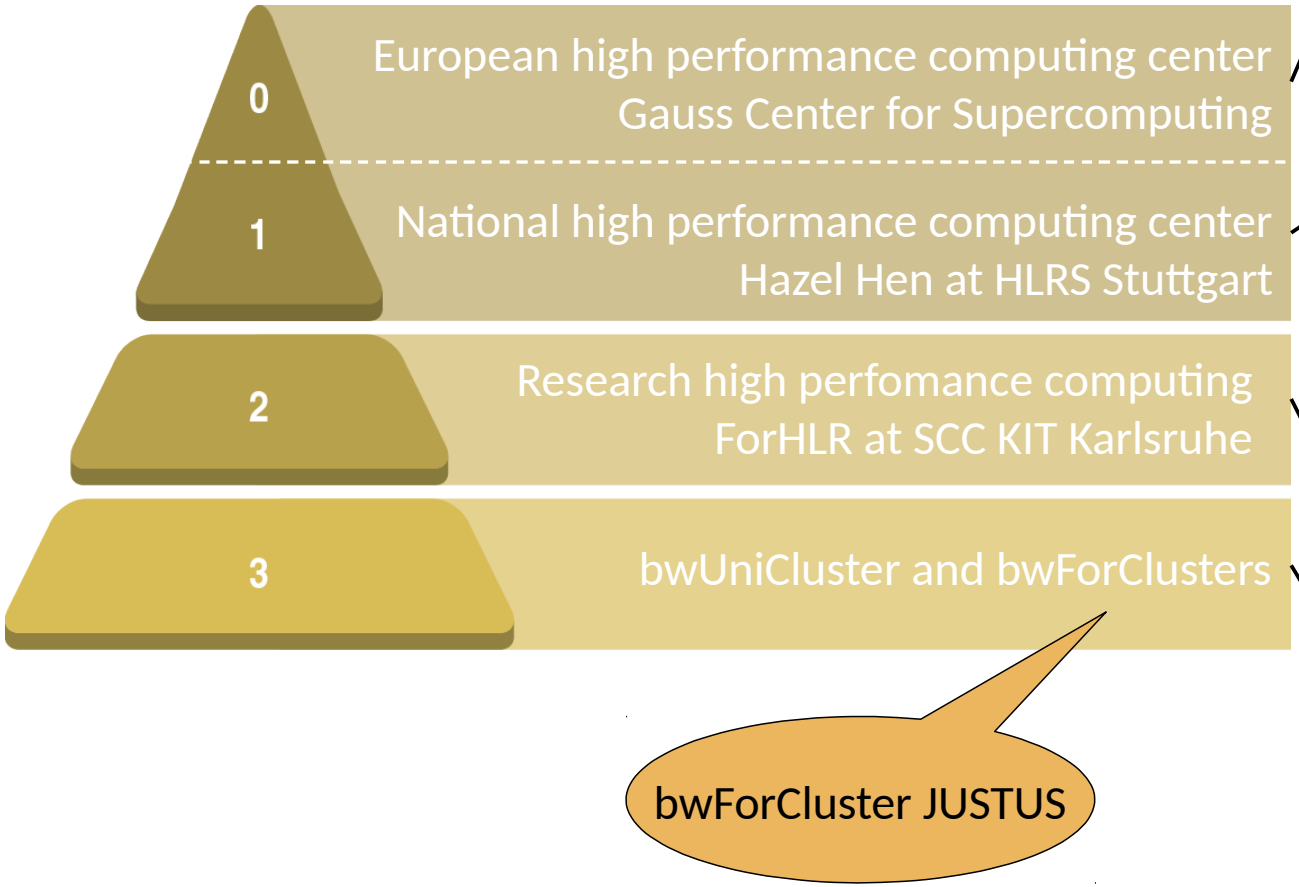
Motivation and Implementation of a Dynamic Remote Storage System for I/O demanding HPC Applications

Jürgen Salk, Christian Mosch, Matthias Neuer,
Karsten Siegmund, Volodymyr Kushnarenko, Stefan Kombrink,
Thomas Nau, Stefan Wesner, Holger Berger, Erich Focht

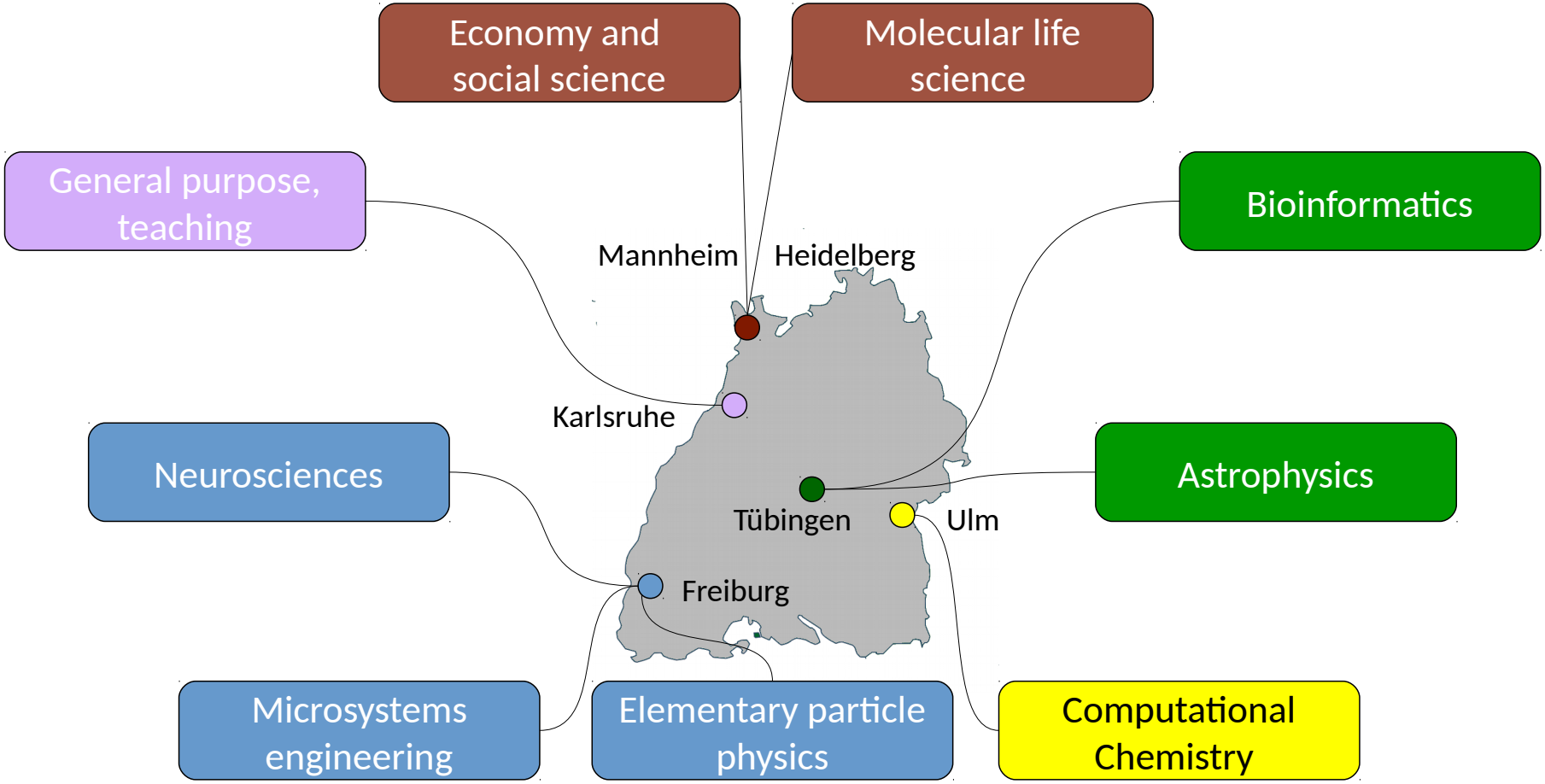
Founded by: Baden-Württemberg MINISTERIUM FÜR WISSENSCHAFT, FORSCHUNG UND KUNST DFG Deutsche Forschungsgemeinschaft

HPC Tier classification in Baden-Württemberg / bwHPC



- **European** High-performance Computing Centers (Tier 0)
- **National** High-performance Computing Centers (Tier 1)
- **Supraregional** state-wide HPC-Centers (Tier 2)
- **Regional** HPC-Enablers (Tier 3)

Introduction



Introduction

- Motivation: JUSTUS Cluster in Ulm for Computational Chemistry.
  Demand for high I/O for coupled cluster calculations (Molpro, Gaussian)

- Presented techniques not restricted to computational chemistry

- Deciding which I/O system is the best for specific application:
  How high is the I/O demand and what is the read / write pattern?

Tracing

iostat

- iostat gives many metrics for device usage
- Statistics are per block device and partition
- Useful: %util:
  Percentage of time the device spent doing I/O

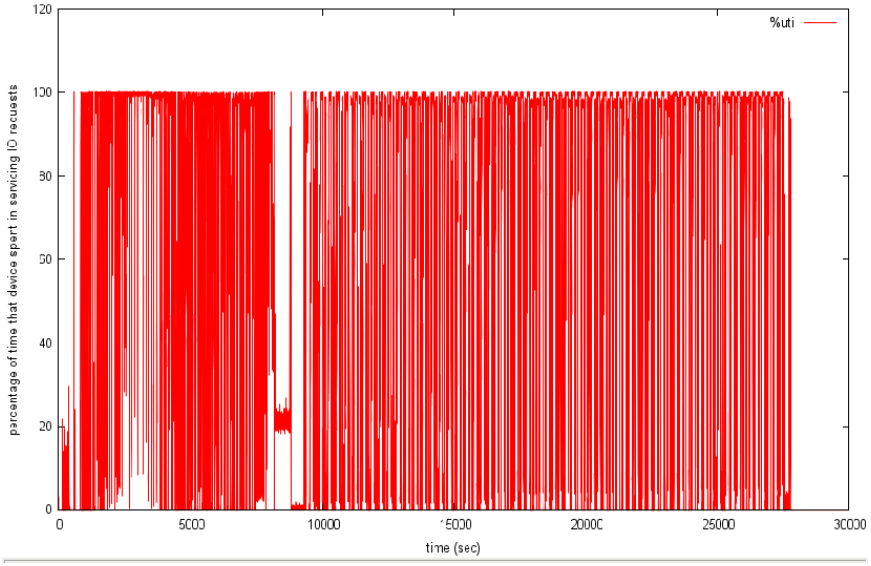
- `iostat -dkx 1`

```
-d     Display the device utilization report.
-k     Display statistics in kilobytes per second.
-x     Display extended statistics.
1      1 second interval between reports
```

iostat

- Watch out for caching effects:
- Same job

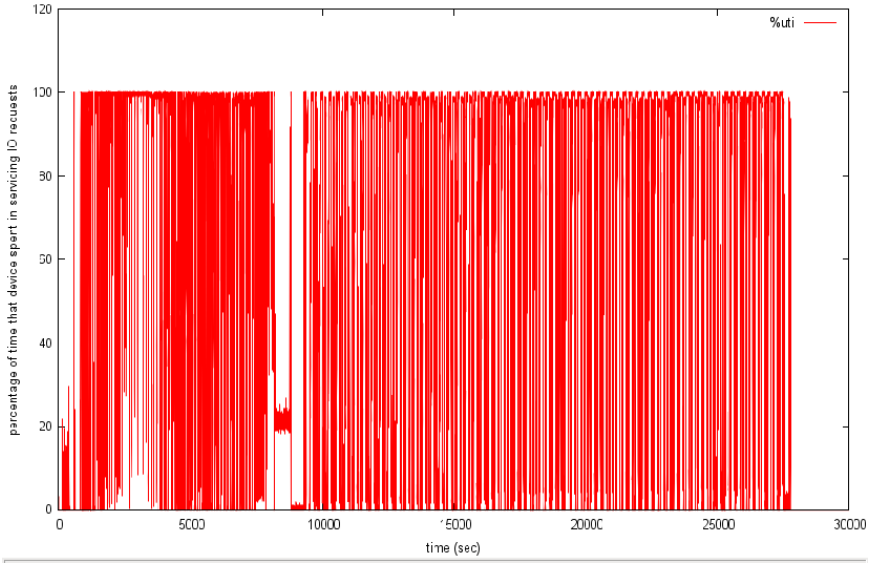
16 GB RAM



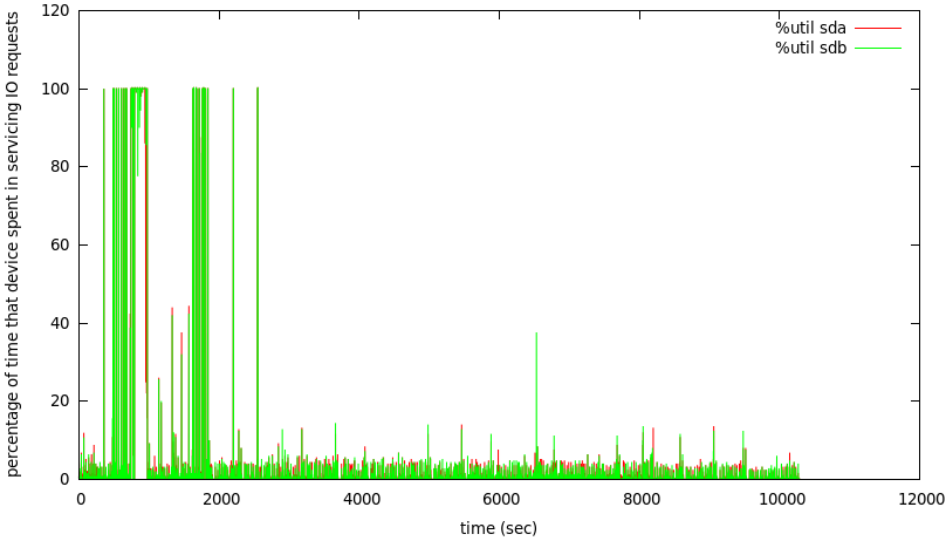
iostat

- Watch out for caching effects:
- Same job

16 GB RAM



48 GB RAM



strace

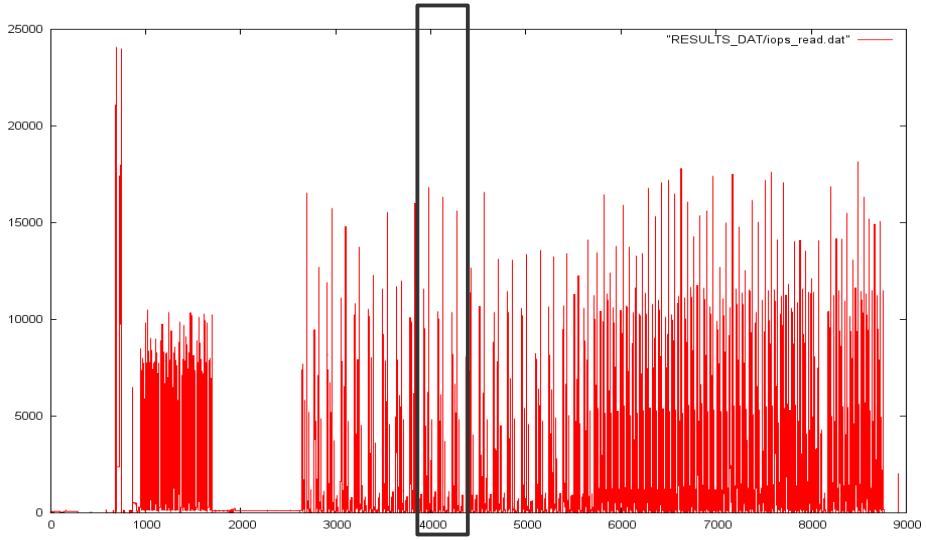
- strace provides insight from the applications point of view
- Intercepts and records system calls

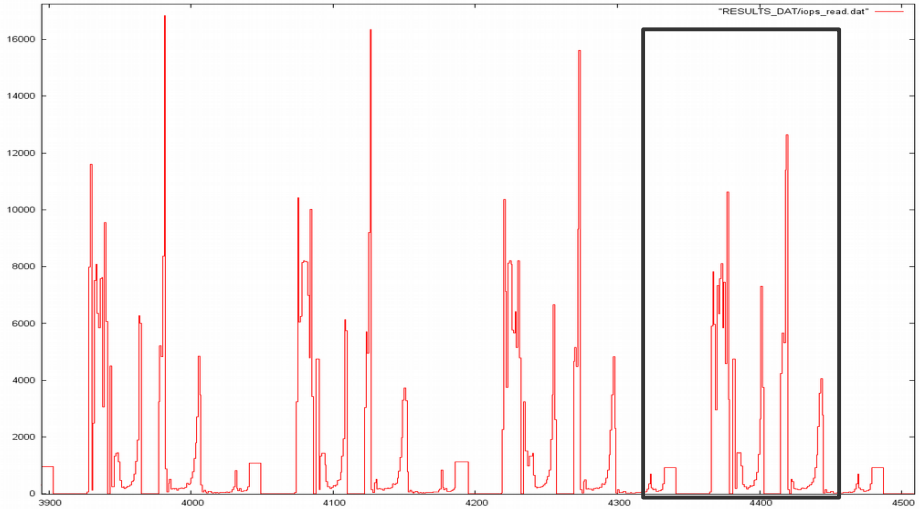
- `strace -T -ttt -f -e trace=file,desc -o trace.out prog`

```
-T                    report elapsed time
-ttt                  microseconds timing
-f                    include child processes
-e trace=file,desc    limit the trace to I/O system calls
-o                    write output  to file
```

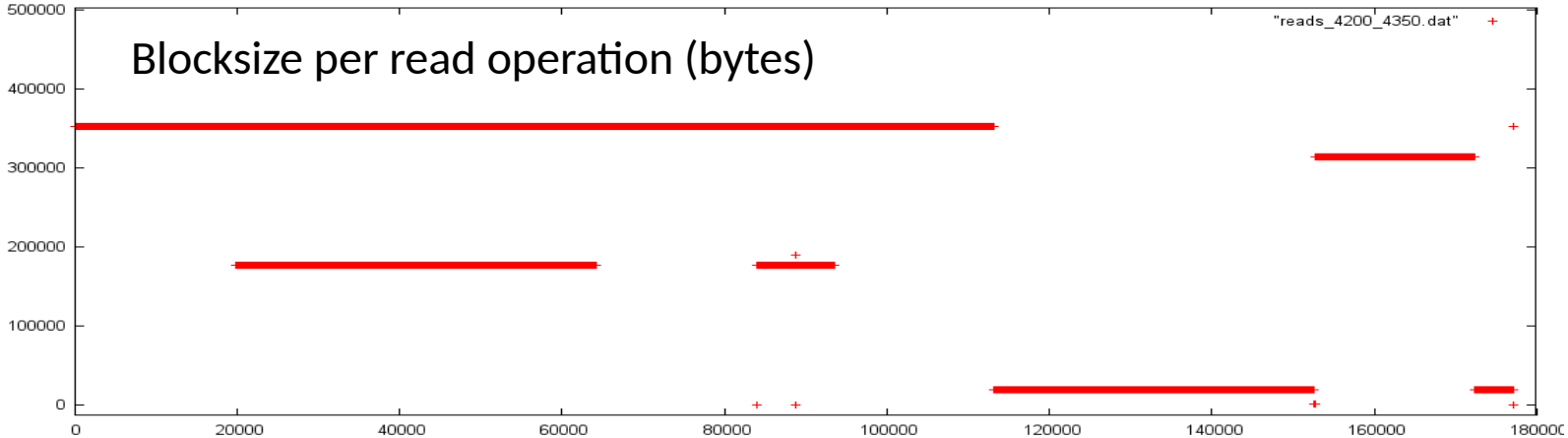
strace

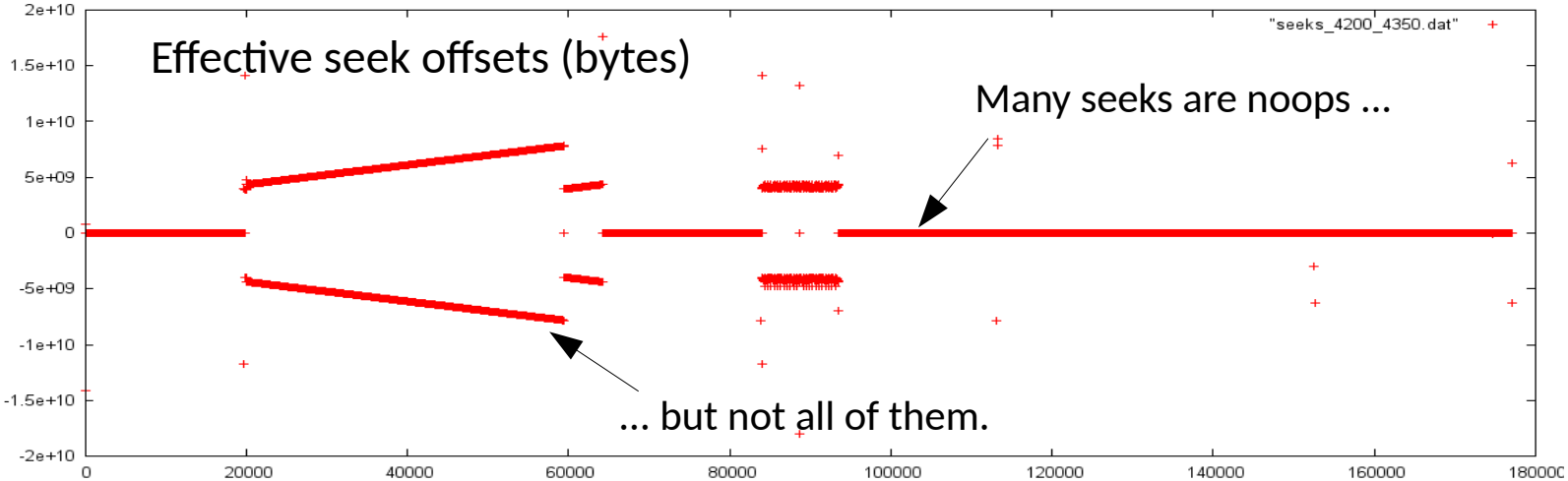
- Large output
- Post-processing and visualization via custom scripts
- Result example: IOPS over time, access pattern





strace


Blocksize per read operation (bytes)
"reads_4200_4350.dat"
500000
400000
300000
200000
100000
0
0
20000
40000
60000
80000
100000
120000
140000
160000
180000


Effective seek offsets (bytes)
"seeks_4200_4350.dat"
Many seeks are noops ...
... but not all of them.
2e+10
1.5e+10
1e+10
5e+09
0
-5e+09
-1e+10
-1.5e+10
-2e+10
0
20000
40000
60000
80000
100000
120000
140000
160000
180000

blktrace

- blktrace generates traces on blocklayer level
- Output is binary, can be converted to human-readable via blkparse

- `blktrace -d /dev/sda -o - | blkparse -i -`

```
-d    device to trace (multiple devices possible)
-o -  redirect output to stdout

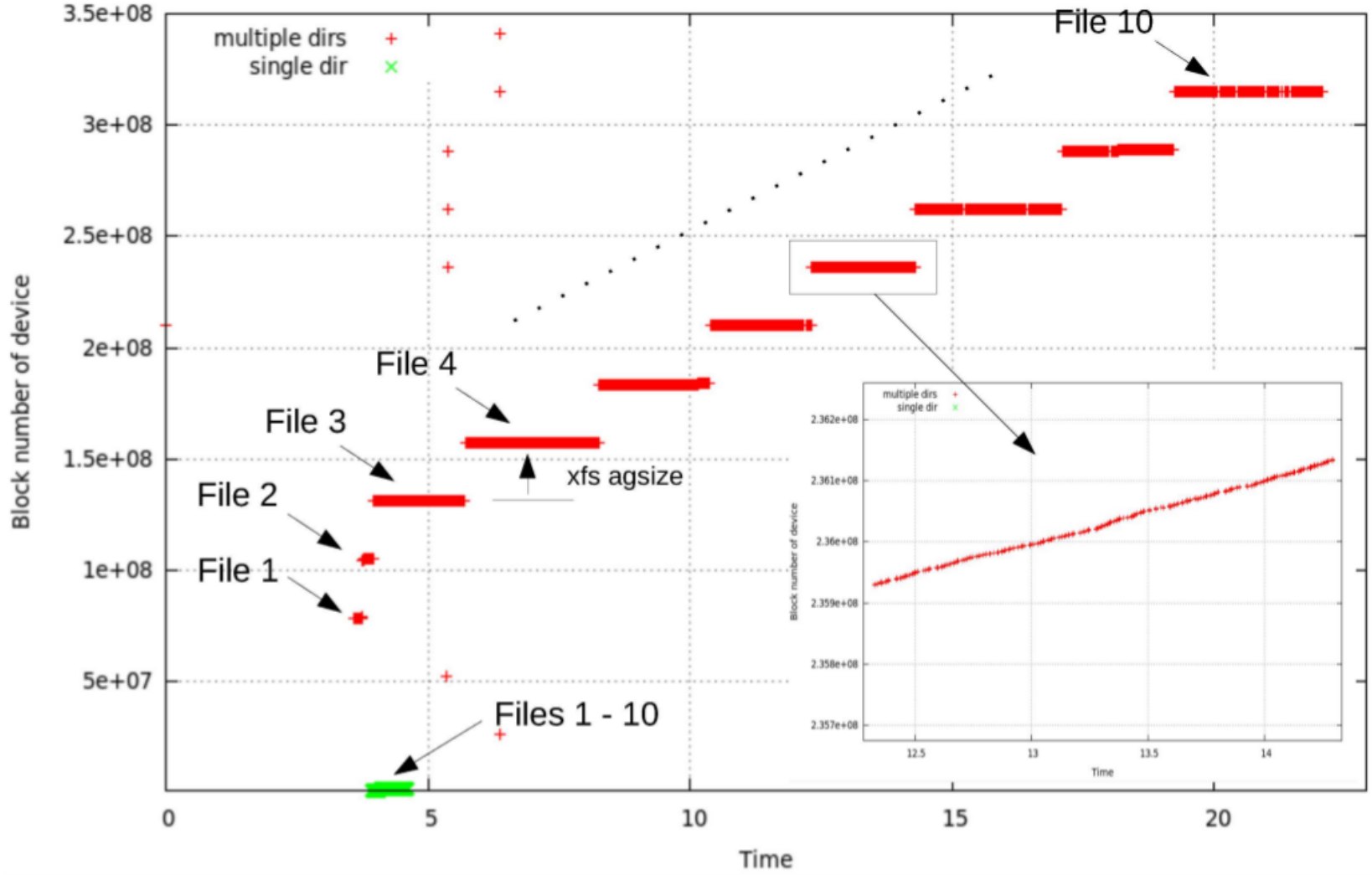
-i -  read from stdin
```

blktrace

- Prints device and accessed block number
- Useful for finding out access pattern on device level
  What are the effects of filesystem, raid, lvm, … ?

- Example:
- Used dd to write 10 test files (100 MB) one after another (no delete)
  - Test case 1: Write all files into a single shared folder
  - Test case 2: Write all files into individual sub folders
- Simultaneously run blktrace in “live mode”
- Record timestamp and location (as block numbers) of write operations on device
- Plot results for xfs filesystem

Results of blktrace analysis



Hybrid Storage Systems

Typical behavior of QC-jobs

- Informations gathered using the tracing methods:

- More read than write IOPs
  => read performance more important
  => cache-like solutions might be interesting

- IOPs beyond capabilities of typical hard disk
  => probably SSDs

- Coverage of current and (expected) future demands on scratch space purely by SSDs is too expensive
  => **hybrid (SSD + HDD)** solution economically reasonable for very large jobs

- Domains with and without I/O on each node
  => combination of central block storage (shared) and node local SSD
  => highest overall throughput

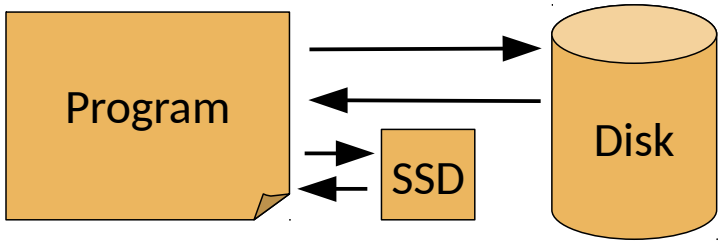
JUSTUS: Remote attached block storage

- **NEC SNA460/060 Storage Array** (+ extension unit)
  - 120 x 4 TB HDDs (Nearline SAS, 7.2k rpm)
  - 2 Controllers, each one with 2 IB QDR ports

- Access by means of **SCSI RDMA Protocol** (SRP)
  - higher throughput and lower latency than with TCP/IP communication protocol
  - Local OS page caching still available just like with local disks
  - No overhead introduced by cache coherency

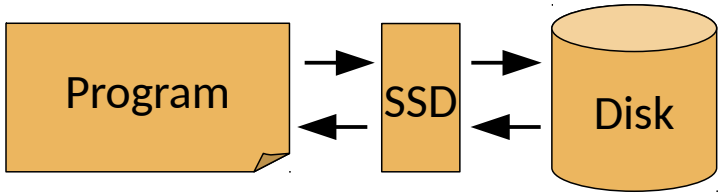
- **How to combine SSDs and remote attached storage?**

Plain Filesystem



- Separate use of SSD and remote scratch, two mount points
  - SSD filesystem for „hot“ files
  - Ideally: Program itself makes decision
  - Alternatively: Can be configured (e.g. Molpro), but
    - Coarse splitting policy
    - Needs a deep understanding of the algorithms used

Cache Based Approach



- Use local SSDs as **cache** for remote attached backend
  - Should work automatically

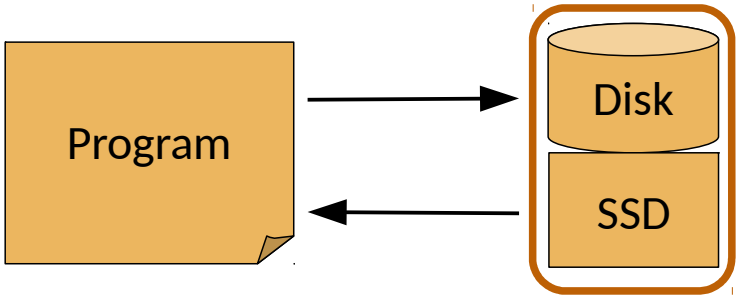
- Solutions:
  - Intel© CAS - Cache Acceleration SW
  - Flash Cache - Facebook
  - Bcache - Block layer Cache (Kernel > 3.10)

Cache Based Approach

- Comparison between solutions

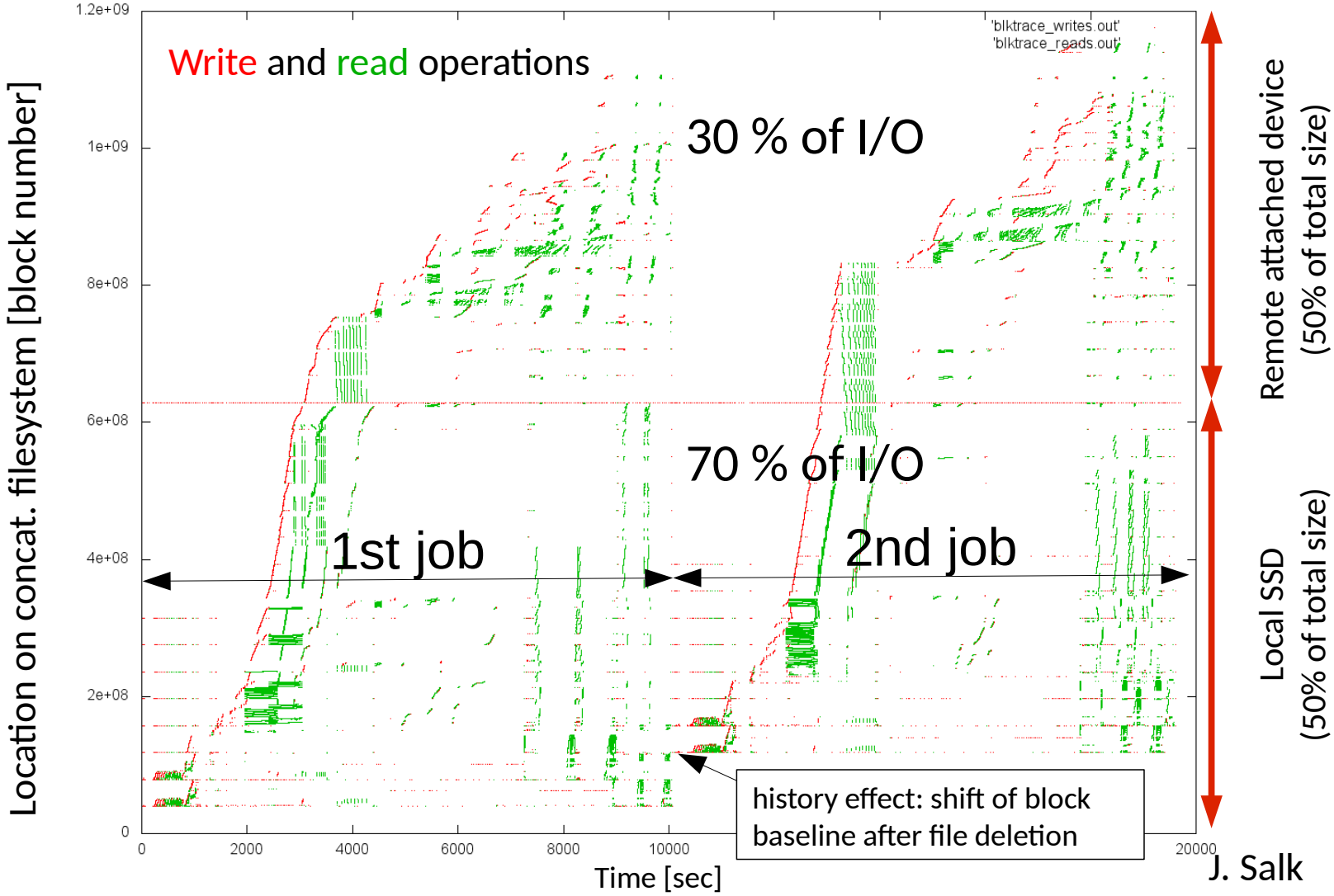
Name	Version	License	Performance	Footprint per GB cache
Intel CAS	2.8	Commercial/ GPL	Best	14 MB
Flashcache	3.1.3	GPL	90% of CAS	6 MB
bcache	1.0.8	GPL	90% of CAS	3 MB

Concatenated Storage System



- LVM – **concatenated hybrid FS**
  - SSD contributes to storage capacity
  - Write with preference for physical extents from SSD

blktrace of 2 * Molpro LCCSD-j2-c8
LVM compound of 3*RAID0-SSD + remote block storage



J. Salk

Concatenated Storage System

- Ensure that
  - Blocks with small numbers are placed on the fast storage system
  - Blocks with high numbers are placed on the slow storage system

- ```
  pvcreate /dev/ssd /dev/hdd
  vgcreate vg0 /dev/ssd /dev/hdd
  lvcreate -l 100%PVS -n lv0 vg0 /dev/ssd
  lvextend -L $LVSIZE /dev/vg0/lv0
  ```

- Choose the right filesystem:
  - ext4: doesn't fill from beginning after file deletion
  - xfs: scatter files when placed in different directories
  ```

Implementation

Implementation

- How it should work from a users perspective:

- `msub -l nodes=1:ppn=16 -l gres=scratch:100%drs_concat:5 job.sh`

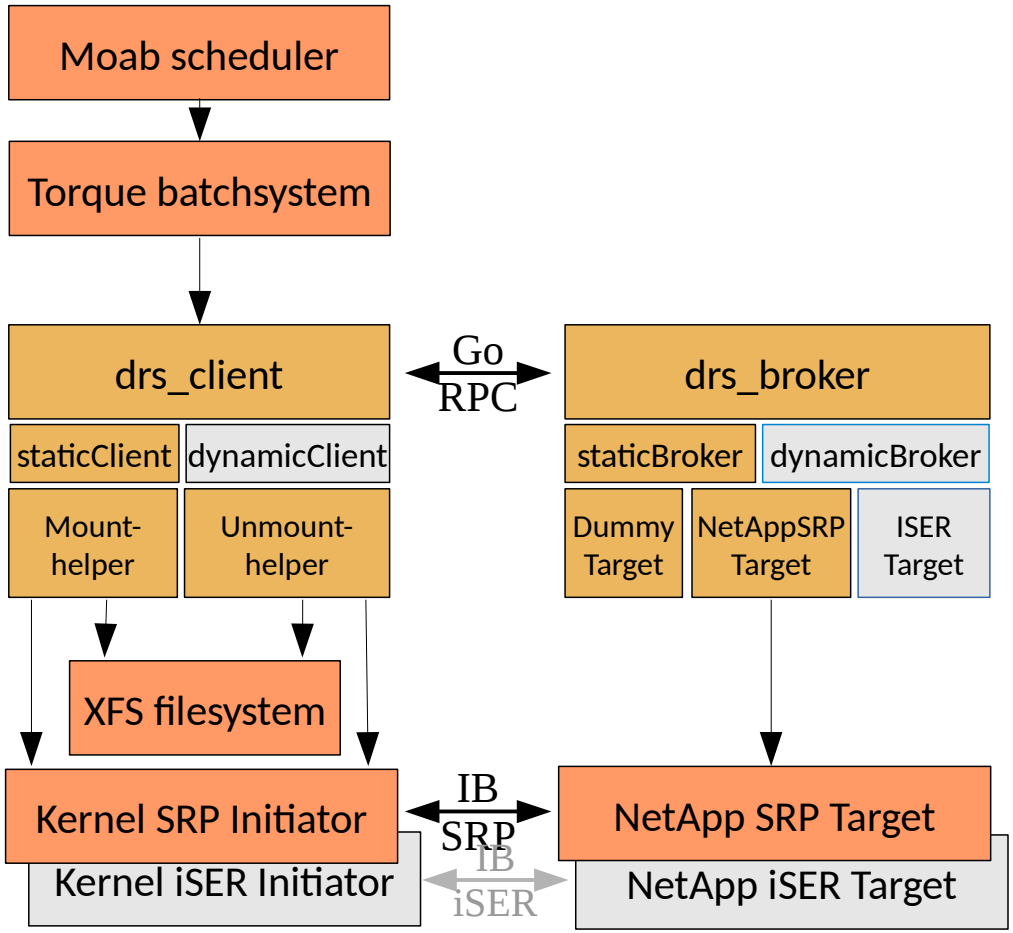
- Submits a job with MOAB which requests
  - 1 node
  - 16 processes per node
  - 100 GB SSD space per process
  - 5 TB space from remote attached storage
  - Combine the two storage systems via lvm

- When job starts there is a mountpoint with 6.6 TB storage space

Implementation

- Components

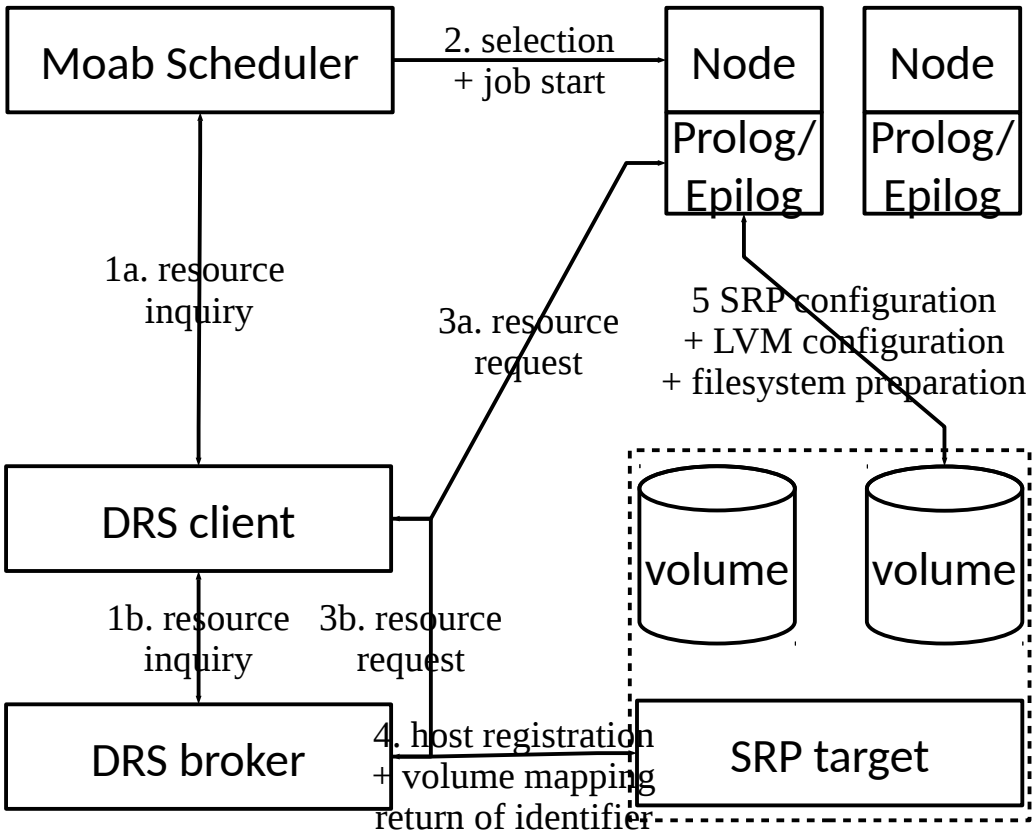
- **Moab**: job scheduler
- **Torque**: resource manager
- **drs_client**: used for sending queries and commands to drs_broker
- **drs_broker**: keeps track of remote storage resource allocations and configures the storage targets



Implementation

- Workflow

- **1**: ask if resources are available
- **2**: start job
- **3**: request the volumes
- **4**: create volume, register host
- **5**: create lvm- or cache-device, create filesystem, mount



Moab Implementation

- DRS volumes are treated as **shared (floating) cluster resource**
  - Similar to integrating floating software licenses in scheduler
  - User Moab‘s existing interface for external FLEXlm license managers
  - The DRS Broker acts as „license server“ in order to keep track of configured volumes and volumes allocated by running jobs
  - The DRS Client queries the DRS Broker for that information.

Moab Implementation

- At job submission the user can not only specify the amount of dynamic remote scratch to allocate, but also **how** the tiered scratch space shall be assembled for the job.
- Has been achieved by introduction of **alias names** for requested DRS resources, e.g.:
  - -l gres=drs_concat:5 → use 5 TB DRS space, concatenate hybrid FS with local SSDs
  - -l gres=drs_cache:5 → use 5 TB DRS space, cached by local SSDs
  - Alias names can be easily configured in DRS Broker
- Requested DRS resources are passed to job environment (as 5th argument to prologue script) by their alias name, such that **prologue script also "knows" what to do with the remote attached storage.**

Moab Implementation

- **Autoadjust Node Access Policy** of job whenever DRS resource is requested
  - Default Node Access Policy of JUSTUS cluster is **SINGLEUSER** (i.e. multiple jobs of the same user may run side by side on one node).
  - For jobs with dedicated DRS resource, we need to build, assemble and format the local scratch filesystem in the job prologue (and also revert these changes in the job epilogue).
  - In order to prevent interferences with other jobs, any job that requests DRS resources must run on a dedicated node with no other job running on that very same node, i.e. node access policy attribute of job must be automatically adjusted to **SINGLEJOB**.
  - This has also been achieved by implementing appropriate rules in Moab's submitfilter facility.

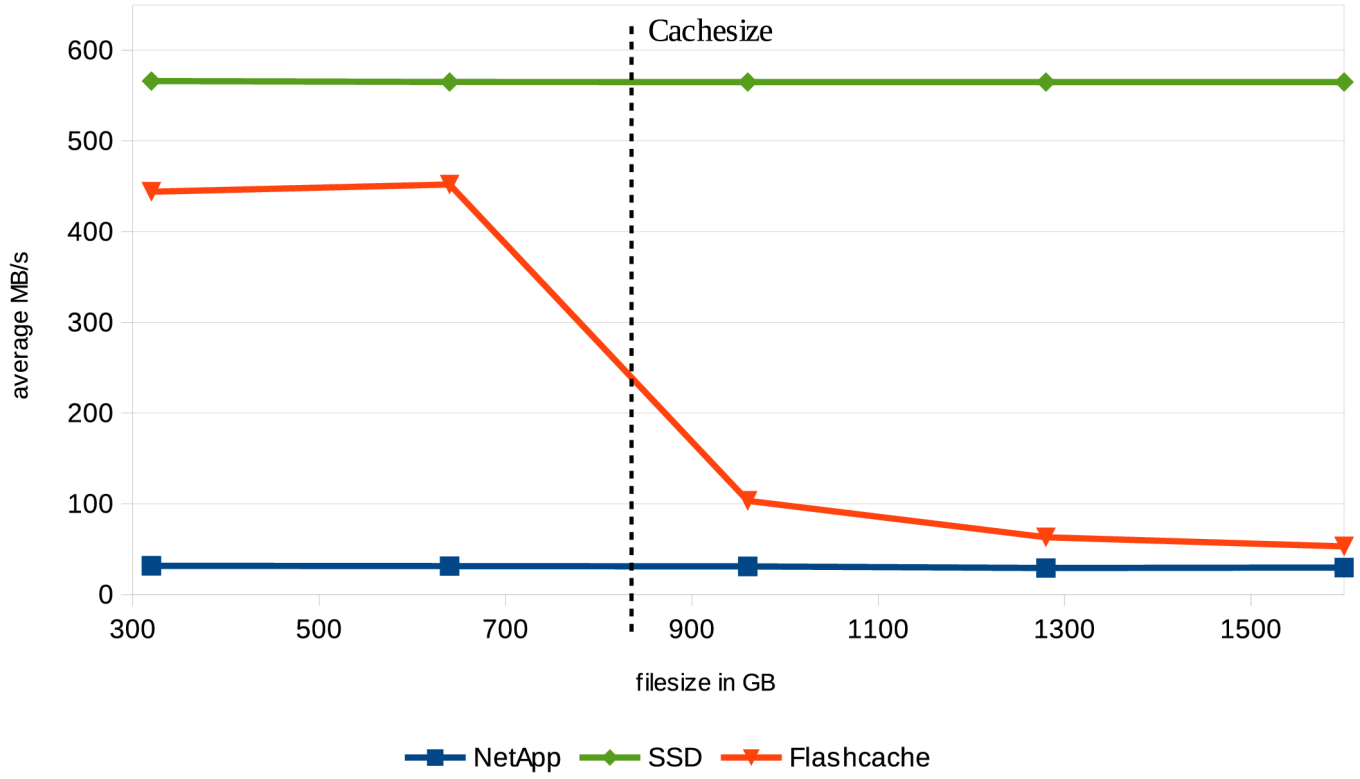
Moab Implementation

- **Final provisioning of DRS resources** at compute node:
  - A customized **torque prologue**, which runs with root privileges on the job execution nodes, decides which remote storage resources are requested for the job (amount of DRS storage and what to do with it, according to its alias name).
  - The prologue script runs the *DRS Client*, which – in turn – queries the *DRS Broker*, which makes those resources available on the remote storage device and keeps track of the allocation. Finally, the SRP initiator is configured and the scratch filesystem is created and mounted.
  - When the job ends, a customized **torque epilogue** is used to unmount the scratch filesystem, clean up and to signal the broker that resources are no longer in use.
  - Broker also takes care of unprovisioning DRS volumens on storage device side

Benchmarks

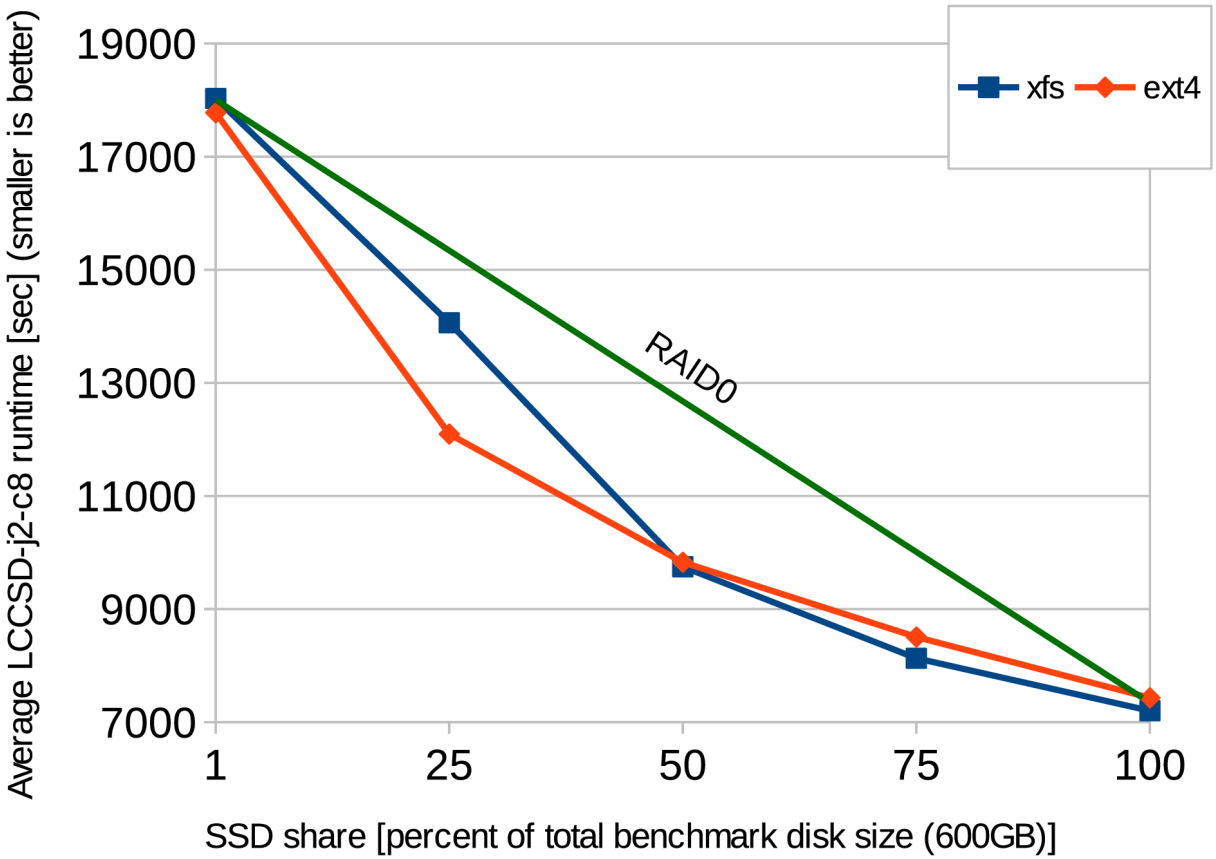
Cache Based Approach

- Synthetic fio-benchmark modeling real quantum chemistry job
- Single process run
- System: 128 GB RAM, a RAID0 array of 4 SSDs with a total of 850 GB



Concatenated Storage System

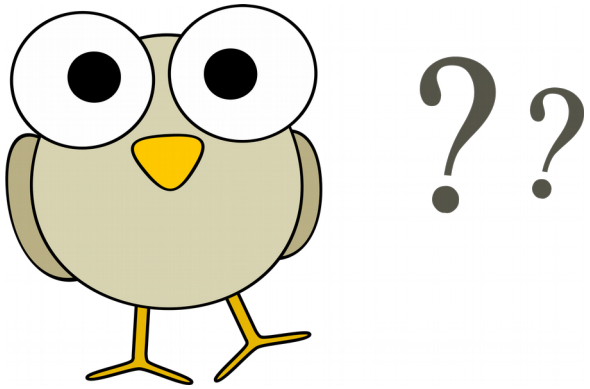
- Molpro LCCSD benchmark



Conclusions and outlook

- Comprehensive I/O analysis possible with standard linux tools
- Combination of local and remote storage can be a solution
- Caching solutions didn't work well in our case
- LVM reveals good performance
- Flexible implementation possible with Moab

Thank you for your attention.
Questions?



Matthias Neuer.
Communication and Information Center (kiz)
Infrastructure Dept. | Scientific Software & Compute Services

Albert-Einstein-Allee 11 | 89081 Ulm | Germany
matthias.neuer@uni-ulm.de | http://www.uni-ulm.de/kiz

Acknowledgements:
* **Thanks to the Deutsche Forschungsgemeinschaft (DFG) and the Ministry for Science, Research and Arts Baden-Württemberg for funding the bwForCluster JUSTUS.**
* **Thanks to the bwHPC-C5 team Baden-Württemberg.**
* **Thanks to all of my colleagues and co-authors who contributed to this talk.**