Pre-GDB 2014

Infrastructure Analysis

Christian Nieke – IT-DSS

What do we want to do?

- Understand our systems
  - Which **parameters** influence the behaviour?
  - What are our **use cases**?
  - Which **metrics** can measure performance?
  - **Quantify**
- Improve
  - **Describe** our expectations and current status
  - **Detect** problems
  - **Understand** what causes them
- Predict
  - Future demand
  - Effect of system changes

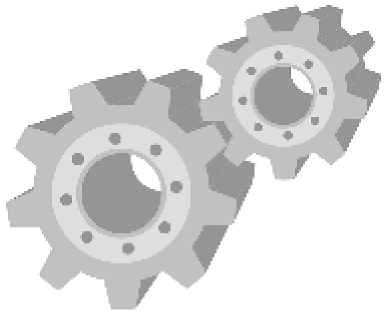
What do we have already?

- Data Sources
  - EOS logs per file access
    - some 20 metrics per open/close sequence
    - very similar to xroot f-Stream (collected by Matevz)
  - LSF - another 20 metrics per job
    - cpu, wall, os, virt/phys, RSS, swap, local I/O
  - Dashboards
    - experiment side classification and (in some cases) event rate

- But combining them is tricky…
  - Lack of unique identifier for cross matches
  - Combine information via xroot plugin?

Important parameters

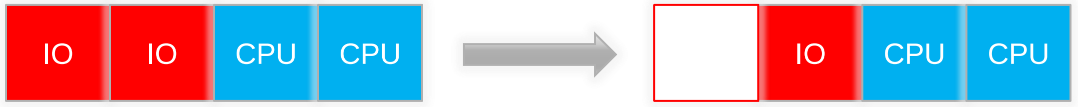
- Analysis and Effects of:
  - Remote/Federated access
    - e.g.CERN <-> Wigner
  - I/O & Network
  - CPU: AMD/Intel
  - Configuration
    - Vector Read/TtreeCache, OS etc.



- Use Cases
  - Analysis/Production
  - Per experiment
  - Internal (EOS-balancing etc.)
  - …

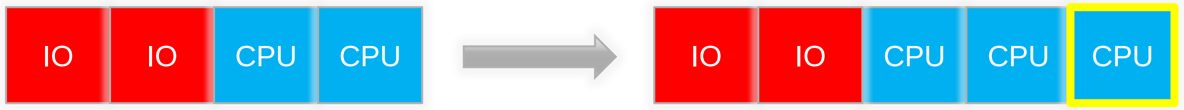
Metrics - How to measure performance?

- CPU-"efficiency"? ($^{CPU}/_{Wall}$)
  - Increases with better I/O throughput…



CPU efficiency: 50% CPU efficiency: 67%

  - … but also with slower CPU

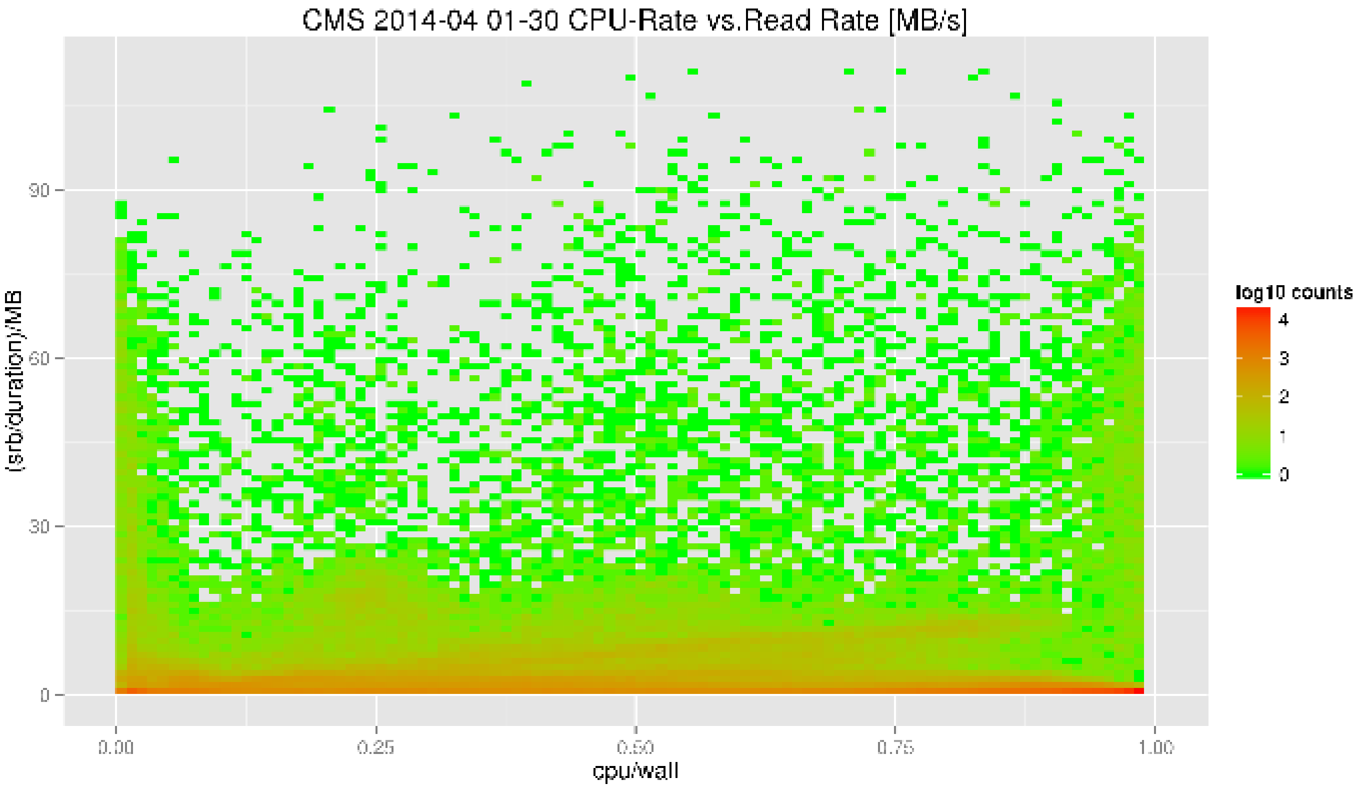


CPU efficiency: 50% CPU efficiency: 60%

- Still useful as an **indicator** for problems
  - But it should not be the only target for optimization
  - Better name: "CPU ratio"

Example: Throughput vs. CPU-Ratio

- The idea is that low CPU ratio points to slow I/O
  - But the relation is not that straightforward…

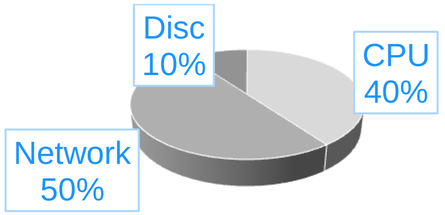


Metrics - How to measure performance?

- Runtime?
  - If the same job runs faster, this is obviously better
  - But **hard to compare**:
    - Different jobs
    - Different data sets (and sizes)
- Event rate?
  - **Intuitive:** Events / second
  - More **stable** than just time(ratios)
    - For the same kind of job
    - For similar data
  - Useful for **comparison** of parameters
    - E.g. CERN/Wigner for similar job mix
  - Right now **not reported** by all experiments

Metrics - How to measure performance?

- Job Statistics:
  - Collect key metrics (time spend in CPU, I/O and network transfer rate, event rate…) per job
  - Allows optimization for users and system
    - User: “50% time spent in network? I should check this! ”
    - System: “95% time spent in CPU? We should upgrade CPUs rather than switches.”



- Categorize into Job Profiles
  - Allow users to compare their jobs to the average in their category

Proposal for xroot

- Proposal:
  - add xroot client plugin to collect cpu and user metrics per session
  - Correlated (by session id) with f-stream
  - eg: cpu / wall / memory / swap / local IO
  - event rate from ROOT / exp. framework
- Define app info field for main workloads
  - Eg: app_info := “app/version/specific_setting”
  - EOS internally uses: /eos/gridftp, /eos/balancing, /eos/draining, etc
- This would allow **passive** analysis of experiment throughput per ‘app’
  - statistical comparison of sites/os/sw_versions
  - One would not be able to average evt/s over different apps
  - but one would be able to sum-up relative increases/decreases

How to improve the system?

- Model for expected behaviour
  - For now: Avg. values within a given group
  - Goal: Models for specific, well defined groups
    - E.g. Analysis/Production per Experiment, Internal(EOS-balancing)

- Detection of unexpected behaviour
  - For now: detection of outliers
  - Goal: automatically detect deviation from models

Example: Results based on simple detection

- LSF-Group:
  - Automatic detection of low CPU-Rate (<50%)
  - Users are informed personally
  - Experiments above 80%

Example: Users lacking information

- One example from new LSF low-CPU-usage probes
  - one user with often only 2-3% CPU utilisation
  - 60-100kB/s average transfer rate
  - CPU@CERN - all data at US-T2
- After vector-read/TTC was enabled
  - improvement by factor 4.5 in turn-around and CPU utilisation
  - remaining factor 6 wrt local EOS access
- Low “visible” impact on users (given sufficient batch slots)
  - even slow jobs are running stable in parallel
  - **no concrete expectation** about what their job duration / transfer speed should be

Summary

- Understanding the system:
  - Several parameters and use cases identified
  - Good metrics are still an open question
- Improving the system:
  - Currently semi-automatic detection of anomalies
  - Working towards higher automatisation
  - But already some success stories
- Prediction:
  - Still in the future…
- Proposal:
  - xRoot plugin for better integration of monitoring information