JG|U

JOHANNES GUTENBERG UNIVERSITÄT MAINZ

Providing QoS-mechanisms for Lustre through centralized control applying the TBF-NRS

Lustre User Group 2017

L. Zeng, J. Kaiser, A. Brinkmann, T. Süß – JGU

L. Xi, Q. Yingjin, S. Ihara – DDN

Mainz, Germany

- Capital of the state of Rhineland-Palatinate
- Directly located at the Rhine
- Founded in the late first century BC
- Member of the Great Wine Capitals Global Network (GWC)

Mainz, Germany

- Capital of the state of Rhineland-Palatinate
- Directly located at the Rhine
- Founded in the late first century BC
- Member of the Great Wine Capitals Global Network (GWC)

- Gutenberg Bible has been printed in Mainz

Johannes Gutenberg University Mainz

- Founded in 1477 and reopened after a 150-year break in 1946 by the French forces
- 35,000 students from about 130 nations
- 4,150 academics, including 540 professors, teach and conduct research in JGU's more than 150 departments, institutes, and clinics
- Extraordinary research achievements in the fields of particle and hadron physics, materials sciences, and translational medicine

Zentrum für Datenverarbeitung

MOGON SPECS

64 — 2100MHz Opteron 6272

Nodes — 555

35520 — Cores

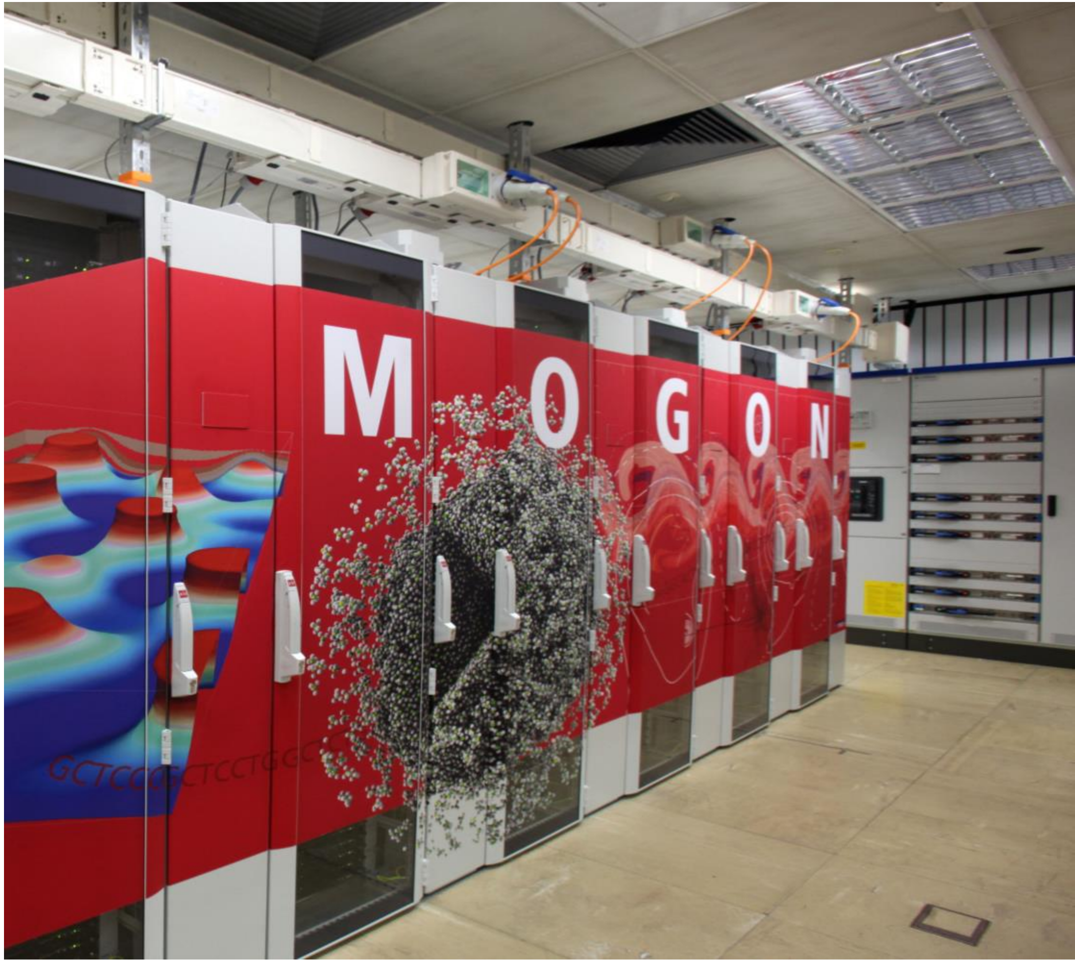
Total RAM — 89TB

1110TB — Local Storage

QDR Infiniband — Fat tree

Peak Performance

298 Teraflops




Ultra-DENCO

Mogon II

Mogon II - MEGWARE MiriQuid, Xeon E5-2630v4 10C 2.2GHz, Intel Omni-Path

Site:	Universitaet Mainz
System URL:	https://hpc.uni-mainz.de/high-performance-computing/mogonbild
Manufacturer:	MEGWARE
Cores:	16,500
Linpack Performance (Rmax)	557.572 TFlop/s
Theoretical Peak (Rpeak)	580.8 TFlop/s
Nmax	2,534,400
Nhalf	220,000
Power:	242.43 kW (Submitted)
Memory:	81,920 GB
Processor:	Xeon E5-2630v4 10C 2.2GHz
Interconnect:	Intel Omni-Path
Operating System:	CentOS

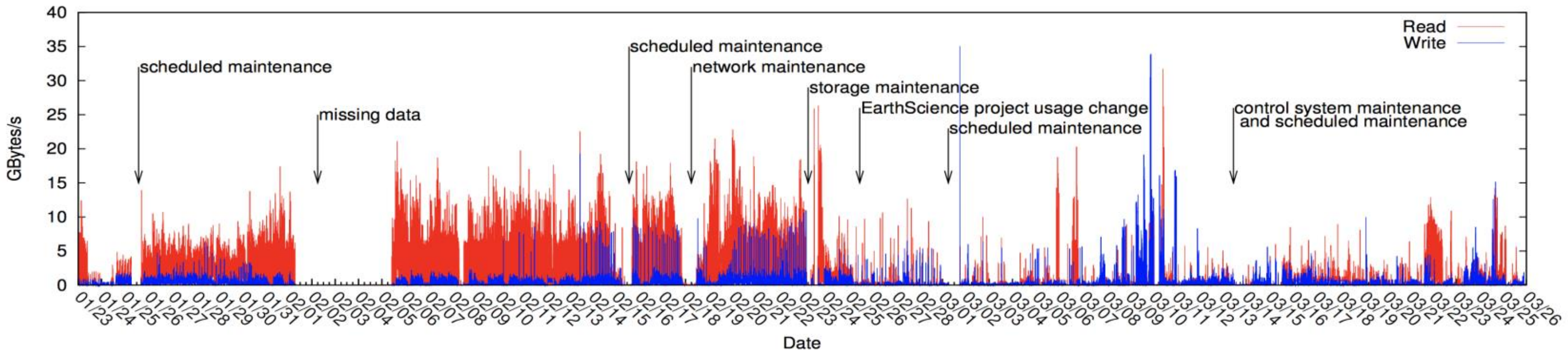
RANKING

List	Rank	System	Vendor	Total Cores	Rmax (TFlops)	Rpeak (TFlops)	Power (kW)
11/2016	265	MEGWARE MiriQuid, Xeon E5-2630v4 10C 2.2GHz, Intel Omni-Path	MEGWARE	16,500	557.6	580.8	242.43

Agenda

- Why do we need Quality of Service for HPC?
- Architectural Approaches
- Keep it simple: Integration of QoS-Manager and extensions for Slurm and Lustre
- Scenarios and Evaluation

Motivation: I/O Burstiness



Throughput at the block device level of Intrepid's main storage devices from January 23rd to March 26th including GPFS and PVFS activity

Philip H. Carns, Kevin Harms, William E. Allcock, Charles Bacon, Samuel Lang, Robert Latham, Robert B. Ross: Understanding and improving computational science storage access through continuous characterization. MSST 2011: 1-14

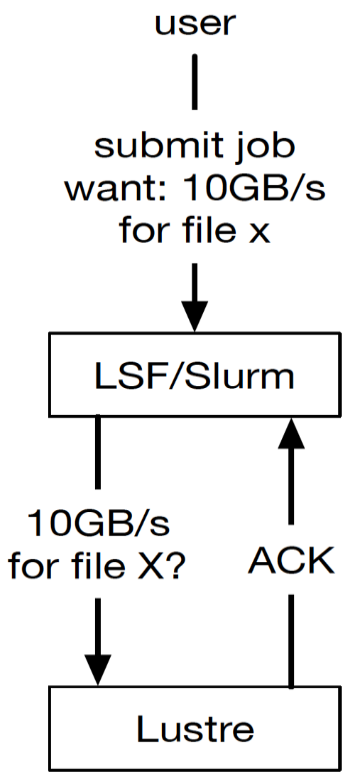
Why Quality of Service?

- I/O resources are typically not part of the scheduling process
  - Users might acquire bigger capacity share of the storage system, but do not receive more bandwidth
  - Individual compute jobs are able to (accidentally) perform denial of service attacks by flooding the parallel file system with many small requests or metadata operations
  - Concurrently running checkpoint operations overload parallel file system bandwidth and therefore prolong application runtimes

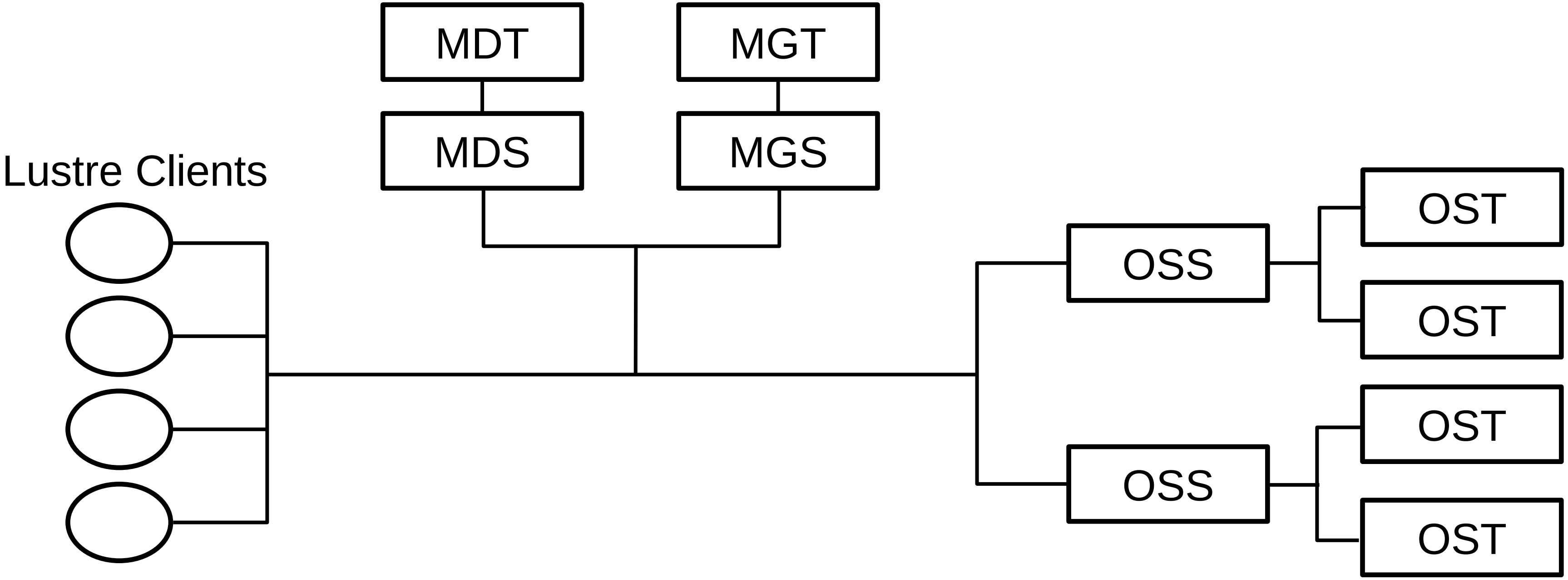
QoS Planning in Lustre

- QoS Planning for storage resources
  - Guarantee x GB/s read throughput
  - Guarantee y GB/s write throughput
  - For specific files?

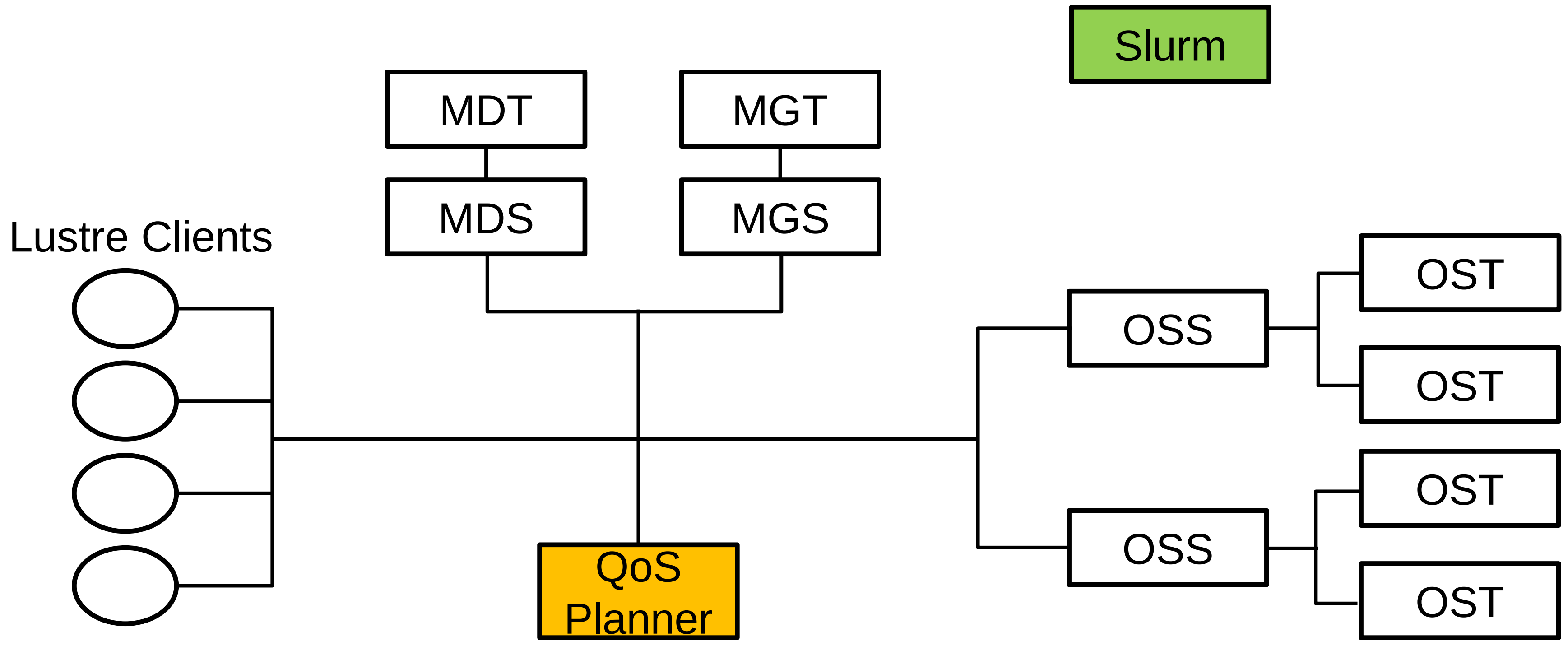
- Architecture includes
  - Batch System
  - Client and/or server component in Lustre enforcing QoS



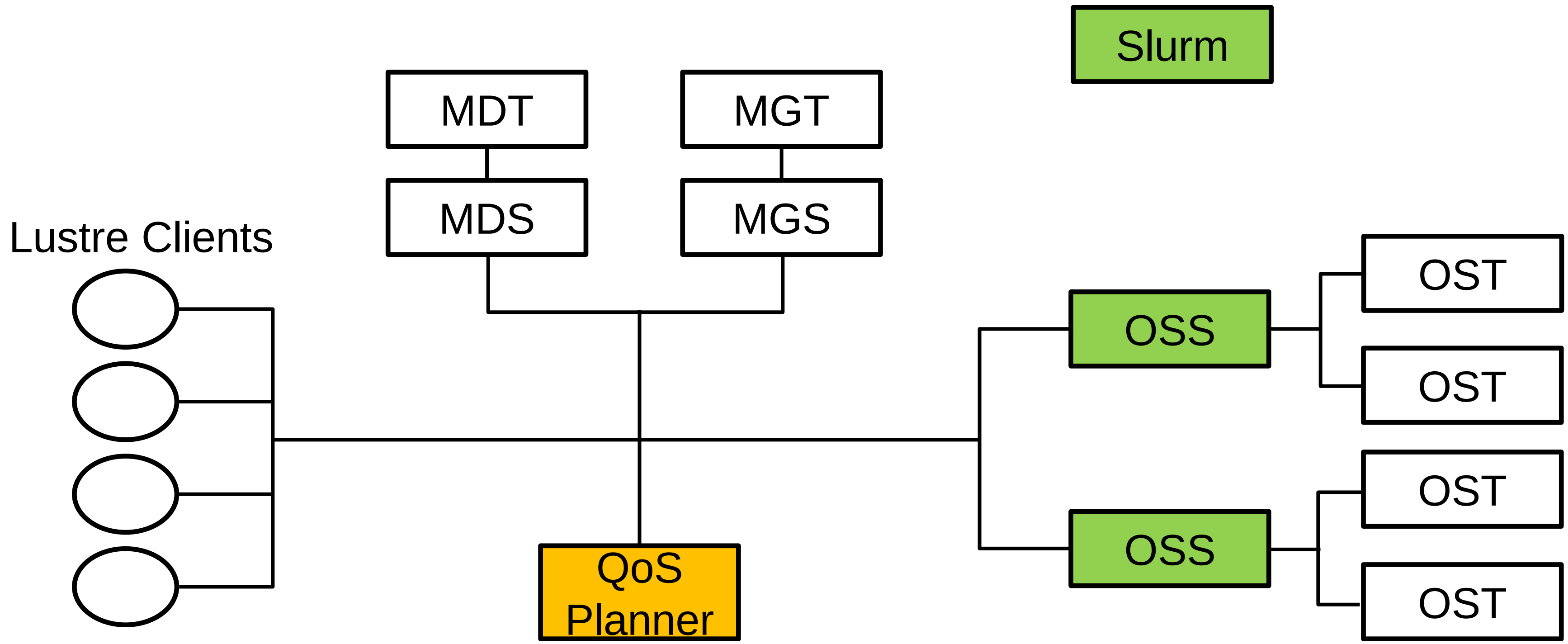
Architecture including QoS-Planner



Architecture including QoS-Planner



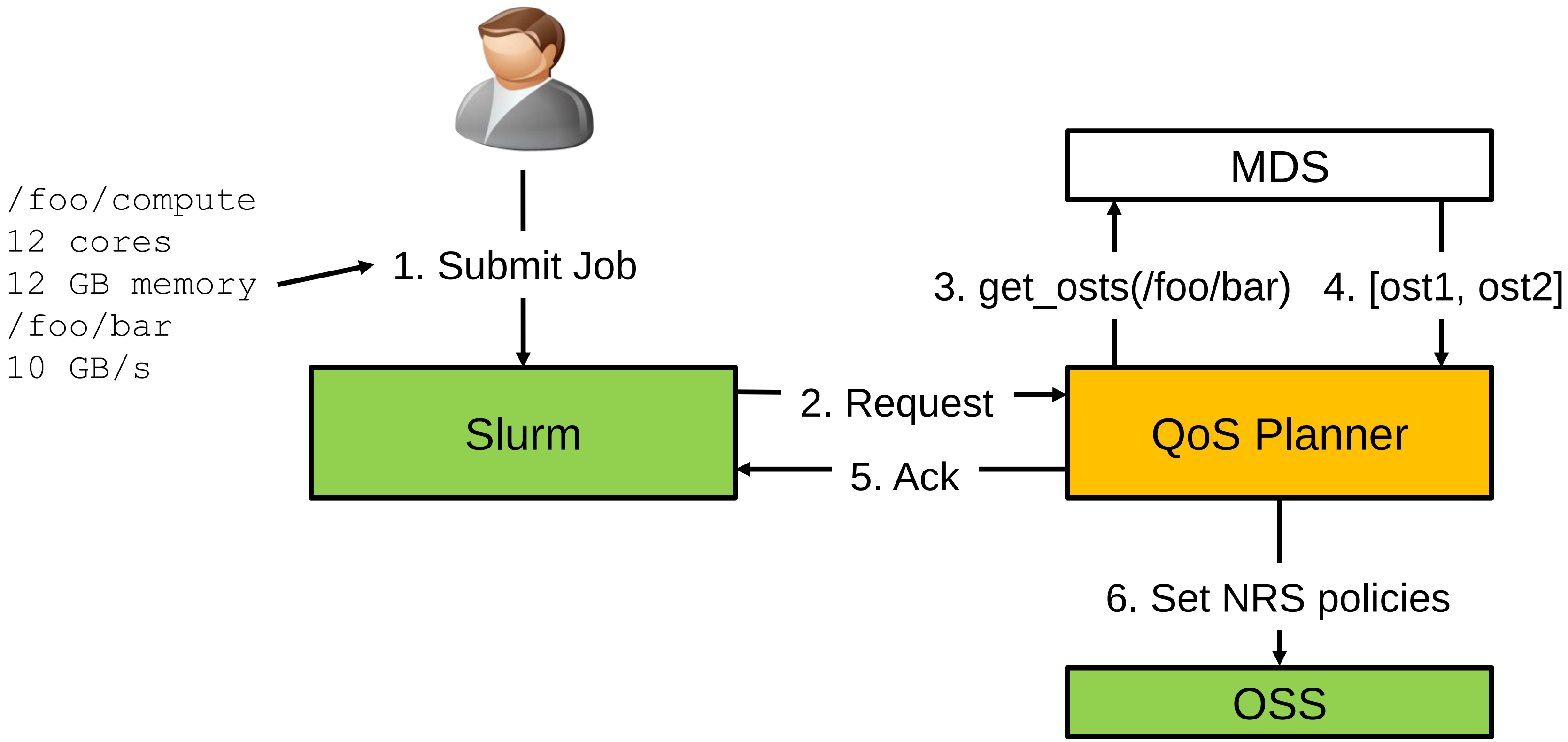
Architecture including QoS-Planner



Initial Approach: Reserve Bandwidth per File

- Lustre's lfs command allows to determine the OST's storing a file
  - Each OST provides a certain bandwidth
  - OST of an OSS can be seen as individual "resources" just like nodes in a cluster

Initial Control Flow

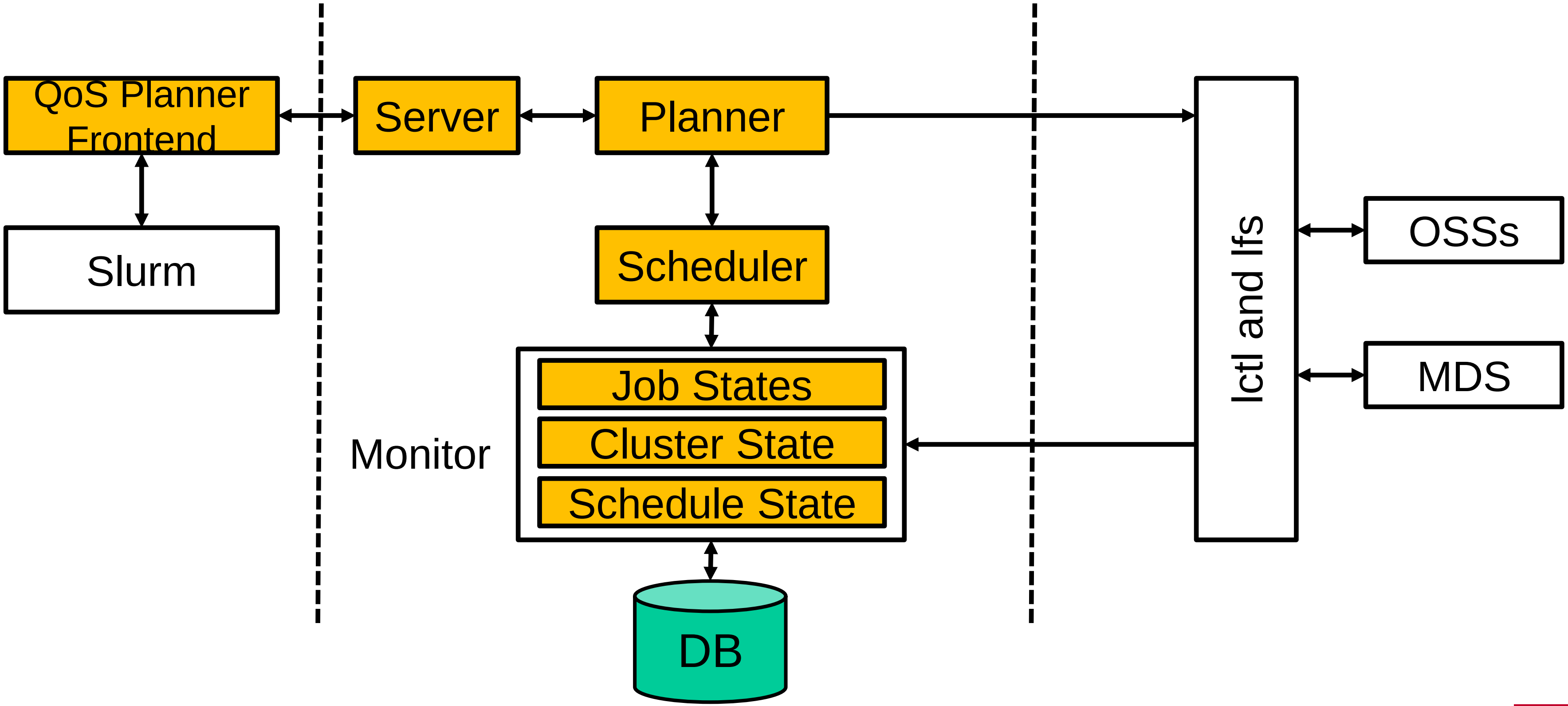


Initial Approach: Reserve Bandwidth per File

- Lustre's lfs command allows to determine the OST's storing a file
  - Each OST provides a certain bandwidth
  - OST of an OSS can be seen as individual "resources" just like nodes in a cluster

- Approach leads to two (major) problems
  - Jobs including many small files prohibit scalability of this approach
  - Lustre (now) allows growing file stripes

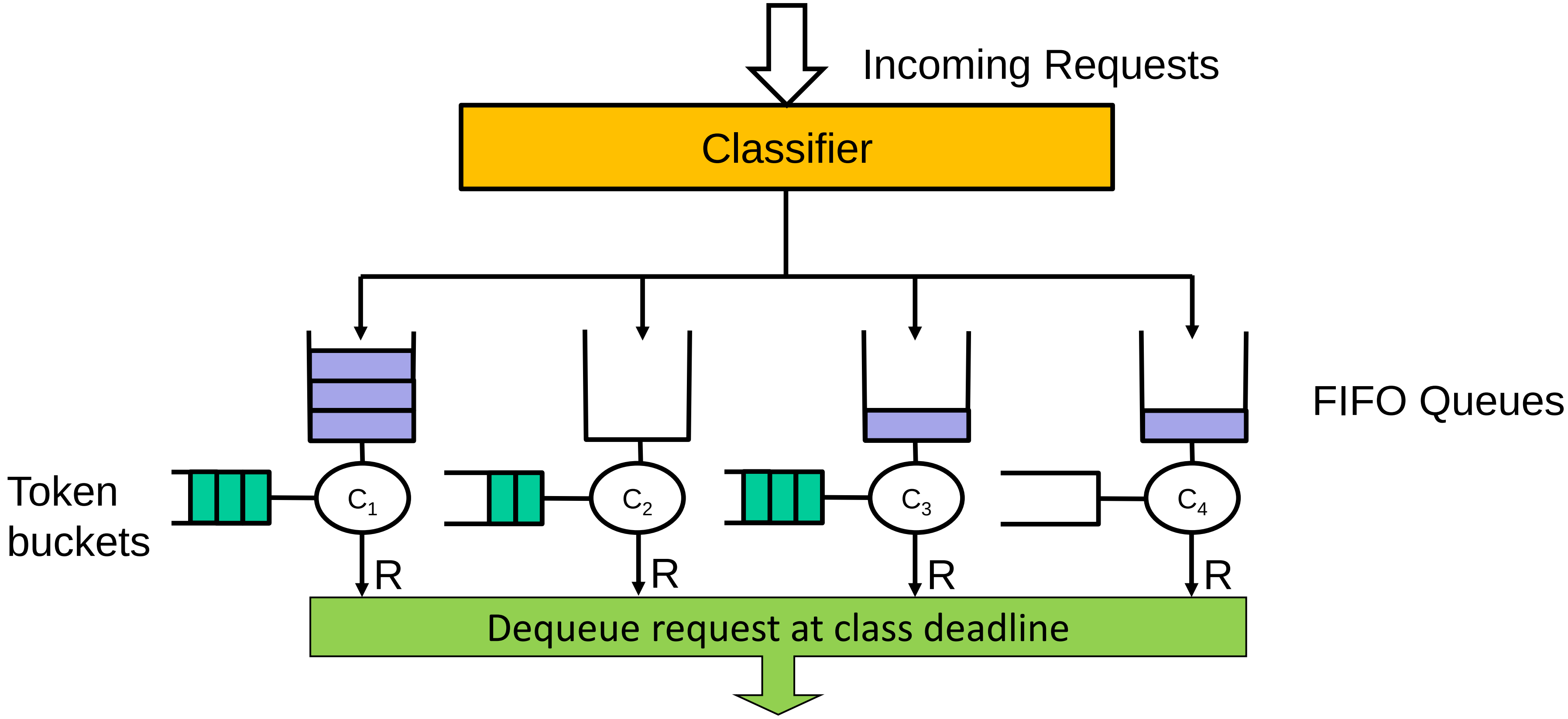
Current Approach: Do not care about files ...



Slurm Integration

- Bandwidth is defined as a global and as a local resource
- Slurm plug-in controls:
  - Globally available bandwidth - treated as license (one license/MB)
  - Local bandwidth - treated as generic resource
- Job gets rejected if one resource is not available
- Example:
  - `srun -N1 -gres=qoslustre:100M -L lustreqos:100 sleep 5`

Token Bucket Filter (TBF)

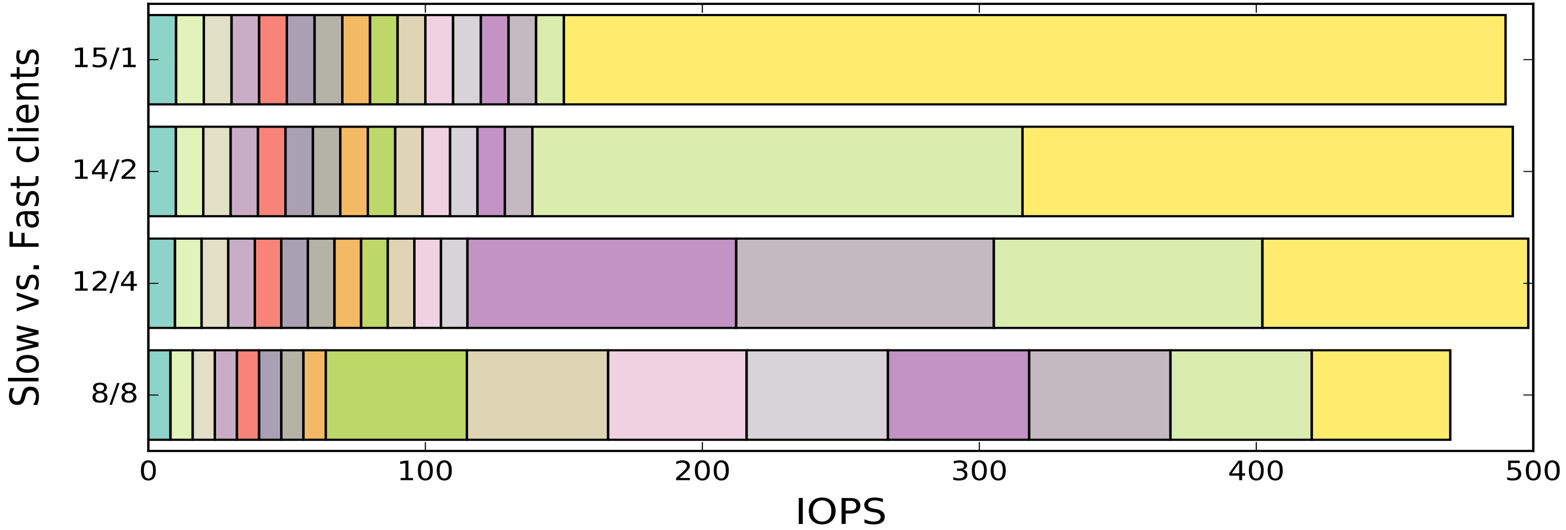


Token Bucket Filter (TBF)

- TBF is implemented inside Lustre‘s Network Request Scheduler (NRS)
- 1 Token = 1 RPC ≈ 1 Mbyte (for 1 Mbyte chunks)
- Class-based TBF can classify by User ID, Job ID, …
- Batch System / Administrator assigns token rates (per OSS)

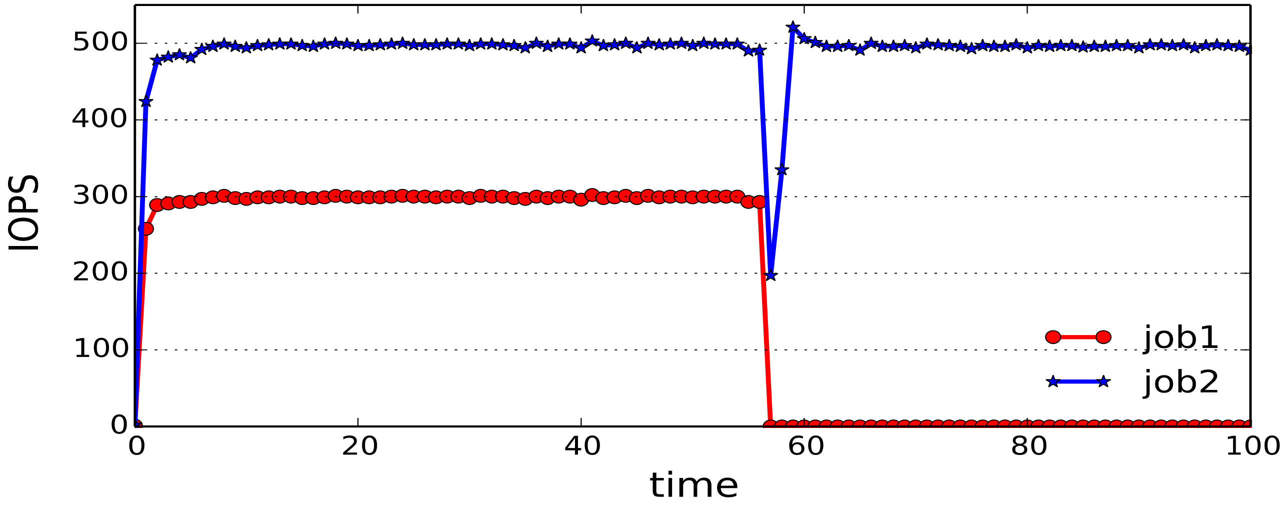
- *Throughput for multiple flows enables* fair bandwidth distribution
- *Proportional Sharing Spare Bandwidth* (*PSSB*) enable utilization of full bandwidth of OSSs

Throughput for multiple flows



- 16 clients are divided into two sets
  - Slow clients are assigned a rate of 10 IOPS
  - 10,000 IOPS are assigned to fast clients
- Clients with same rate receive same bandwidth

PSSB Evaluation



- 16 clients working in parallel on job1, where each client wrote 1 GB data at an initial rate setting of 150
- 16 clients were running job2, each writing 2 GB data at an initial rate setting of 100

JG|U

Putting it all together ...

We have integrated our QoS-Planner on our productive system Mogon II

- QoS server runs on two OSSs responsible for scratch file system
  - **nrs_policies="tbf jobid"**
  - **jobid_var=procname_uid**
- OSS use Lustre's TBF version 2.8
- QoS client is installed on compute nodes
  - **jobid_var=procname_uid**

JG|U

A priori Reservation

A client application for reserving bandwidth has been developed for Slurm

```
# qosp reserve -throughput 100 -duration 100 \
  -filenames /path/to/folder -id=slurm_job_id
```

- Command reserves a **throughput** of 100 RPCs for 100 seconds
- OSTs are identified via **filenames** respectively paths
- Available shares can be identified via **id**

A priori Reservation

Slurm-plugin uses `qosp` command for reserving bandwidth
Throughput is taken from global and local resource

Further integrations are possible:

- Coupling users or groups with QoS manager
  - Groups that gave additional money for storage get more shares
  - Malicious users/groups can be throttled down
- Credit bandwidth of reservations that terminate earlier

Spontaneous Reservation

Many programs require high I/O bandwidth only for a short time period

- Loading input data during initialization
- Checkpointing
- Storing final results

We provide a C++ API for spontaneous I/O accesses

- Reserve bandwidth for a certain time span
- Test if reservation is available
- Remove reservation after I/O is done

Spontaneous Reservation

Most important API functions:

```
// none-blocking reservation
string addReservationAsync(int tp, int sec, string fs);
// blocking reservation
string addReservationSync(int tp, int sec, string fs);
// delete a specific reseravtion
bool removeReservation(string id);
// test the status of a reservation
// (UNDEFINED, SCHEDULED, ACTIVE)
// required for asynchronous reservation
int testReservation(string id);
```

Spontaneous Reservation

QoS scheduler currently uses backfilling, thus a reservation start time may change during waiting period

Asynchronous functions supports this behavior

```
// none-blocking reservation
string addReservationAsync(int tp, int sec, string fs);
// test the status of a reservation
int testReservation(string id);
```

Programs like Espresso++ or tools like SCR can use these features to request bandwidth for asynchronous checkpoints

Espresso++

After every simulation step the checkpointing function `DumpXYZQoS::dump()` is called

```
void DumpXYZQoS::dump() {
  if(!qos_waiting){
    qosId = qosp.addReservationAsync(1000, 10, filename());
    qos_waiting = true;
    conf.gather()
  }
  if(qosp.testReservation(qosId) != ACTIVE) return;
  qos_waiting = false;
  ... // write checkpoint
}
```

Thank you for your attention.

JOHANNES GUTENBERG UNIVERSITÄT MAINZ