Exploitation of network-segregated CPU resources in CMS

C. Acosta[1,2], A. Delgado Peris[3], J. Flix[2,3], J. Frey[4],
J. Hernández[3], A. Pérez-Calero Yzquierdo[2,3], T. Tannenbaum[4]

1 - IFAE 2 - PIC 3 - CIEMAT 4 - UWM

Outline

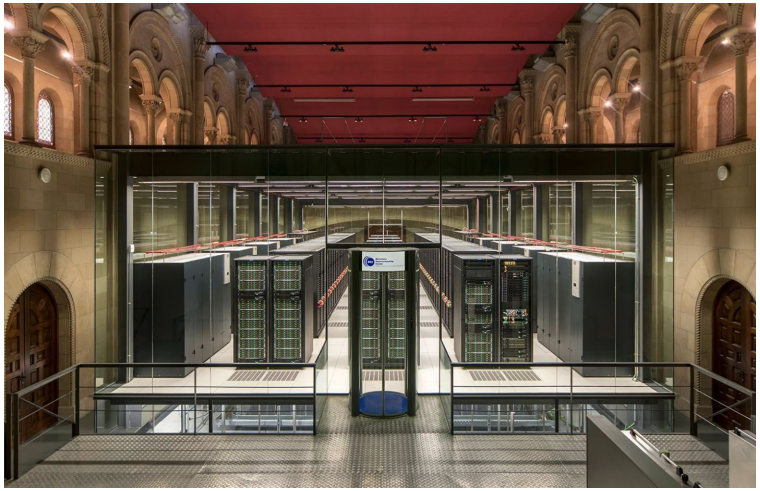
- Motivation and context
- Challenges
- Technical solutions
- Achievements & Status
- Conclusions and outlook

Motivation and context

- HPC facilities are attractive for WLCG computing
  - To help cover increasing CPU needs despite flat funding
  - Lot of public funding went to HPCs
  - Several ongoing HPC integration efforts in CMS
- BSC - Barcelona Supercomputing Center
  - Largest HPC center in Spain
  - Current MareNostrum 4 (***MN4***) general-purpose cluster:
    - 11.5 Petaflops (165,888 CPUs), 390 TB RAM, 24 PB disk
    - SLURM as batch system, SUSE Linux Enterprise as OS
    - GPFS as storage back-end (mounted on login/compute nodes)
  - Next MareNostrum5 (~17xMN4, ~200 petaflops), available in 2022
    - One of Europe's first pre-exascale supercomputers

Motivation and context (II)

- In 2020 BSC has designated LHC computing as a **strategic project**
  - Agreement promoted by WLCG-ES community and funding agency
- Allocations of up to a 7% share of MN4 for LHC
  - ~95M coreHours/year)
  - ~20M hours allocation for CMS in 2021
- Potentially, very significant contribution for LHC computing in Spain
  - Comparable e.g to all CMS simulation needs in the country



We sincerely thank the BSC and the Spanish Supercomputing Network (RES) for the contributed resources.

Outline

- Motivation and context
- **Challenges**
- Technical solutions
- Achievements & Status
- Conclusions and outlook

Challenges

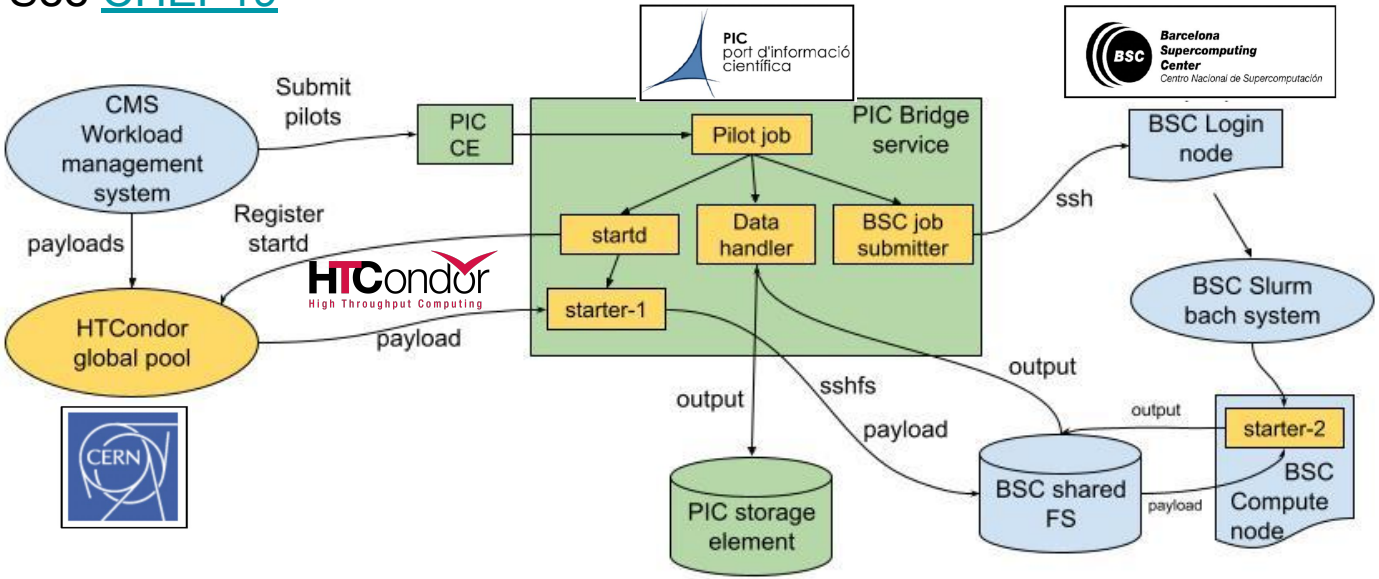
- BSC imposes very restrictive network connectivity conditions
  - No incoming *or outgoing* connectivity from compute nodes
  - No services can be deployed on edge/privileged nodes
  - Only incoming SSH/SSHFS communication through login nodes
- This is a major obstacle for CMS workloads
  - Pilot with late binding model execution of payloads
    - Workload management system (glideinWMS - HTCondor services)
  - Access to external services
    - Application software (CVMFS)
    - Conditions data (FrontierDB)
  - Consuming and producing experiment data
    - Input/output data files (Storage Elements)

Outline

- Motivation and context
- Challenges
- **Technical solutions**
- Achievements & Status
- Conclusions and outlook

Solutions - Workload management system

- HTCondor Split-starter
  - HTCondor development to use shared file system as communication layer
  - Bridge service node at PIC connects CMS WMS and BSC
  - See CHEP19



Solutions - External services

- Either... encapsulate and ship software and conditions data
  - Using *fat* singularity images, including required CMSSW and conditions data
  - Not trivial, and not produced routinely by CMS for every production campaign
- Or... pre-place them at BSC
  - Copying whole CVMFS tree to BSC storage (+ continuous updates)
  - Copying required condition data files (or also including them in CVMFS)
  - Provides greater flexibility!
- Developments in CMSSW were required
  - To enable transparent consumption of conditions data from SQLite files
  - Available at *CMSSW_11_2*

Solutions - Input/Output data

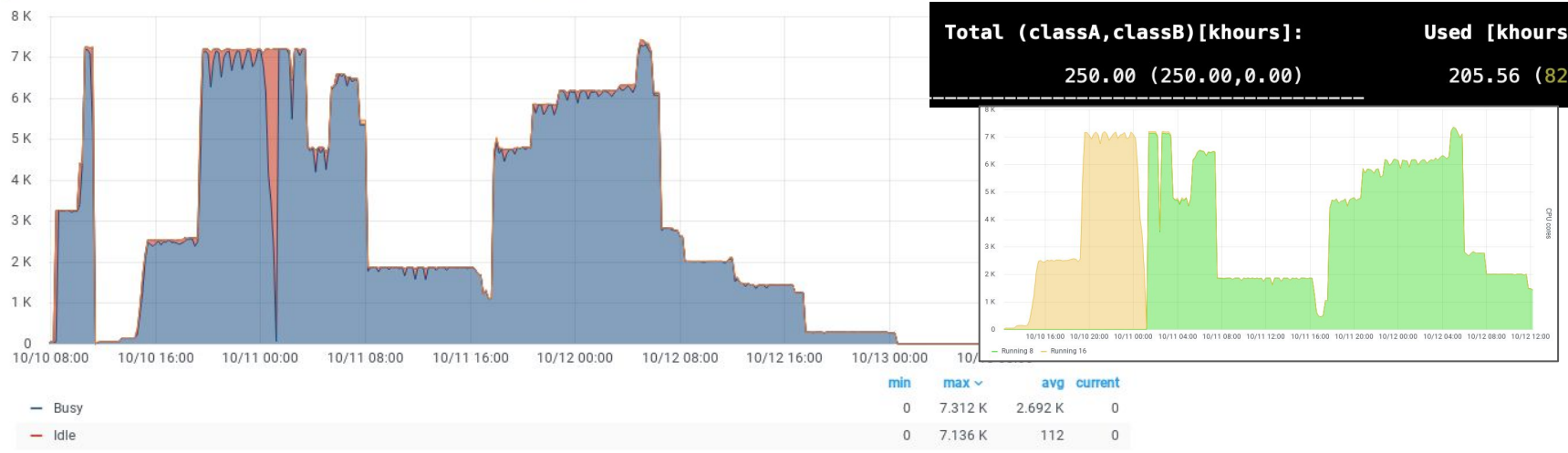
- Synchronous movement of data (coupled to jobs) preferred
  - Avoid race conditions on data consumption
    - Since data will be marked as available at PIC storage
- For produced data (*stage-out*)
  - Let jobs at BSC *cp* output files to per-job area on shared filesystem (full path)
    - Standard CMS method (with proper *site* configuration)
  - HTCondor starter at PIC then just *copies* whole paths to PIC's storage
    - Run by starter's hook *before* the job is considered finished
- Analogous mechanism could be used for input data (stage-in)

Outline

- Motivation and context
- Challenges
- Technical solutions
- **Achievements & Status**
- Conclusions and outlook

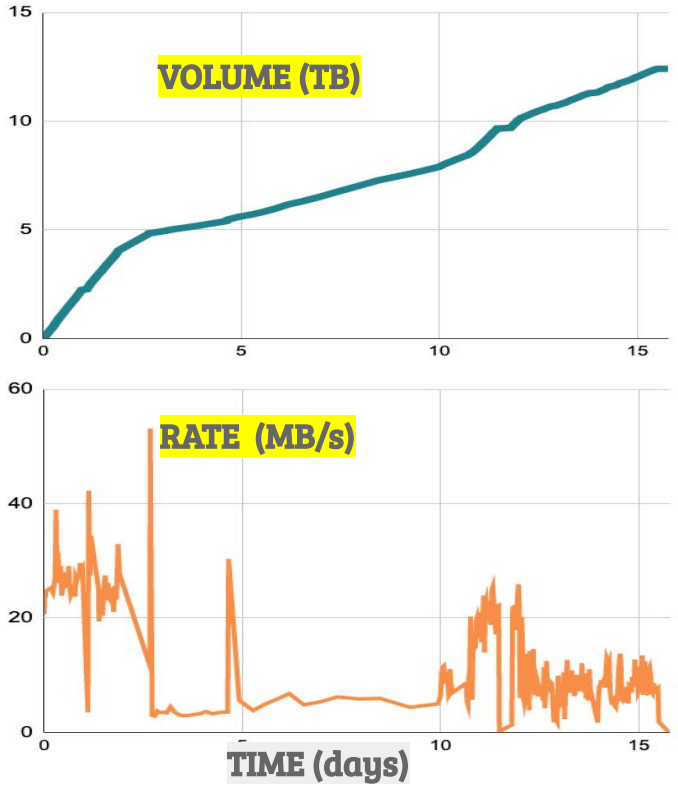
Proof of concept and scale tests

- Very successful *proof of concept* with custom singularity images
  - Excellent slot utilization, with payload SIM jobs configured with 8 or 16 cores
  - Single bridge machine (24 GB memory) can fill up to ~10K cores at BSC
  - More powerful HW, or several bridge hosts at PIC, required to scale further



Copy of whole CMS CVMFS tree

- Copied whole *cms.cern.ch* repository:
  - 12.6 TB, 183M files (37M files de-duplicated)
- Used *cvmfs_preload* tool
  - Avoids duplication, skips already present files
  - Run at PIC, directly into an SSHFS mount of BSC shared filesystem
- Took ~2 weeks to complete
  - After initial phase with frequent transfer errors
    - Intervention at stratum 0 was required
  - Directories with large files showed higher rates
  - Following periodic updates (*deltas*) should be much faster



Functional validation

- Run GEN-SIM job without the need of custom (*fat*) singularity image
  - Used standard OSG CentOS 7 singularity image (like Grid jobs)
  - Used cvmfsexec to access copied CVMFS repository for CMSSW
  - Read conditions data from pre-placed SQLite file
- Stage-out
  - Prototype implemented, and partially tested
- Integration with CMS WMS
  - Implementation and testing in progress
  - First, run payloads from CMS HTCondor pools on manually launched pilots
    - Match appropriate payloads only
  - Next, run pilots sent by CMS

Outline

- Motivation and context
- Challenges
- Technical solutions
- Achievements & Status
- **Conclusions and outlook**

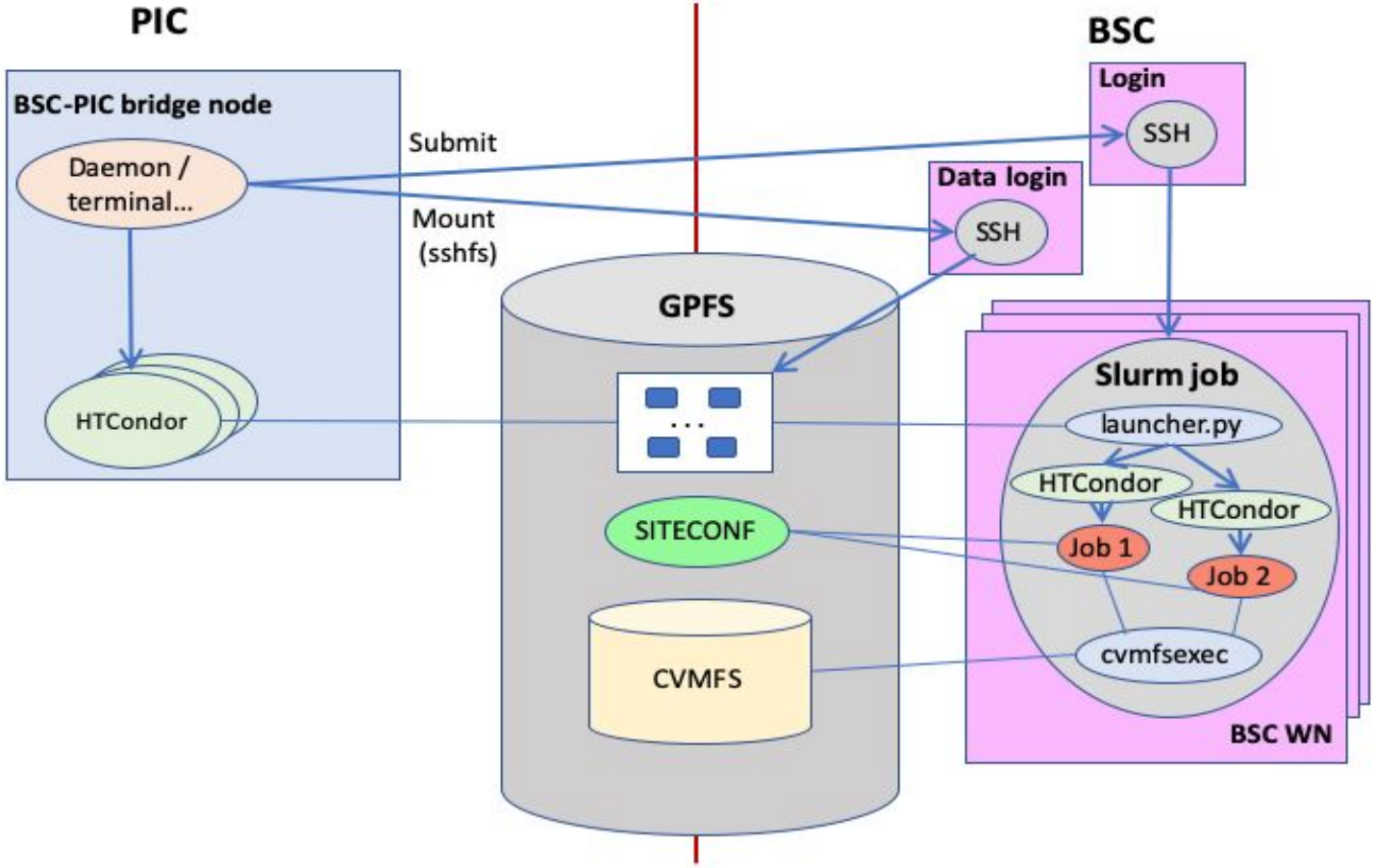
Conclusions and outlook

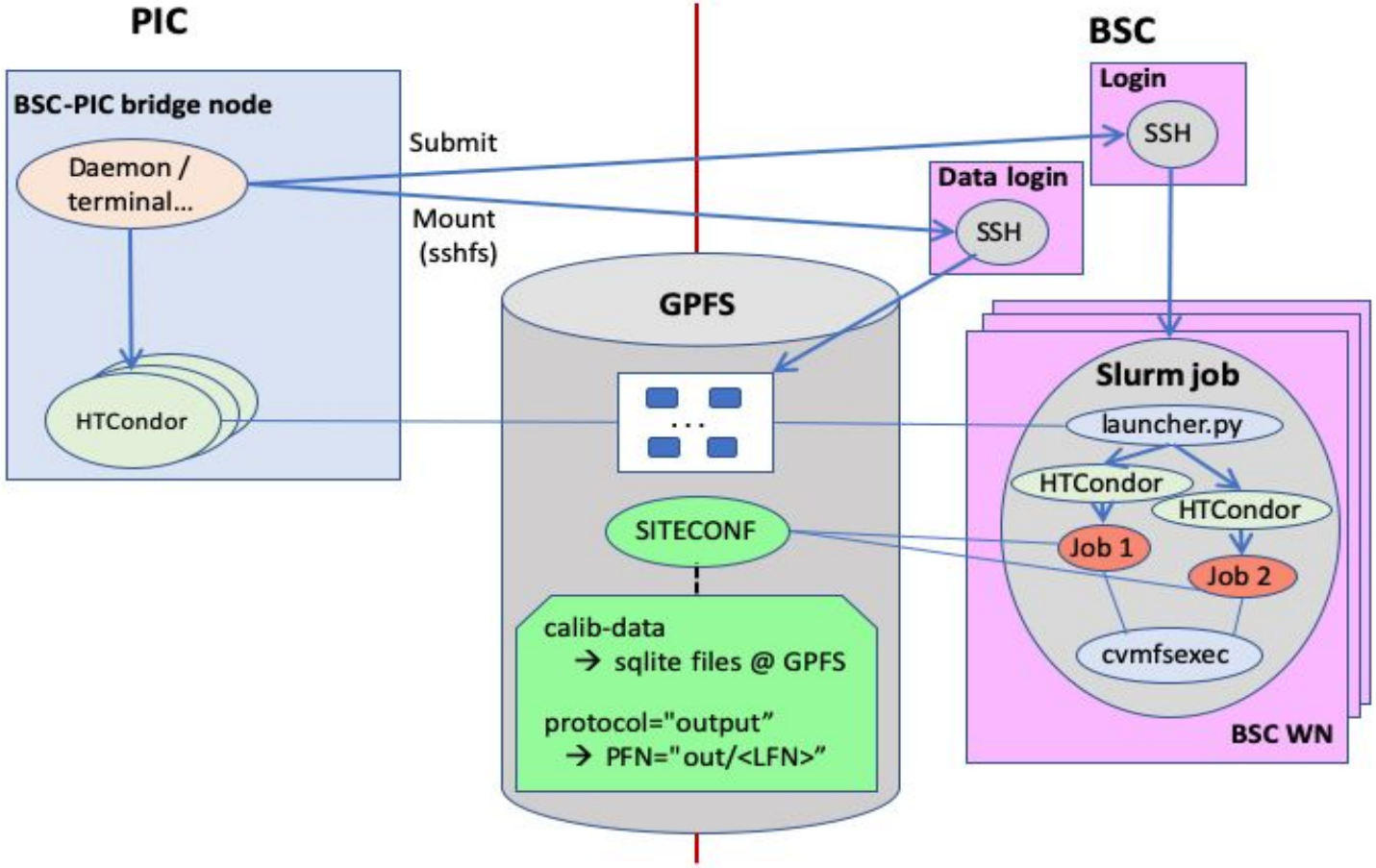
- We have demonstrated that BSC can be used to run CMS simulation workflows at scale
- The copy of CVMFS repository to BSC gives us much more flexibility
- Designed & implemented a procedure to deal with data staging
- Next tasks:
  - Testing new functionalities at scale
  - Integrate the system with CMS WMS
- New allocation (July - October 2021, 6 Mhours) just submitted
  - Expect allocations of >= 6 Mhours every 4 months

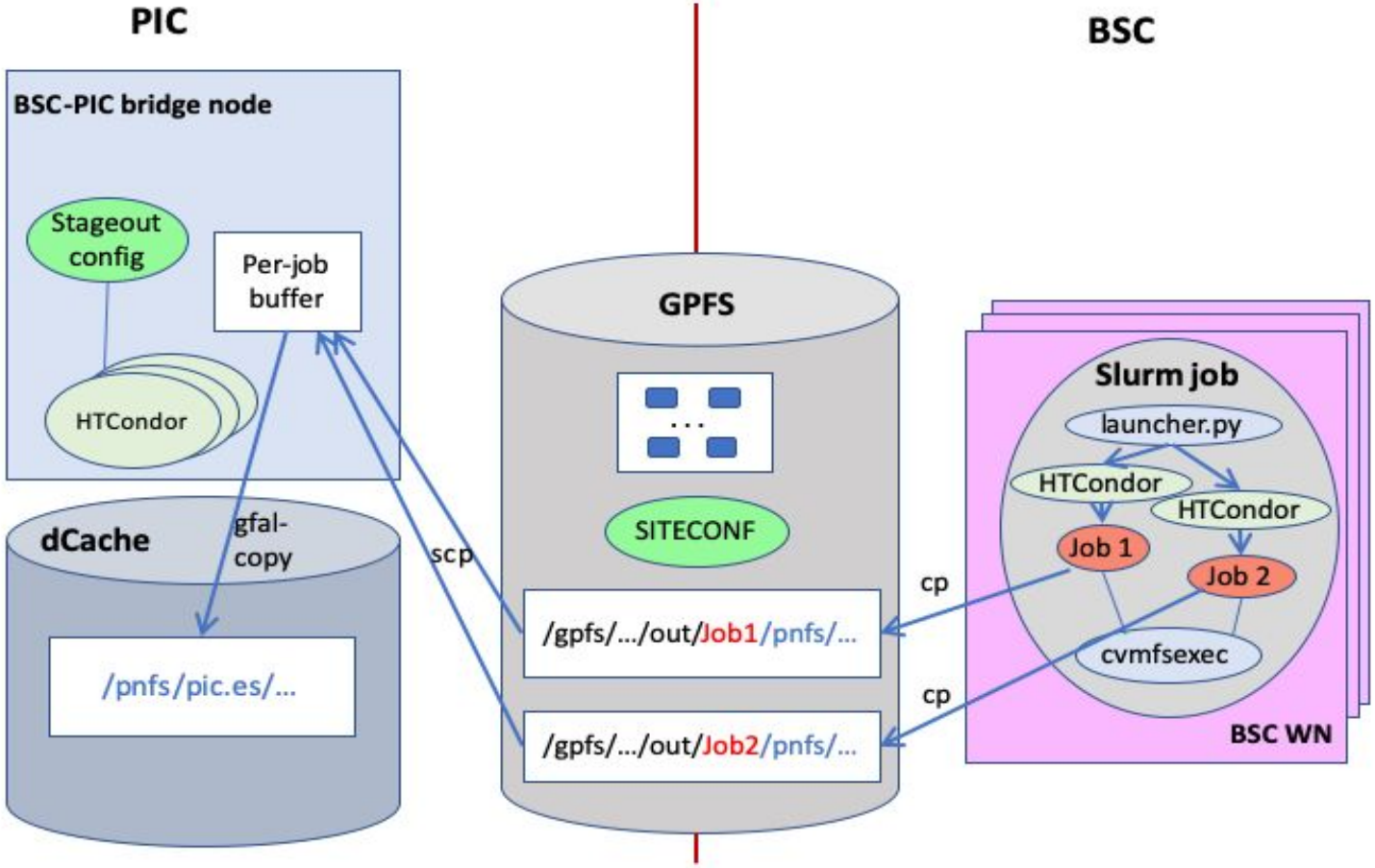
Thank you!

Questions?

Back-up Slides

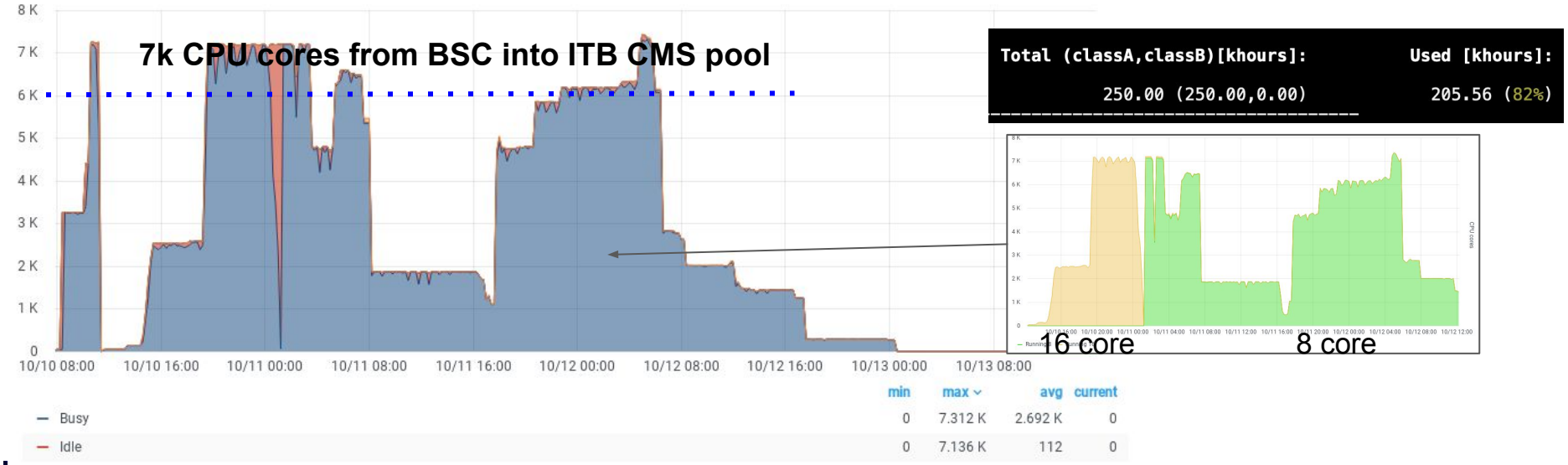

PIC
BSC
BSC-PIC bridge node
Daemon / terminal...
HTCondor
Submit
Mount (sshfs)
Login
SSH
Data login
SSH
GPFS
...
SITECONF
CVMFS
Slurm job
launcher.py
HTCondor
HTCondor
Job 1
Job 2
cvmfsexec
BSC WN


PIC
BSC
BSC-PIC bridge node
Daemon / terminal...
HTCondor
Submit
Mount (sshfs)
Login
SSH
Data login
SSH
GPFS
SITECONF
calib-data
→ sqlite files @ GPFS
protocol="output"
→ PFN="out/<LFN>"
Slurm job
launcher.py
HTCondor
HTCondor
Job 1
Job 2
cvmfsexec
BSC WN


PIC
BSC
BSC-PIC bridge node
Stageout config
Per-job buffer
HTCondor
dCache
gfal-copy
/pnfs/pic.es/...
scp
GPFS
SITECONF
/gpfs/.../out/Job1/pnfs/...
/gpfs/.../out/Job2/pnfs/...
cp
cp
Slurm job
launcher.py
HTCondor
HTCondor
Job 1
Job 2
cvmfsexec
BSC WN

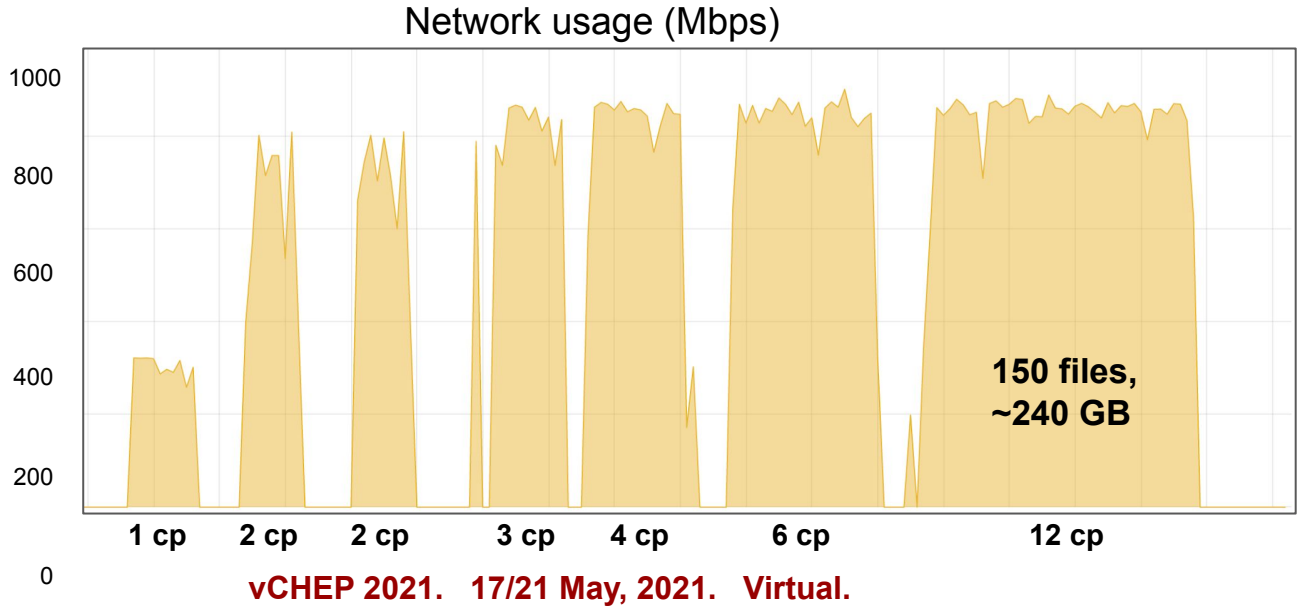
Test at scale in BSC

- Scalability tests executed with acceptable performance from all components
- Reached a reasonable 7k CPU cores at peak
- Excellent slot utilization, with payload SIM jobs configured to run as 16 and 8 core jobs
- Consumed our test allocation in a couple of days



Data transfer tests

- Mounted both BSC's gpfs (over sshfs) and PIC's pnfs (NFS) on the bridge node
- Then: *cp /gpfs/project/.../ /pnfs/pic.es/data/cms/store/.../*
- Test scalability and transfer rate of the naive approach, copying sets of multiple files (*cp -r*), with N simultaneous *cp* calls: **saturation reached at ~100 MB/s**

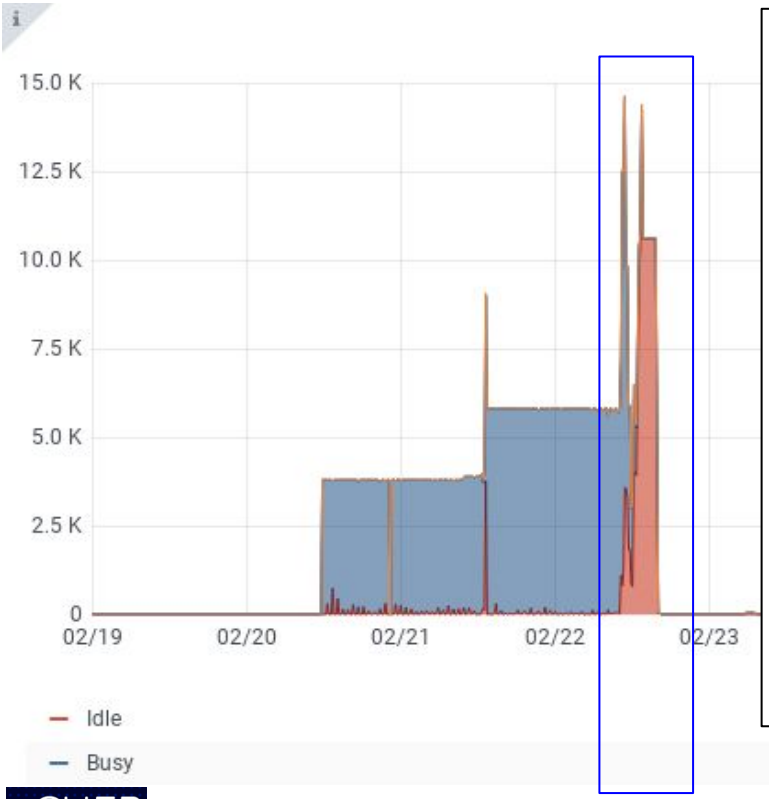


Feb BSC exploitation tests

Number of CPU cores allocated at BSC and connected to the CMS Global Pool

Issues limiting exploitation (I)

Number of CPU cores allocated at BSC and connected to the CMS Global Pool



During the initial phase, always kept up to 300 slurm jobs (*pilots*) in total at BSC queue (300x48 = 14.4k cores).

Idea: have same amount of pilots running and in queue (~150)

We first had ~80 pilots (for 24h), then ~120 pilots running (again 24h), for 3.8k and 5.7k cores respectively. This was handled ok by the bridge, and efficiently by the pool

The number of running pilots then increased to 300 at once, which saturated the bridge capacity. The main limiting factor was memory, proportional to # of startd and starter processes

Unresponsive bridge caused slots to fail matchmaking for new jobs: all slots went idle

Issues limiting exploitation (I)

Memory increase in the bridge node (td513) as a result of the increment in running processes.

At one point we had an allocation of 300 pilots = 14400 CPU cores (~10% of total Mare Nostrum)

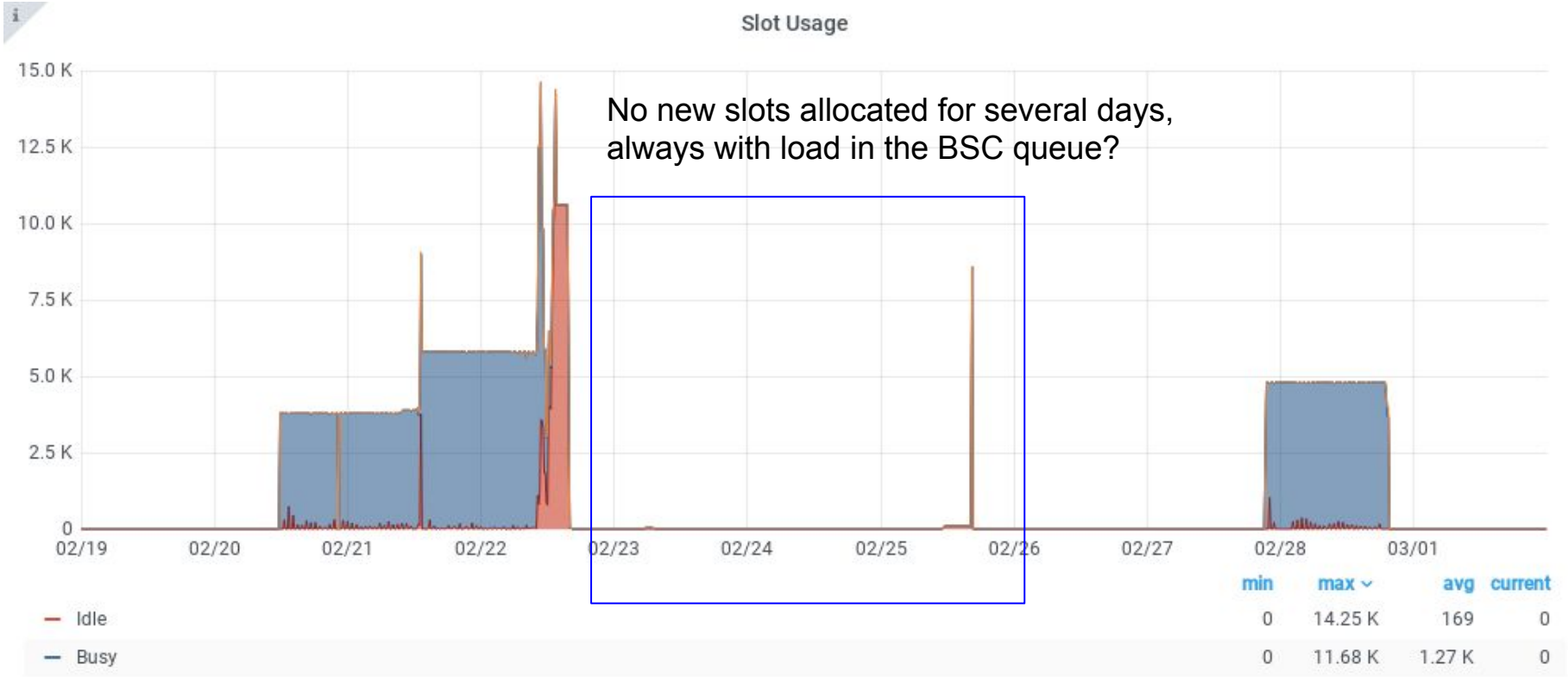
Excessive for the single bridge to handle.

Solution proposed:

- Limitations in the number of pilots running to protect the bridge
- Deploy several bridges in parallel, to be able to absorb peaks in allocated capacity

Issues limiting exploitation (II)

Number of CPU cores allocated at BSC and connected to the CMS Global Pool



Issues limiting exploitation (II)

The problem was in the way each BSC pilot was programmed: each process running in the bridge continuously checks if the BSC slurm job is running or still queued, before launching the HTCondor startd daemon

But this means 300 pilots in queue running a squeue query every 5 minutes = 3600 squeue's per hour, 1 Hz!!

Queries got killed and unnoticed until notified, for a couple of days no new jobs started!

Solution proposed:
Either centralize the squeue polling for job status (a single query for all jobs), or provide alternative for each job individually (check the sshfs rendez-vous directory)