Evaluation of distributed file systems using trace and replay mechanism

Jiří Horký

horky@fzu.cz

Institute of Physics, AS CR, Prague (FZU CZ)

HEPiX 2011, GSI

Outline

- Motivation
- Choosing the right benchmark
- IOreplay – benchmark using trace-and-replay
- Evaluation of distributed filesystems
- Conclusion

Motivation

- Task: Compare the performance of file system/disk array A vs B relevantly for HEP experiments (e.g. tender requirements)
- Three possible options:
  - Run the real jobs
    - hard to setup, hard to only measure the disk performance
  - Use a synthetic benchmark
    - easy to run, but does it give relevant results?
  - Use trace and replay mechanism
    - again, does it give relevant results?

Trace-and-replay mechanism

- **Trace** - record all operations that affect storage performance (read, write, seek... + metadata operations – stat, access, mkdir)
- **Modify** (optional) – change delays between calls, ignore some operations, change file locations...
- **Replay** - replay the recorded operations back
  - with the same delays between individual calls
- **Theory:** as we perform the same disk operations (in the same order and time), we should get the same behavior as the original application

Choosing the right benchmark

- Does it really matter?
  - Real-life use case: comparison of two similar NASes:

Storage A (ARC1880ix-24)

- Intel Xeon E5620
- 12GB RAM
- Areca 1880ix-24 (2GB)
- 12x 2TB Seagate ES
- All drives in RAID6

Storage B (ARC1680ix-12)

- Intel Xeon E5620
- 12GB RAM
- Areca 1680ix-12 (2GB)
- 12x 2TB WD GP RE4
- All drives in RAID6

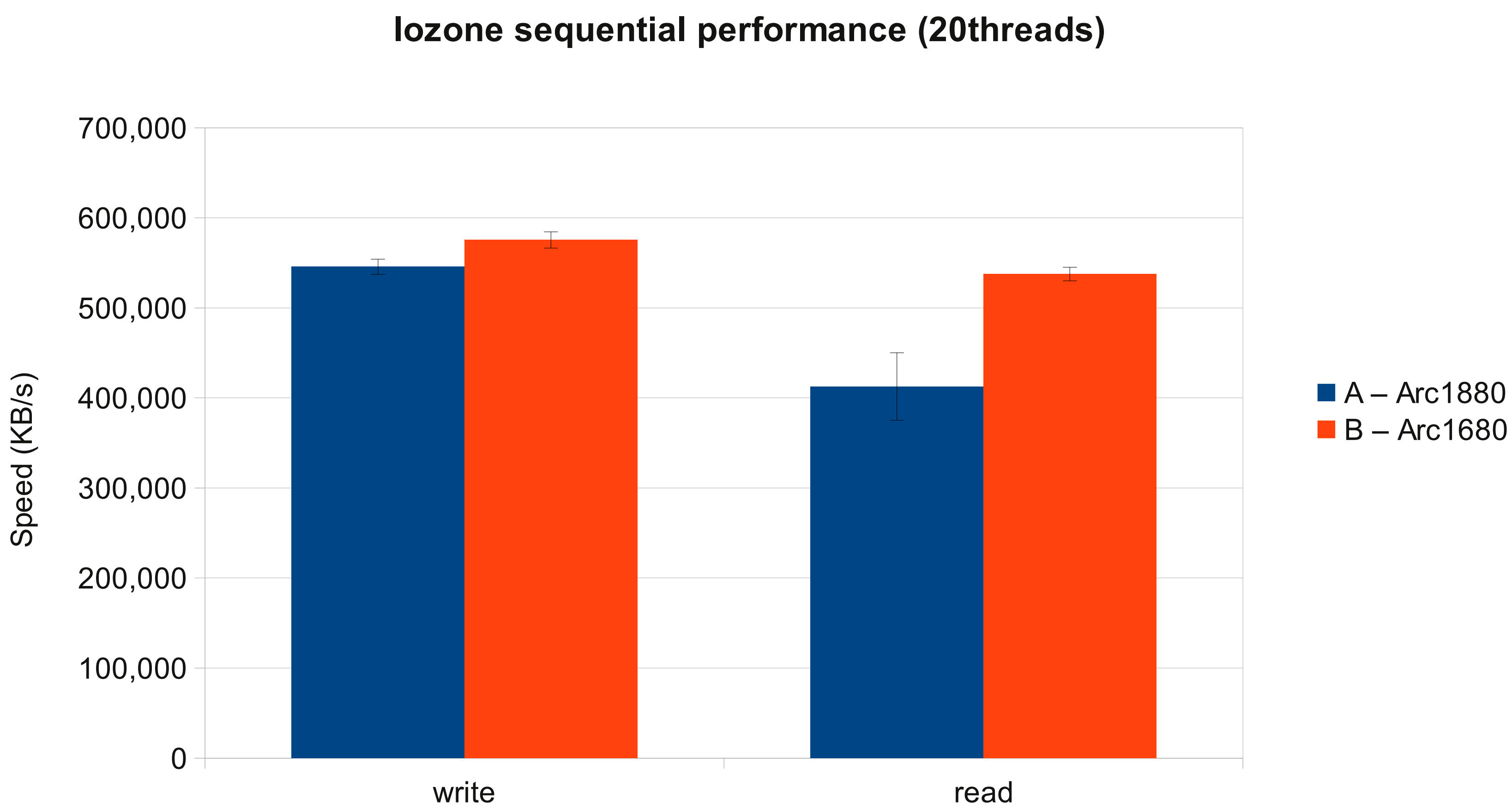
Choosing the right benchmark

- Two benchmarks:
- iozone – 1 thread per 1TB of usable capacity, each thread reads and writes sequentially 8GB:

```
iozone -Mce -t20 -s8g -r512k -i0 -i1 -F [FILES]
```

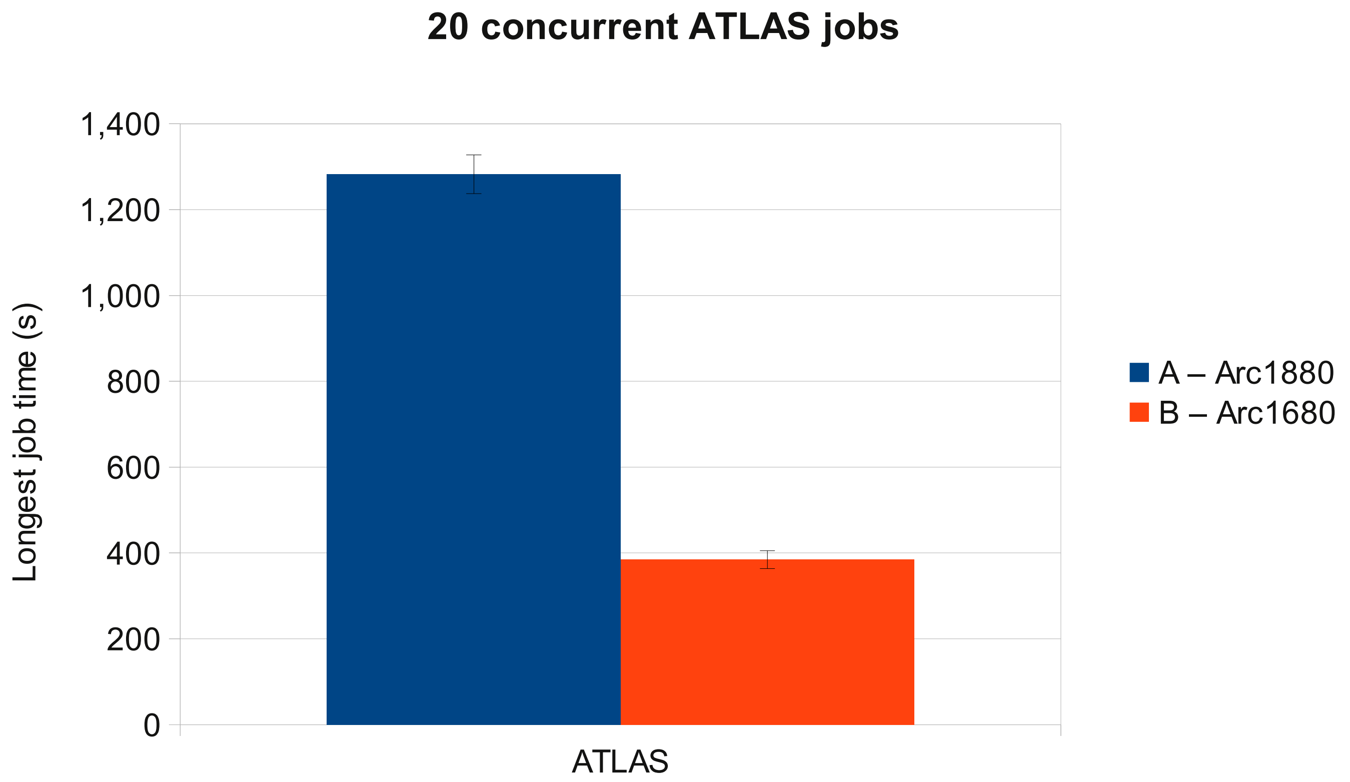
  (actual benchmark used for tender evaluation at FZU in 2010)
- real-life ATLAS analysis
  - 1 job per 1TB of usable capacity
  - strictly sequential, forward seeking

Choosing the right benchmark



Storage A with ARC1880 RAID controller aprox. 25% slower in read test

Choosing the right benchmark



Storage A with ARC1880 RAID controller aprox. **300%** slower !

IOreplay – benchmark using trace-and-replay

- **Trace**
  - several options possible
    - LD_PRELOAD, blktrace, systemtap, strace...
  - difference in the overhead, ease of use
- We decided to use `strace`
  - installed on virtually every Linux
  - works without root privileges, no modifications needed
  - ability to trace already running applications (strace -p)
  - considerable overhead at high system calls/sec rate
  - unable to record memory-mapped IOs

IOreplay – benchmark using trace-and-replay

- Example of strace output:

```
.....
1765 1279445178.319030 open("/etc/group", O_RDONLY) = 21 <0.000088>
11155 1279445178.319168 read(3, ""..., 10) = 10 <0.000081>
1765 1279445178.319261 read(21, ""..., 512) = 512 <0.000081>
1765 1279445178.320078 close(21) = 0 <0.000078>
```

- One has to keep mapping between file descriptor numbers and real files
  - across all traced processes
    - e.g. `21` == `/etc/group` at process 1765
- There are surprisingly many system calls that can modify it (`pipe, dup, socket, clone...`)

IOreplay – benchmark using trace-and-replay

- Problem: application traced on one server but we want to benchmark another one
  - with different mount points
  - with missing files
- One has to prepare the environment
  - accessed files should be at least of the same size
  - not every file is performance significant (files in /etc...)

IOreplay – benchmark using trace-and-replay

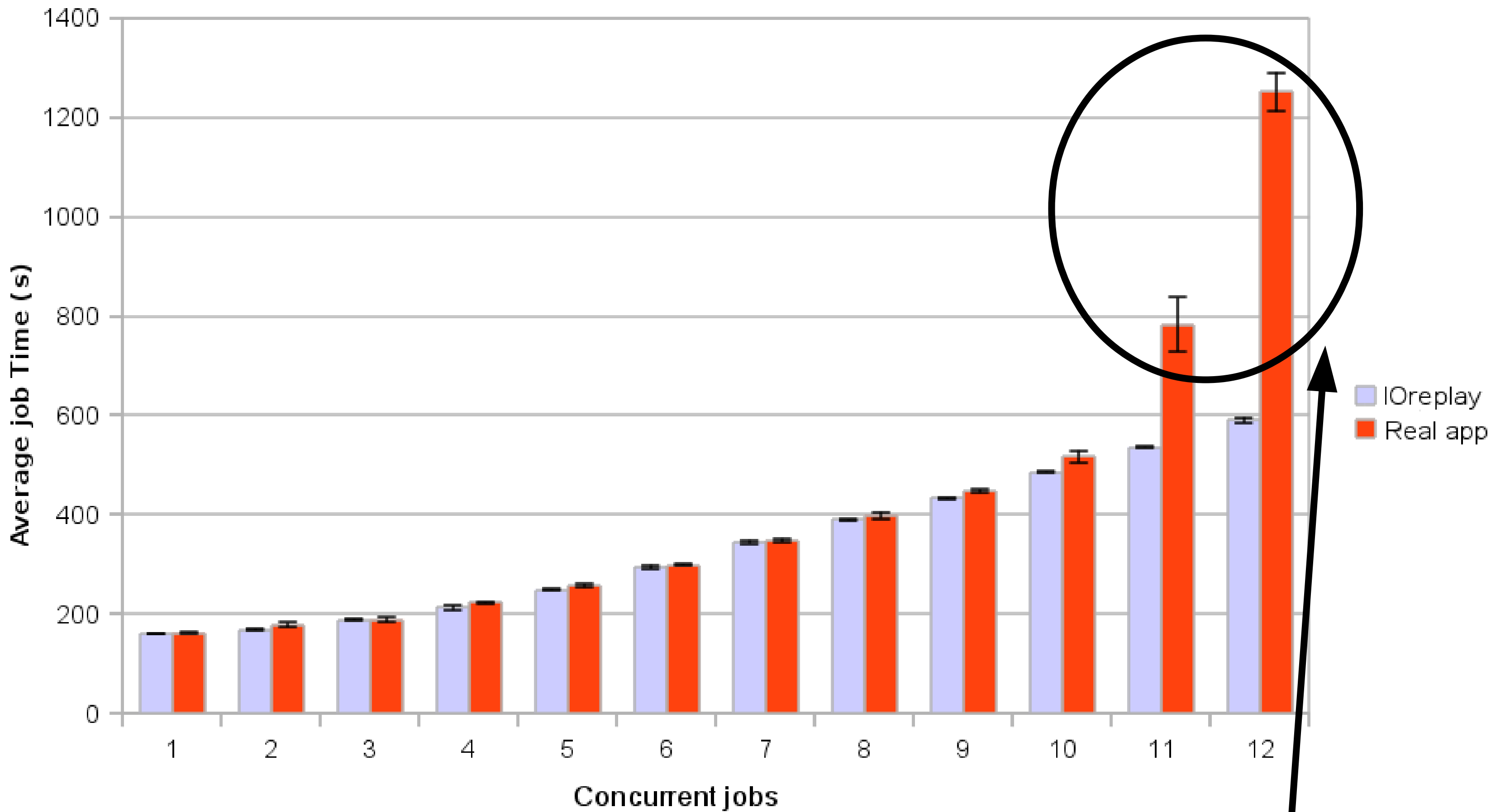
- IOreplay has measures to ease the preparation:
  - “dry” run that only reports missing files
  - ability to define files that should be **ignored**
  - ability to define **mapping** between original file name and file name on the machine being benchmarked

IOreplay – benchmark using trace-and-replay

- How fast to replay the calls?
- Multiple modes available in **`ioreplay`**
  - **diff -** keep the delays between calls (active waiting)
    - should give the same behavior as the original application
    - uses CPU instruction counter for fast time determination
  - **asap –** deliver the calls as fast as possible
    - absolute execution times differ

IOreplay – benchmark using trace-and-replay

- **Result:** realistic replaying and scaling



real ATLAS analysis job (ROOT 5.26 data format), run on 8-core machine

IOreplay – benchmark using trace-and-replay

- **Result:** realistic replaying and scaling
  - provided that recording (strace) didn't have high overhead
  - other aspects (usage of memory, network) can also have considerable impact

- You should always confirm it yourself

IOreplay – benchmark using trace-and-replay

Step by step usage

- Trace the job:

```
strace -q -a1 -s0 -f -ttttT -oTRACE_FILE -e
trace=file,desc,process,socket APP <PARAMS>
```

- Define files that should be ignored (e.g. don't access shared software area):

```
cat ignore.txt
    /software/atlas..../...
    /software/atlas..../...
    ...
```

- Define files that should be mapped:

```
cat mapping.txt
    /scratch/user...AANT1._00001.root   atlas/datafile.01
    /scratch/user...AANT1._00002.root   atlas/datafile.02
    /scratch/user...AANT1._00003.root   atlas/datafile.03
```

IOreplay – benchmark using trace-and-replay

Step by step usage

- Create zero-filled missing files (20-lines script):

```
./create-file-atlas.sh
```

- Run ioreplay:

```
./ioreplay -r -f TRACE_FILE -i ignore.data.only -m mapfile.data.only -t asap
```

Evaluation of distributed filesystems - methodology

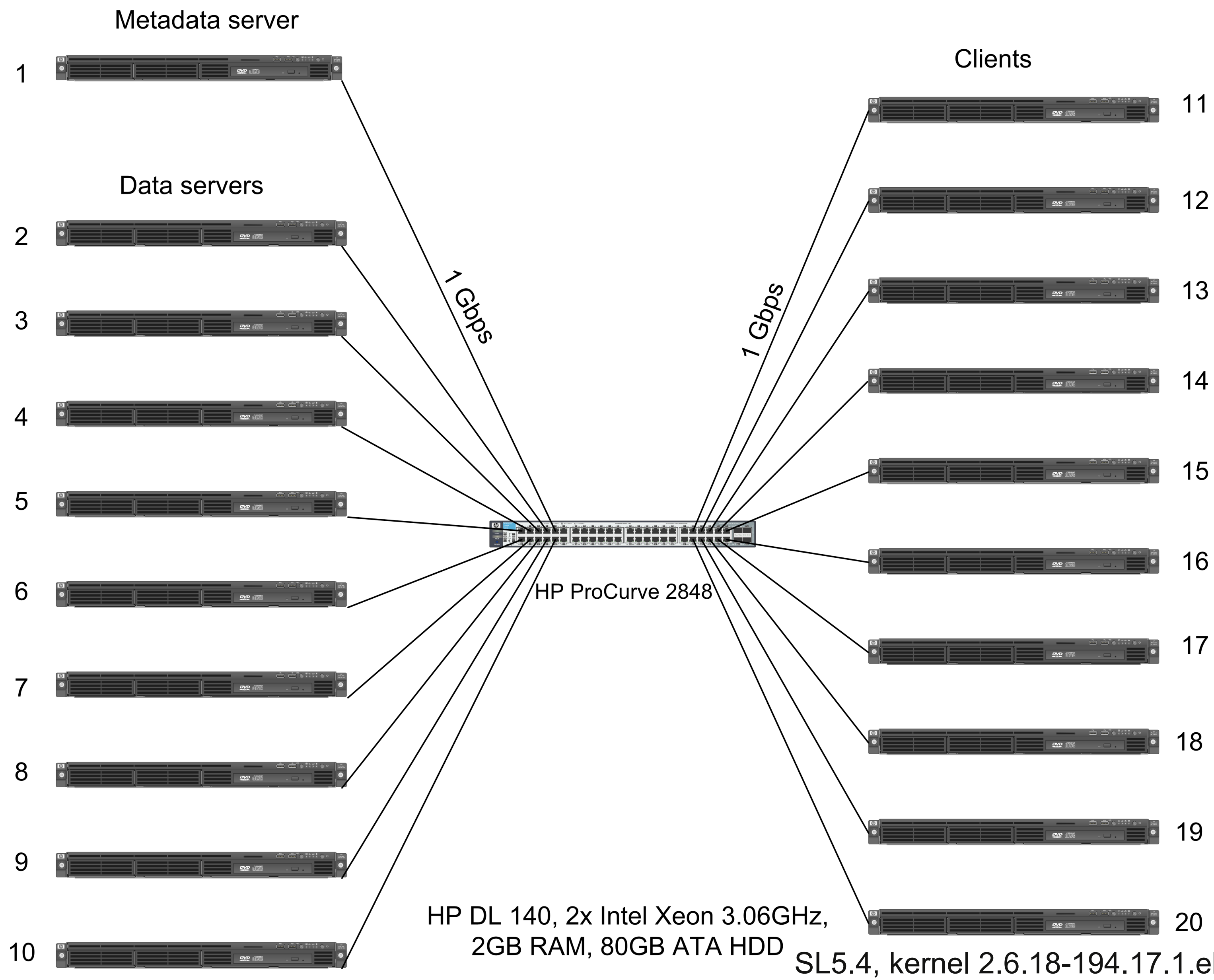
- Lustre v. 1.8.4, 100MB stripes, 3 servers per file
- HDFS (Hadoop) v0.21, 128MB blocks, using FUSE to provide file system layer, 2 replicas of all files
- GPFS v3.4.0.2, 256K blocks, 2metadata replicas, 1 data replica
- using IOreplay with 4 different real-life jobs
- running 1,2,4,8,10,20 concurrent instances
- measuring average job-time and network usage

Evaluation of distributed filesystems - methodology

ATLAS analysis, CMS reconstruction and analysis jobs

- **AtlasOld** – unordered ROOT files, 9x 250MB out of 1GB files read. The seeks were usually within few megabytes.
- **AtlasNew** – ordered ROOT files, 9x 250MB out of 1GB read. Strictly sequential with client caching.
- **CMSAn** - 1984MB read from 4GB file. Strictly sequential, caching, small gaps between individual calls.
- **CMSReco** - 424MB read from a 4GB file (just beginning of the file). Mostly sequential, backward seeking by ~30MB every 30MBs.

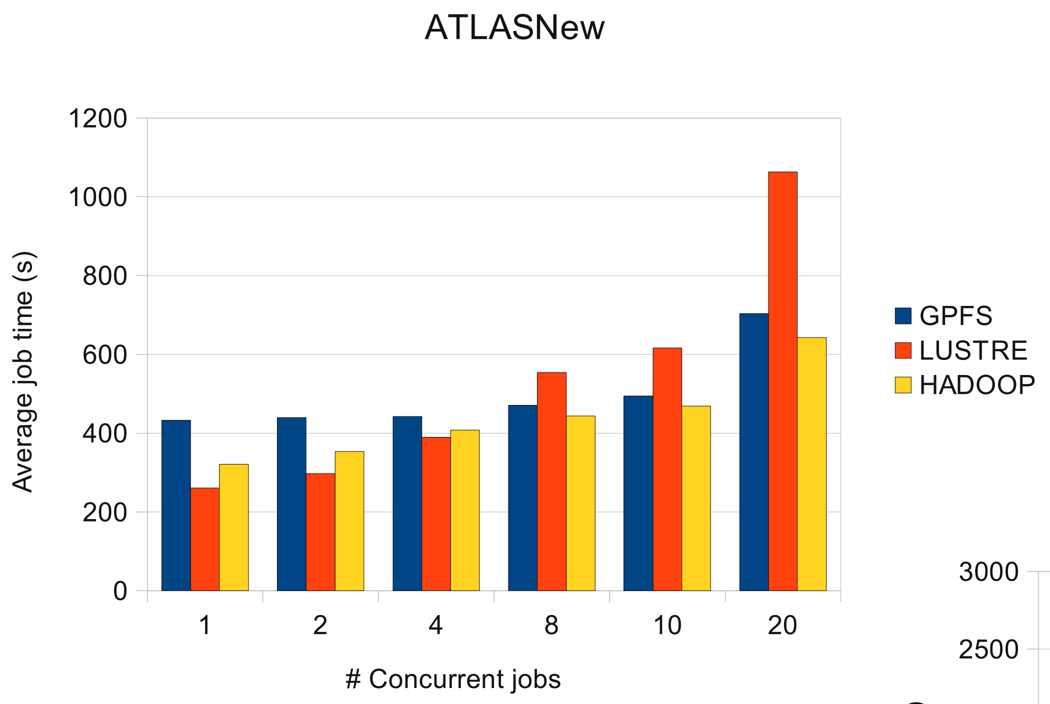
Testbed

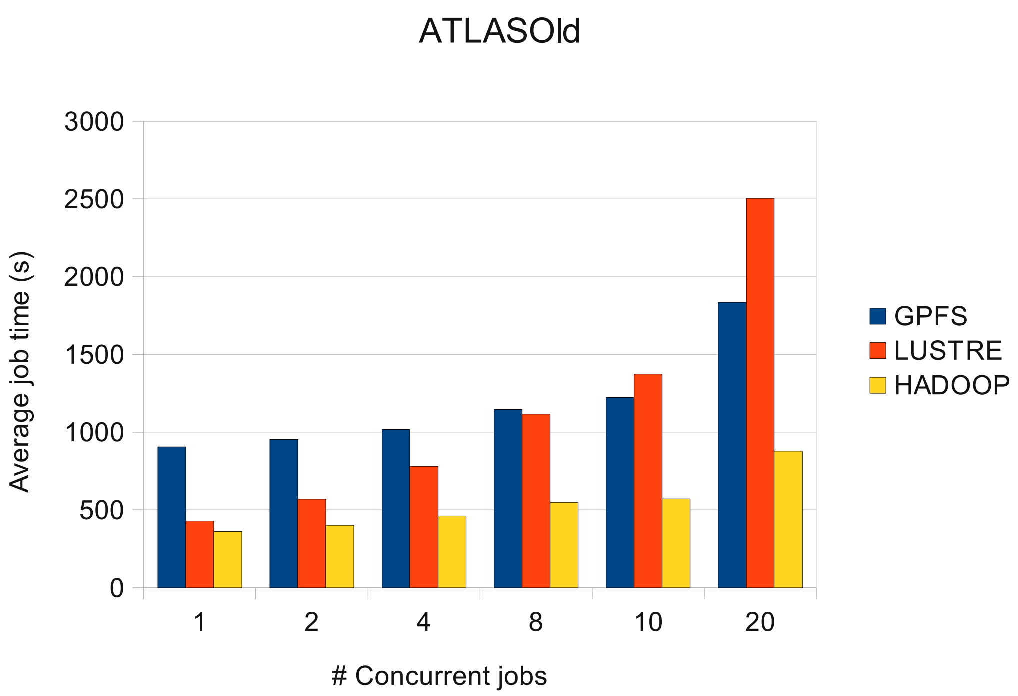


HP DL 140, 2x Intel Xeon 3.06GHz,
2GB RAM, 80GB ATA HDD

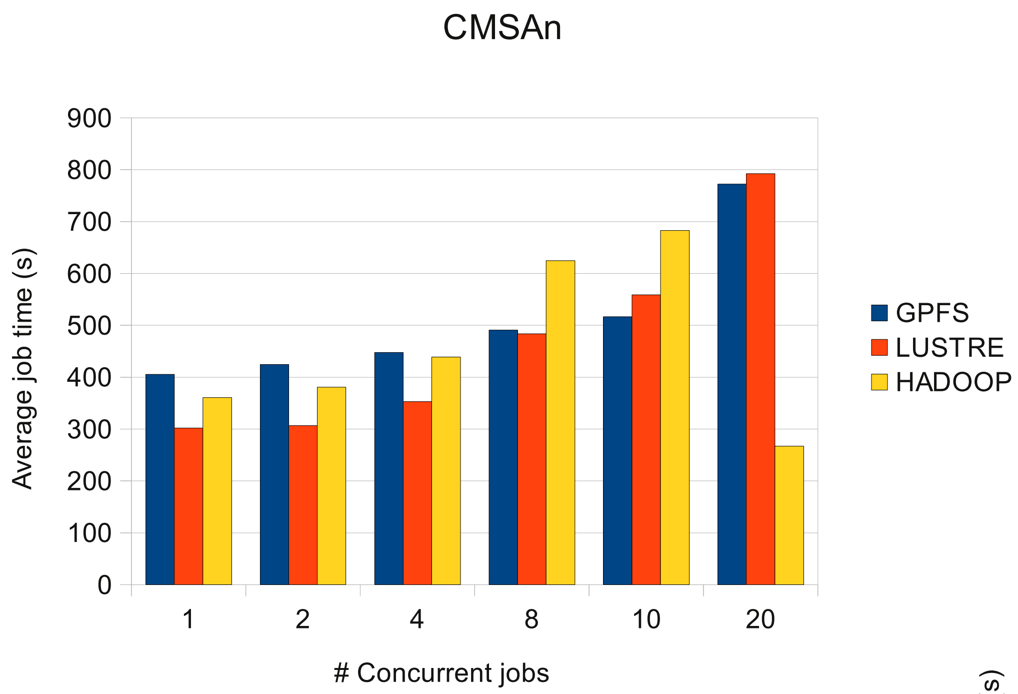
SL5.4, kernel 2.6.18-194.17.1.el5

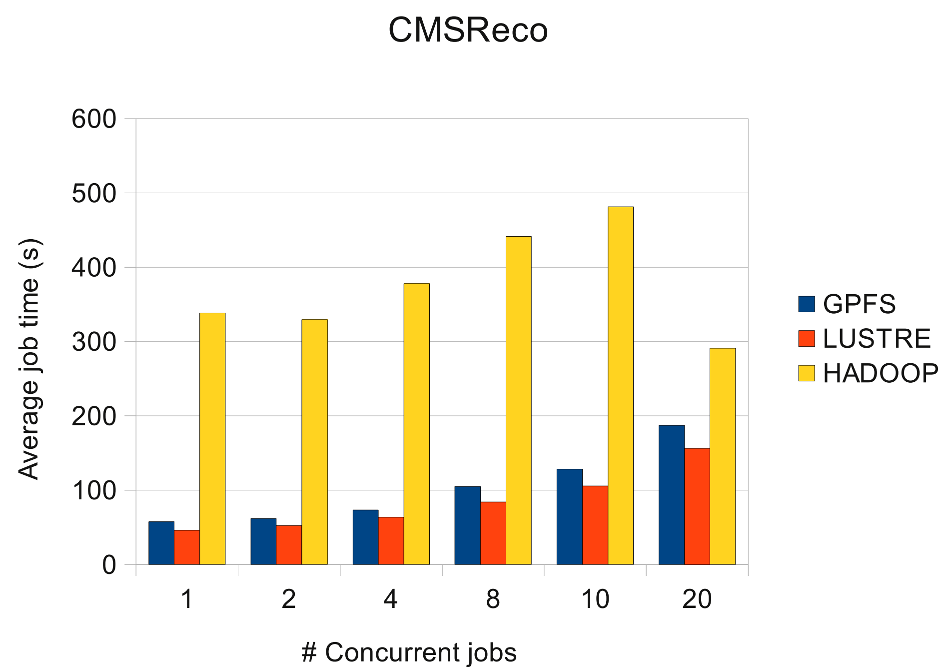
Evaluation of distributed filesystems



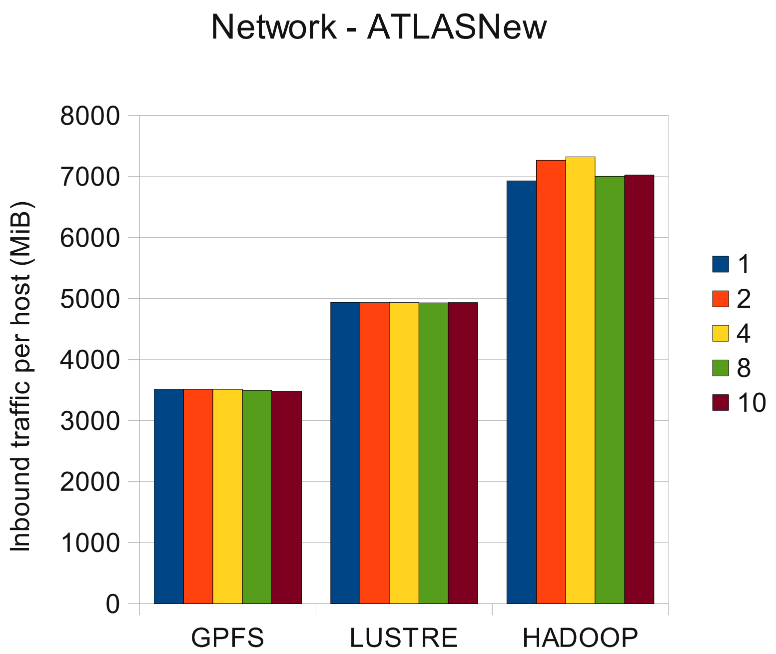


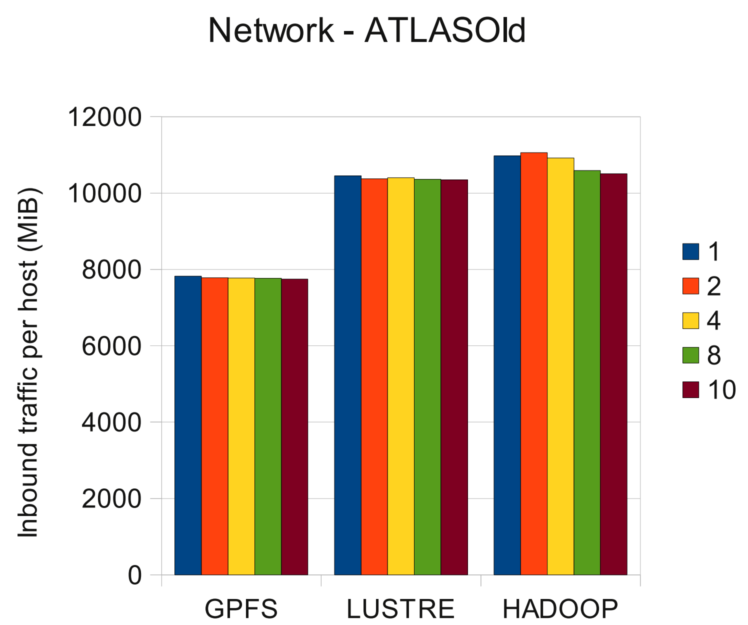
Evaluation of distributed filesystems



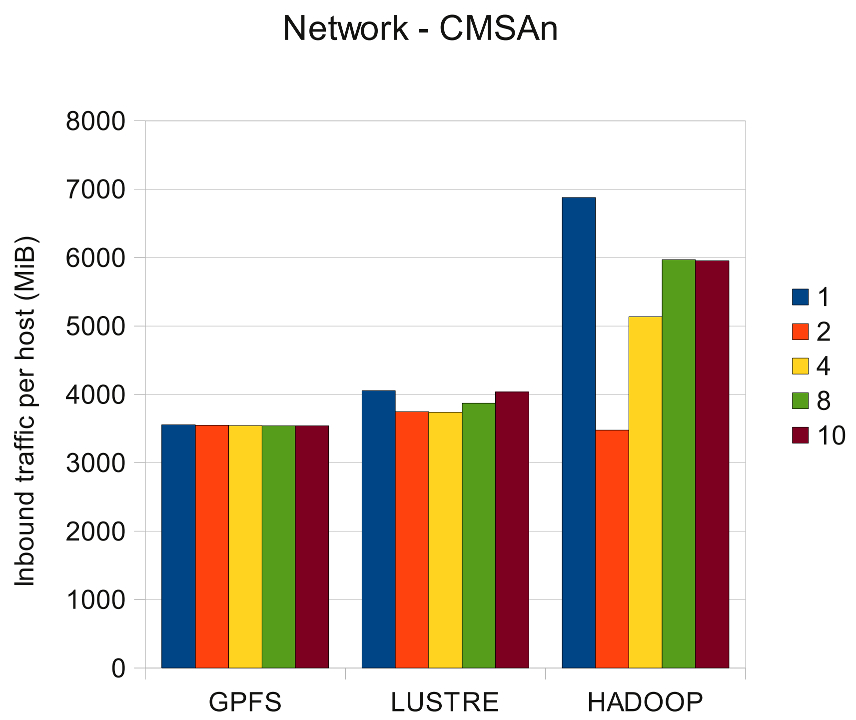


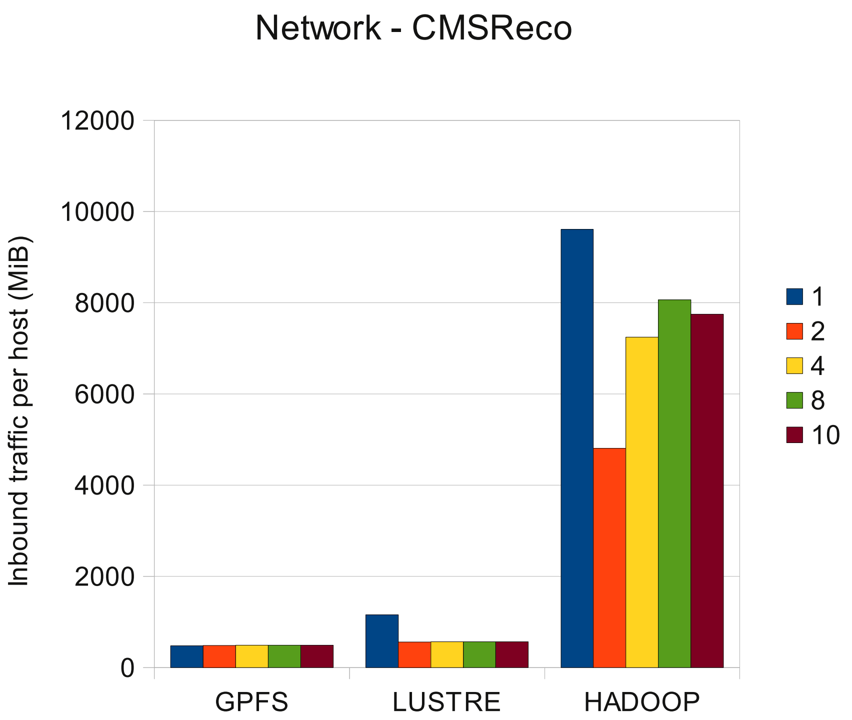
Evaluation of distributed filesystems





Evaluation of distributed filesystems





Conclusion

- No single 'silver bullet'
  - it really depends on application
- Hadoop seems to be the most network-demanding solution, GPFS the least one (block size advantage?)
- Hadoop works well with sequential access, but loses a lot with backward seeking
- Replaying of traces works and is fairly easy to setup
  - useful for a standalone (no dependencies) local access performance testing
  - useful for tenders - FZU will probably use it this year

Thank you for your attention!

Questions?

- The software is freely available at:

  http://code.google.com/p/ioapps/