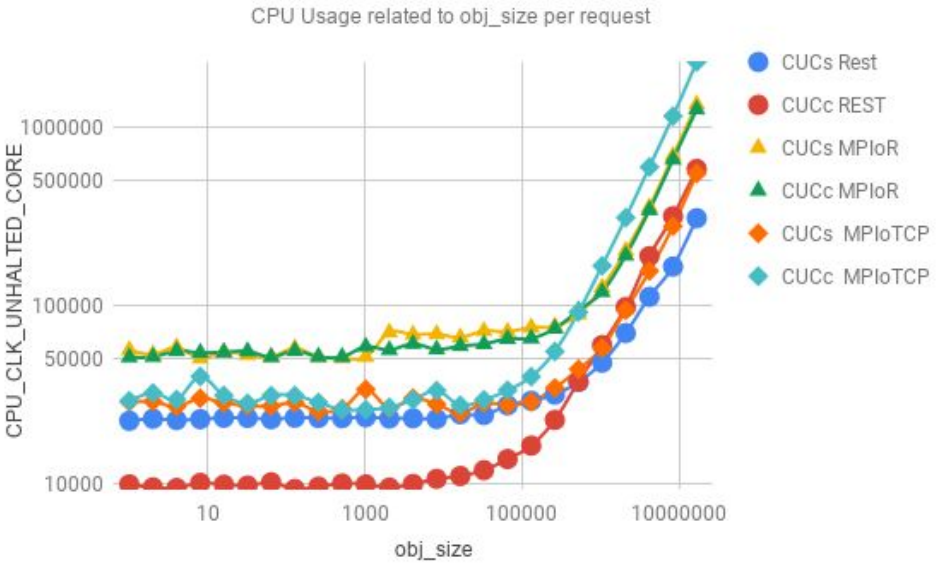
- RANK1 calls a fixed number of back-to-back MPI_Gets and then waiting on a synchronization call (MPI_Win_complete)
- After several iterations, the bandwidth is calculated based on the elapsed time and the number of bytes received by the origin process

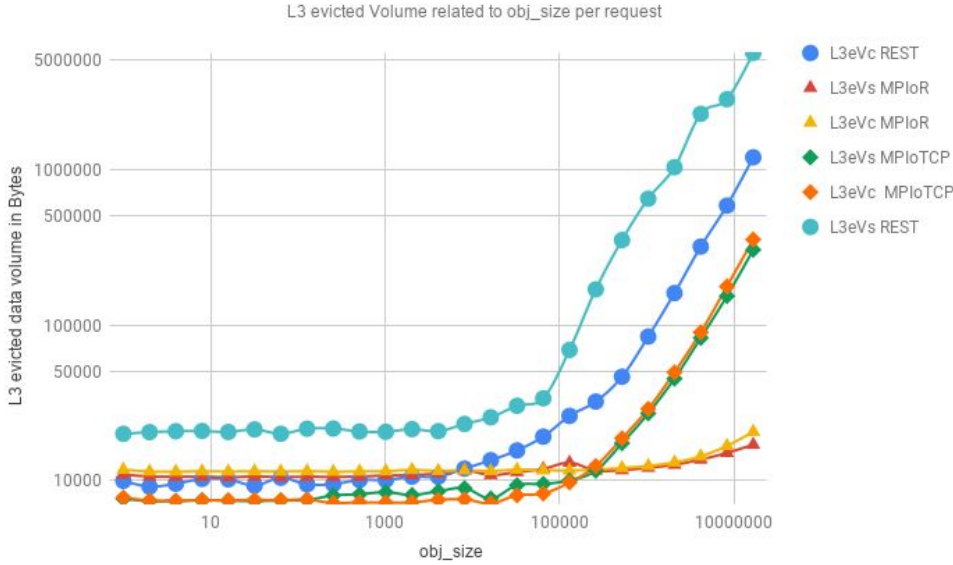
REST vs. MPI : Latency and Throughput



- For small object sizes, the latency of REST is higher
- MPI and REST over TCP have similar throughput especially for small and large file -> TCP Overhead
- MPI performance dip is due to eager & rendez-vous
- The system isn't optimal configured

REST vs. MPI : Resource Usage





- Minimal L3EV for MPI over RDMAoIB because of the direct data transfer
- L3EV for both REST and MPI over TCPoIB is constant for files smaller than 100 KB but increases exponentially afterward
- Higher CPU usage for MPI

Model Evaluation

Hardware and Network Parameters:

- rtt = 0.06ms
- mem_tp = 17.088 MB/s
- net_tp = 5.9 GByte/s
- R=2.49 GHz

β_i in our model are calculated using the Excel Regression Tool

RESToTCPoIB

$$t(request) = rtt + \frac{CUCs}{Rs} + 6 \cdot \frac{L3EVs}{mem\_tp} + \frac{CUCc}{Rc} + \frac{L3EVc}{mem\_tp} + \frac{3}{2} \cdot \frac{Obj\_size}{net\_tp} \quad (11)$$

MPIoTCP

$$t(request) = 0.1 \cdot rtt + \frac{CUCs}{Rs} + \frac{L3EVs}{mem\_tp} + \frac{CUCc}{Rc} + \frac{L3EVc}{mem\_tp} + 2.7 \cdot \frac{Obj\_size}{net\_tp} \quad (12)$$

MPIoRDMA

$$t(request) = \frac{1}{2} \cdot \frac{CUCs}{Rs} + \frac{L3EVs}{mem\_tp} + \frac{1}{2} \cdot \frac{CUCc}{Rc} + \frac{L3EVc}{mem\_tp} + \frac{Obj\_size}{net\_tp} \quad (13)$$

HTTP1.1 vs HTTP2 vs HTTP3 : WR Cluster

- Support for the 3 protocols:
  - Webserver: openlitespeed
  - Benchmark tool: h2load
  - Patched version of OpenSSL provided by the ngtcp2 team

h2load

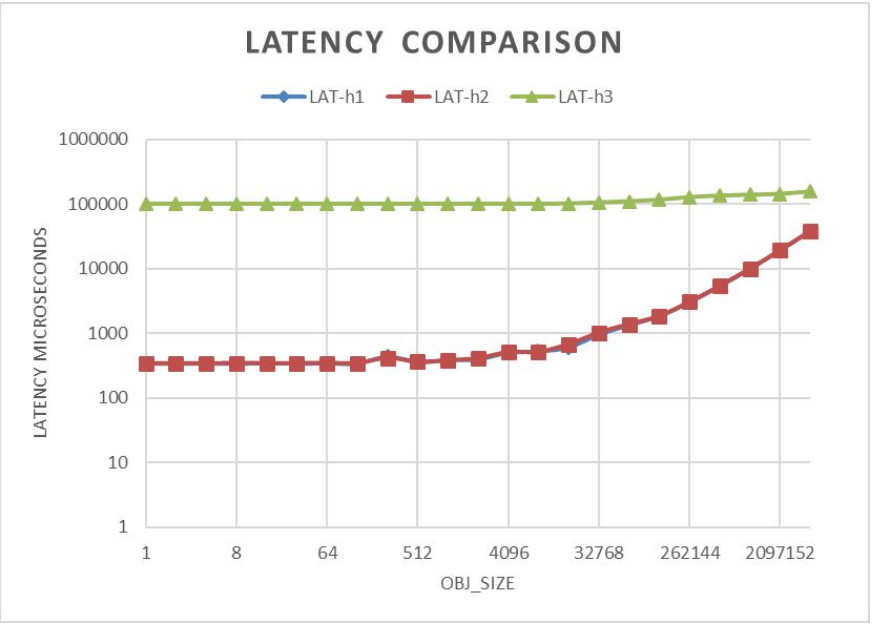
HTTP/2

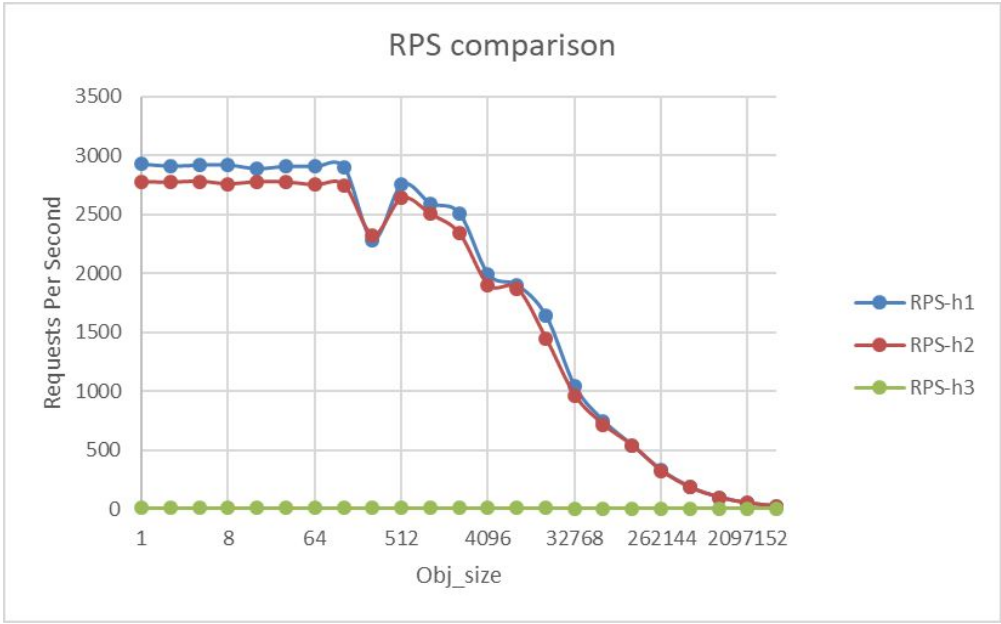
- Binary streams
- Multiplexing single TCP connection
- HTTP headers compression
- Server Push

HTTP/3

- UDP Based
- Stream multiplexing
- Stream and connection-level flow control
- Low-latency connection establishment
- Connection migration

HTTP1.1 vs HTTP2 vs HTTP3 : WR Cluster





Conclusion: HTTP1.1 and HTTP2 offers similar performance, the one from HTTP3 seems buggy!

HTTP1.1 vs HTTP2 vs HTTP3 : On Mistral, using IB

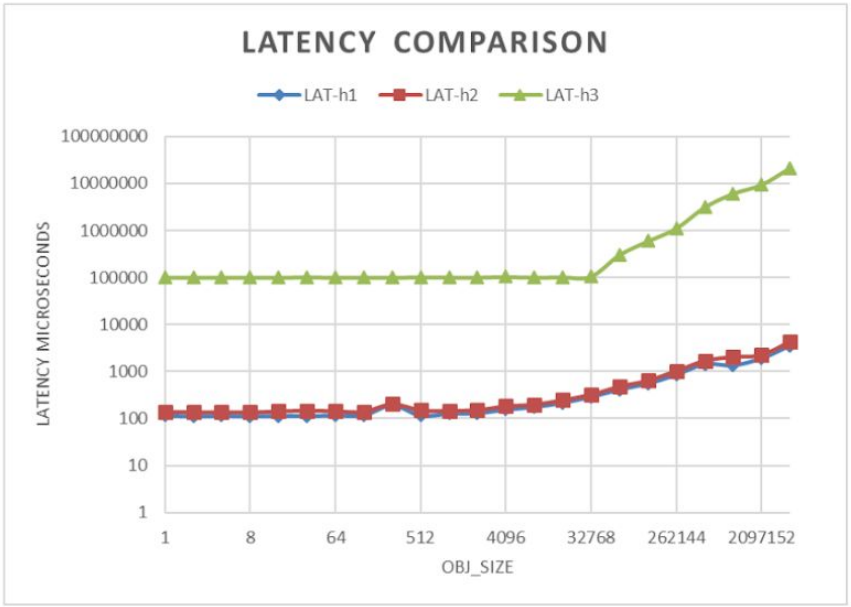


Fig. 16. Latency results for the different protocols

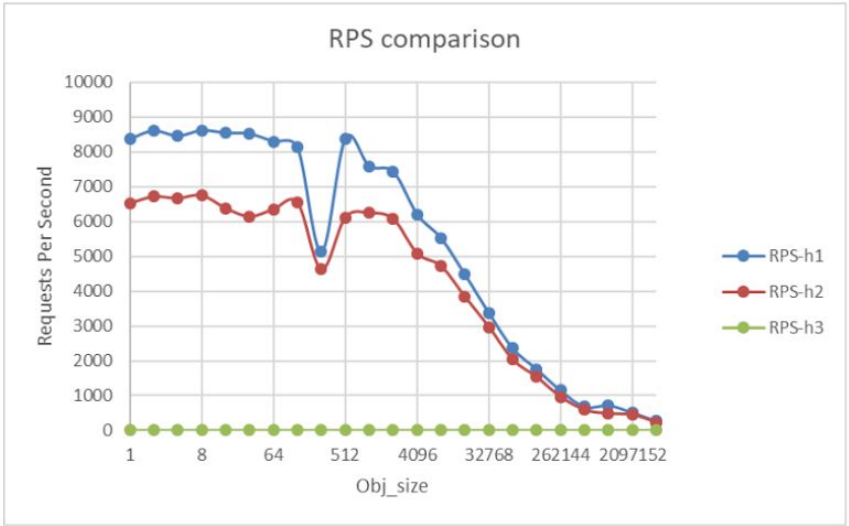


Fig. 17. Throughput results for the different protocols

HTTP1.1 vs HTTP2 vs HTTP3: Resource Usage

A closer look at the evolution of the parameters defined in our model reveals the cause

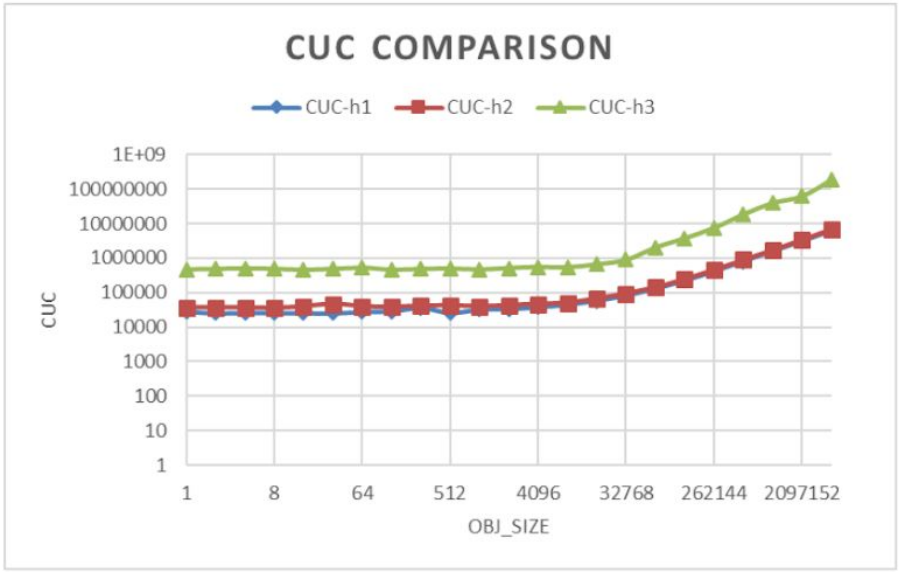


Fig. 18. Client CPU Consumption for the different protocols

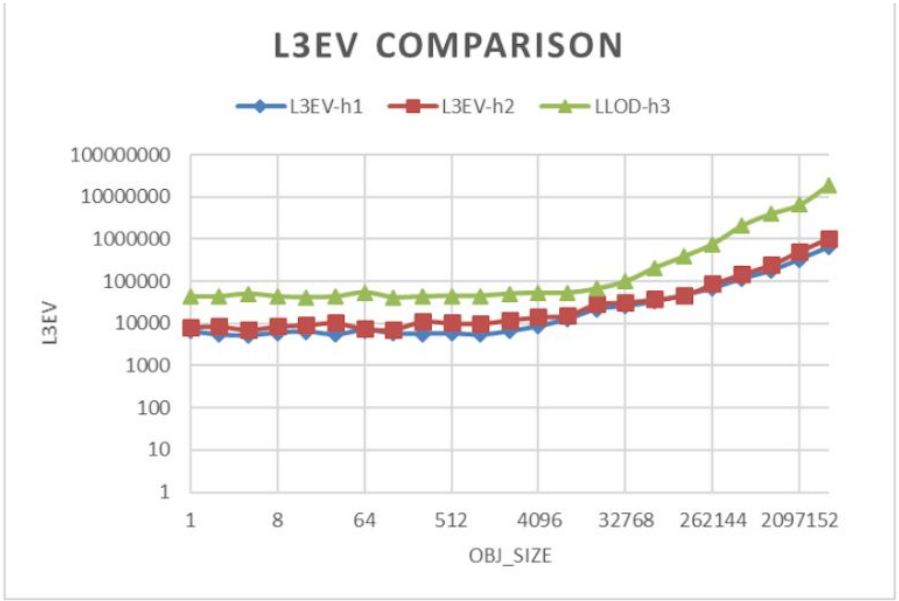


Fig. 19. Client Memory consumption for the different protocols

Conclusions

- An assessment of using REST as an IO protocol in an HPC environment
- A performance model based on hardware counters is provided and validated
- Same transport protocol is used, i.e., TCP, REST can provide similar latency and throughput to MPI while enabling better portability
- Newer versions of the HTTP protocol may have potential to improve performance
- Future Work: validate that REST is a performant and efficient alternative to common HPC I/O protocols in an actual HPC scenario
- Seamless convergence between HPC and Cloud
- Please discuss the paper in the JHPS incubator