Diskpool and cloud storage benchmarks used in IT-DSS

Geoffray ADDE

Outline

- I- A rational approach to storage systems evaluation
  - Hypothesis, metrics and experiments
  - Some examples
- II- Introduction to the benchmarking framework
  - Why? What? Where? Who? How?

I- A Rational Approach

A storage system

- Architecture (features, hardware, software)
- Physical Resources (nodes, CPU, RAM, HD, SSD, network, …)
- User Interface (POSIX, GridFTP, S3, ...)

A question

- What is the aggregated capacity of the system for a given operation? How does it scale?
- What is the impact of a given setting of the system on the system on a given operation?
- How the system behaves in degraded mode? during rolling upgrades?

An experiment:

- a workload
  - A set of clients (number, resources)
  - A set of tasks (read/write, small/big files, random/pattern/seq access, combinations)
- selected metrics
  - Host specific metrics (CPU, mem, IO, network)
  - Client specific metrics (request processing time, request througput, ...)

I- Metadata Read Performance

Storage System:

- 7 head nodes 2x10Gb fiber network
- 400 storage nodes, 700TB
- no encryption, 1 session per operation

Question:

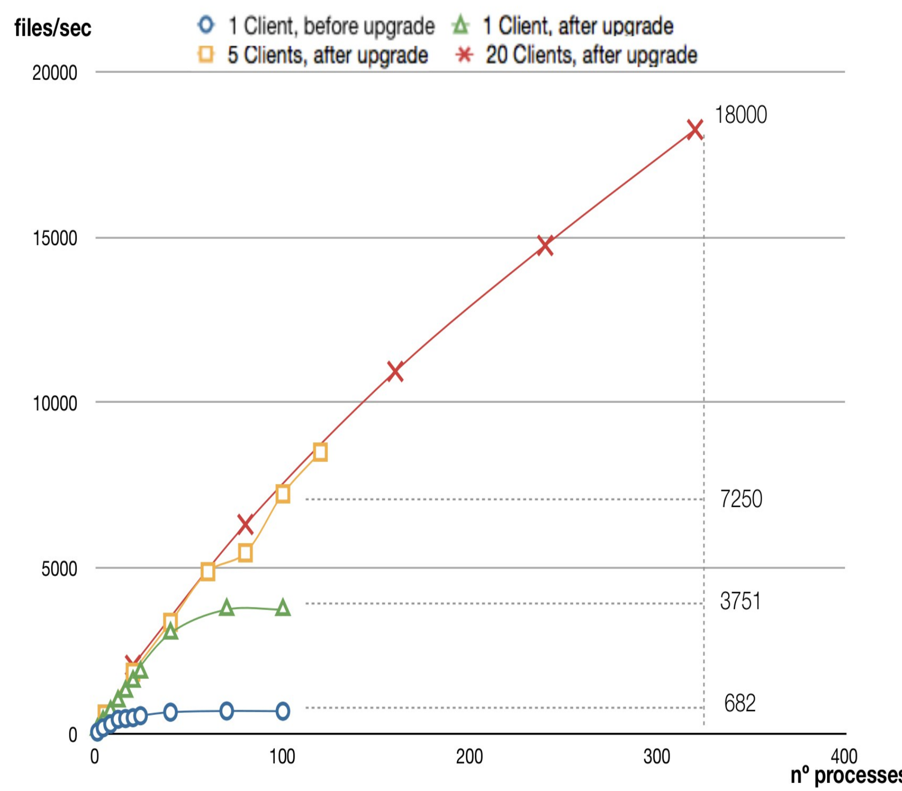
- How does the metadata read performance scale?

Experiment:

- 20 boxes 1Gb network 24 cores 48GB memory up to 20 processes/box
- One repeated task : read an entire randomly chosen 4k file
- Host specific metrics (CPU, mem, IO, network) check any client bound
- Counting the number of completed requests

Results:

- Scaling is almost linear
- Saturation of the metadata subsystem is not reached



I- Read Throughput

Storage System:

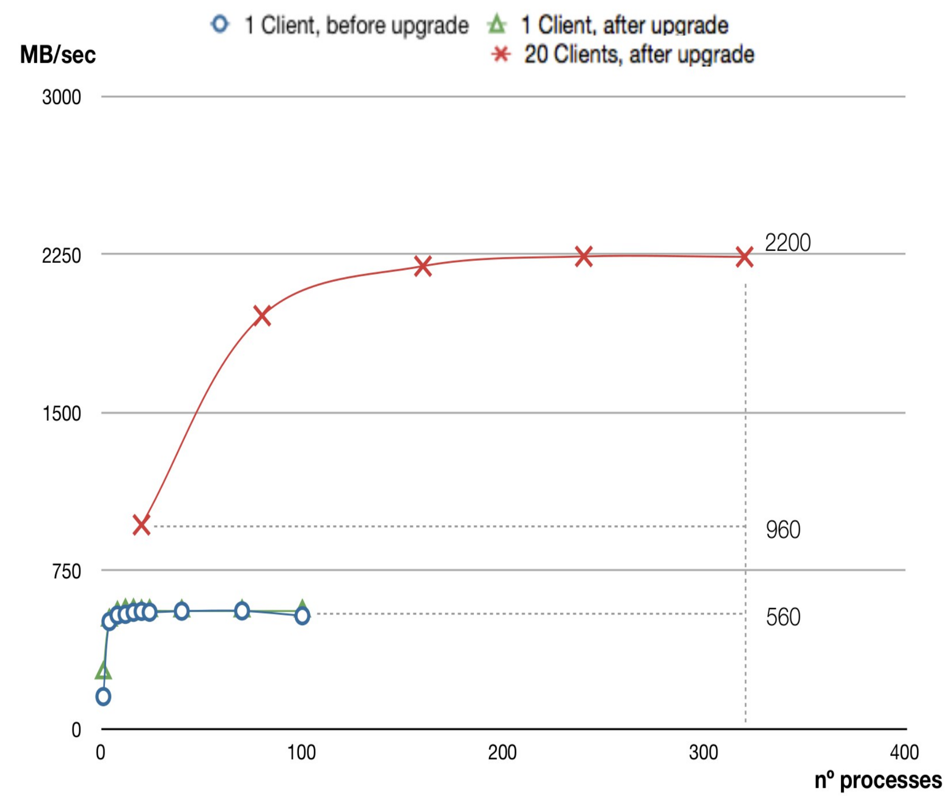
- 7 head nodes 2x10Gb fiber network
- 400 storage nodes, 700TB
- no encryption, 1 session per operation

Question:

- How does the read throughput scale?

Experiment:

- 20 boxes 1Gb network 24 cores 48GB memory up to 20 processes/box
- One repeated task : read an entire randomly chosen 100MB file
- Host specific metrics (CPU, mem, IO, network) to check any client bound
- number of completed requests, run time, network



Results:

- Scales linearly up to 80% of the max bandwidth
- Beyond it's still growing slower and slower and reaches the max at 240 processes.

I- S3 plug-in for ROOT

Storage System:

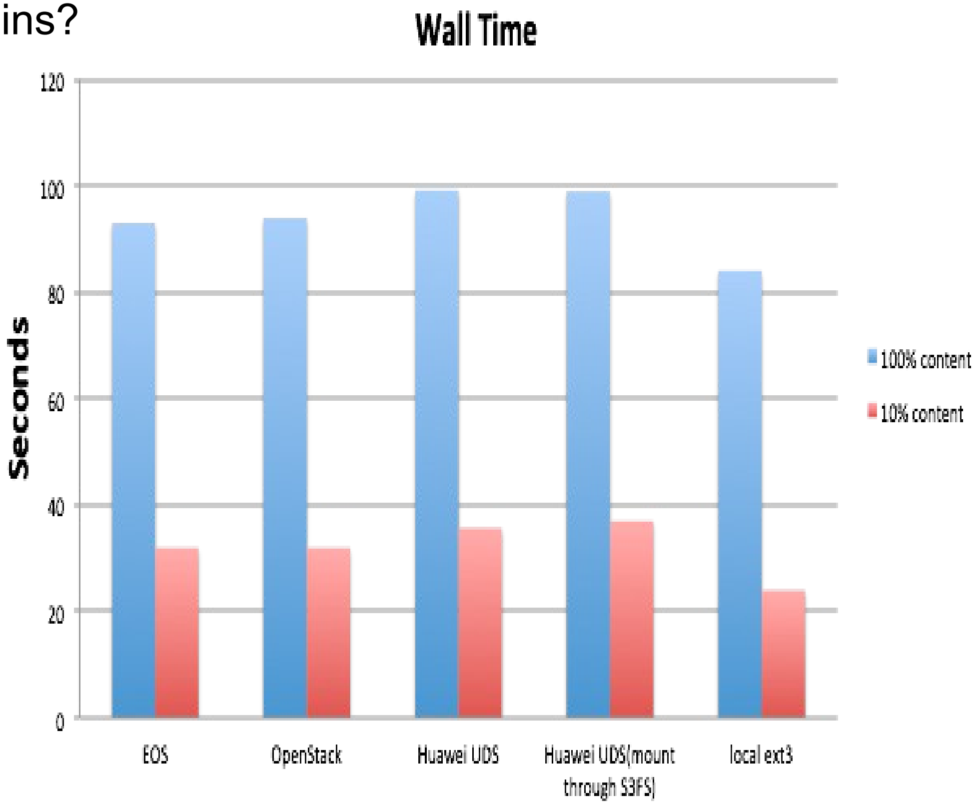
- 7 head nodes 2x10Gb fiber network
- 400 storage nodes, 700TB
- no encryption, 1 session per operation

Question:

- How fast is the ROOT S3 plugin compared to other storage plugins?

Experiment:

- 20 boxes 1Gb network 24 cores 48GB memory up to 20 processes/box. Idle
- One real ATLAS ROOT file : 793MB on disk, 2.11GB after decompression, 12K entries, 6K branches, cache size = 30MB
- One client reads in entries sequentially in “physics tree”
- Time to process the task. CPU client to check that the client is not CPU bounded.

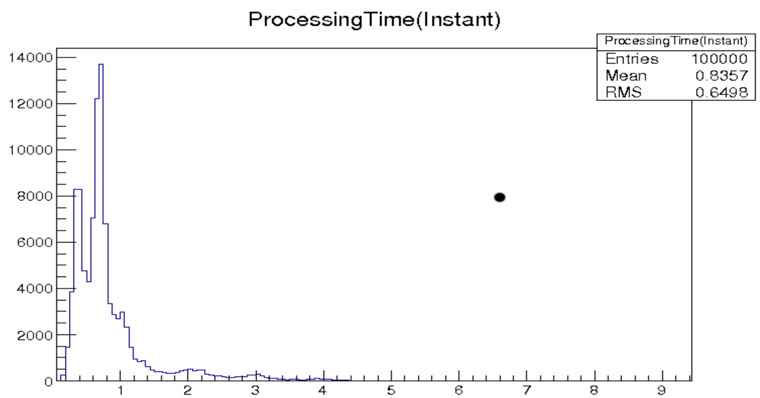


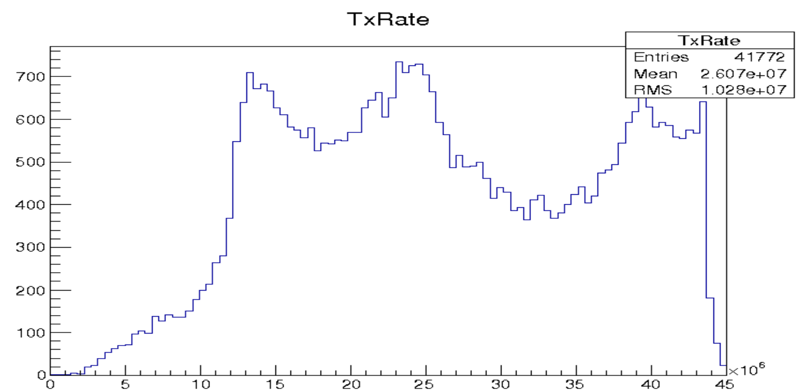
Results:

- Negligible gap of performance compared to EOS

Evaluation of a storage file system:

- Measuring a global quality of service
- Looking at aggregated distributions





- Host process-specific and machine-specific
  - CPU (User / Kernel / Wait)
  - RAM (mainly remaining RAM)
  - Network (transmit / receive, only machine-specific)
  - IO (local storage)

- Host process-specific and machine-specific
  - Request response time
  - Request throughput
  - Dedicated user-defined metric

- Dedicated metrics on the server side

- Network metrics

II- A tool for the evaluation of storage systems

Context:

- Metering the raw performances
- Evaluate the scalability(ies)
- Understanding the bottlenecks

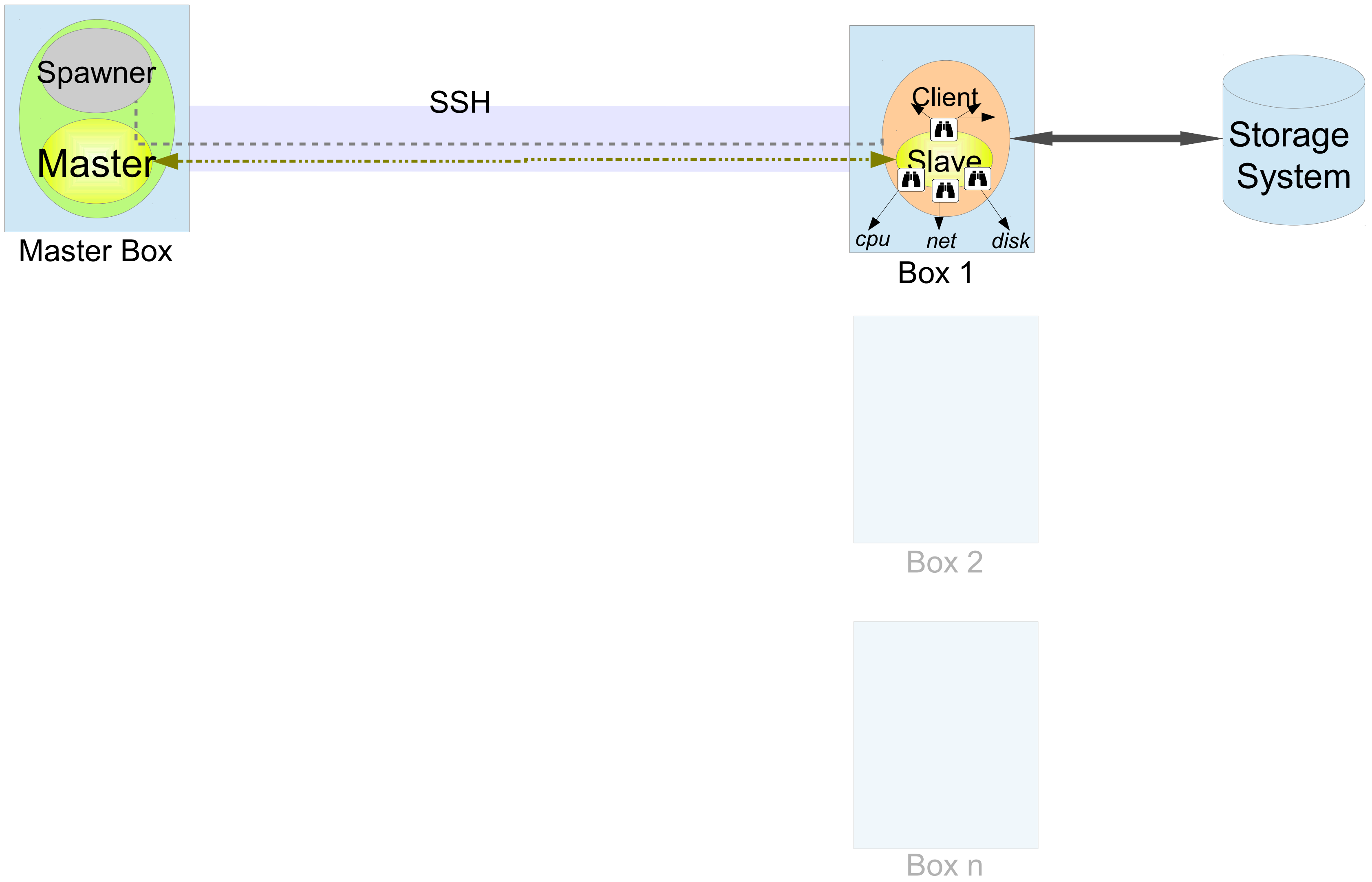
Need:

- Simulate the requests from a pool of clients (1~10000)
- Collect (if possible in real-time) metrics from these clients
- Don't interfere with this evaluation (low CPU/Net/Disk overhead)
- Ease of use (call, writing client, network, io)

Proposed Solution:

- A C++ client/server framework
- Based on SSH (Kerberos)
- Using Root as a way of compacting/transmitting/presenting the data

II- Framework OverView



II- Main Features

Architecture:

- SSH tunneling avoid any firewall issue (vs cyphering overhead)
- Timestamped data
- Built-in CPU, Network and Disk IO loads reporting (yet configurable)

Master side:

- Clean Management of remote processes (signal 2, 15 and 9)
- Real-time reporting
- Built-in graphs for histograms (1D and 2D) and plots
- Command line interface
- Some debug facilities

Remote side:

- One class, one instance, few methods.
- Can report
    - histograms or raw data
    - instant, elapsed, rate (./sec)
- A set of thread-safe methods and a SubId identifier to cope with multithreaded clients
- A Python wrapper is available

II- Status

Technical facts:

- requires root and boost
- build with cmake
- 4600 lines of code

Level of maturity:

- Successfully tested on Ubuntu x86, x86_64, SLC 6.x
- Daily run on SLC 6.x on 5 to 20 boxes at a 1Hz reporting rate

Concrete usage:

- Evaluation of Huawei S3 implementation (stress-test and ROOT)
- Evaluation of OpenStack

Perspectives:

- A web platform to collect measurements from many storage systems to build-up a benchmark.