CouchDB-based system for data management in a Grid environment

Implementation and Experience

Hassen Riahi
IT/SDC, CERN

Outline

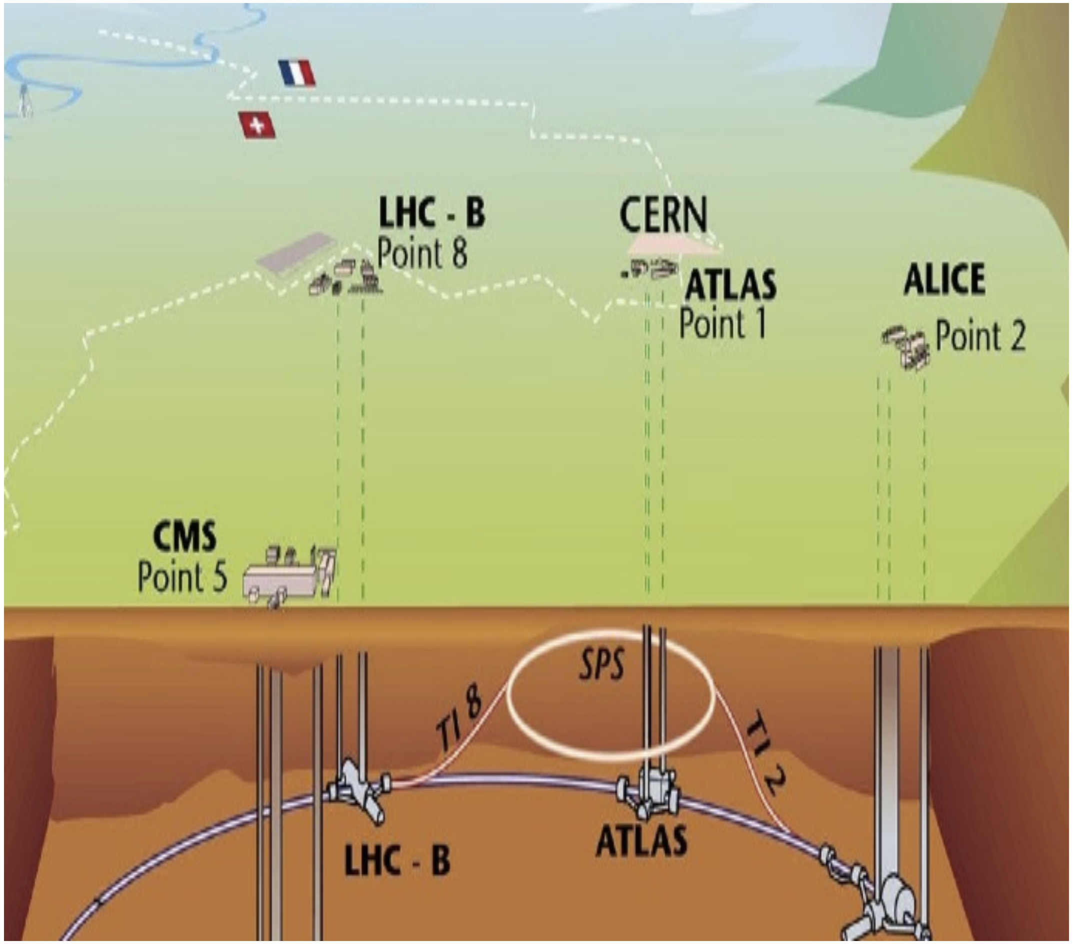
- Context
- Problematic and strategy
- System architecture
- Integration and deployment models
- Experience and lessons learnt

Who am I?

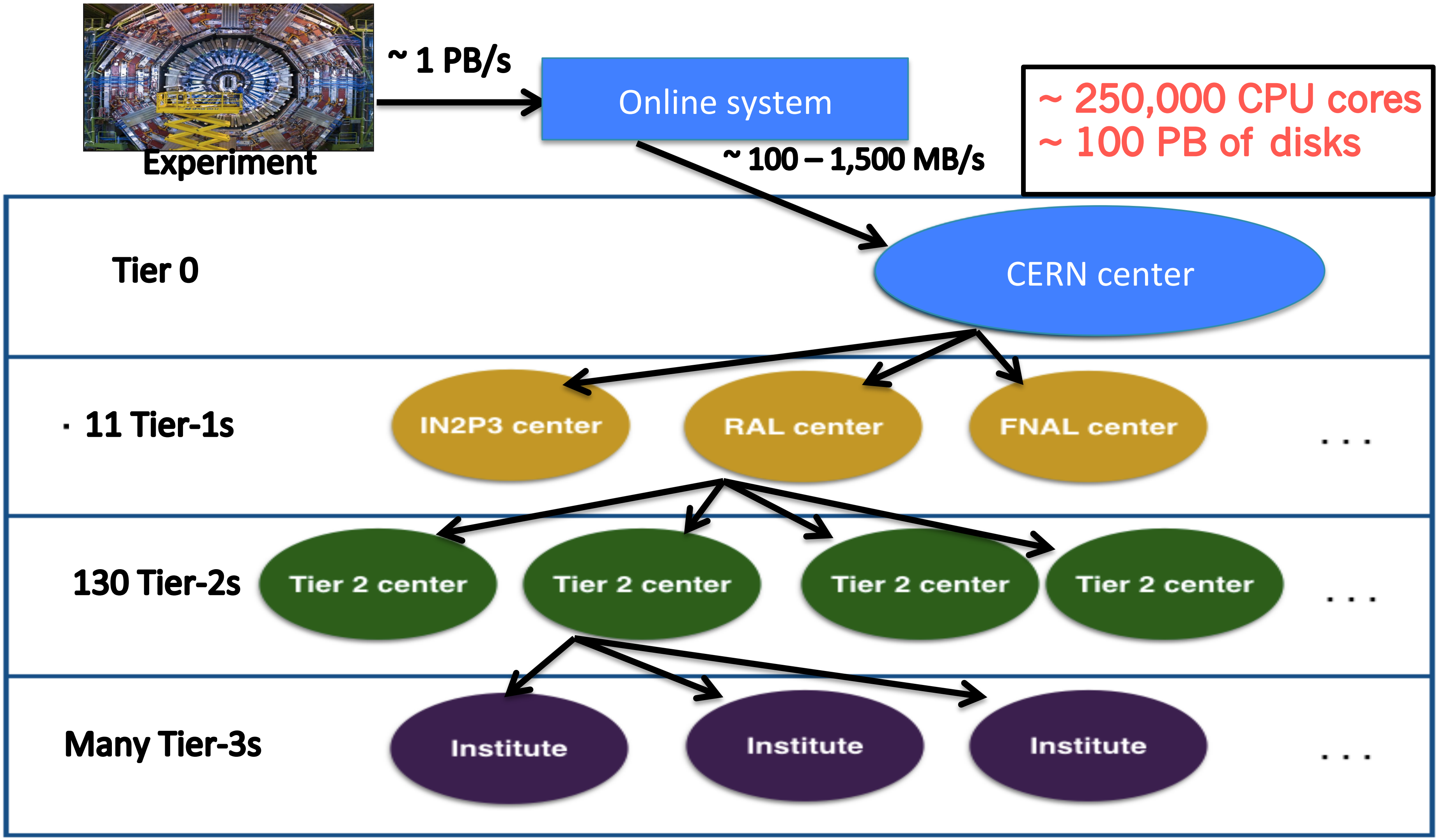
- Experiment distributed computing support for 7 years
- Working in the implementation/integration of solutions for data movement and monitoring for experiments @CERN

CERN and Large Hadron Collider experiments

- The **Large Hadron Collider (LHC)** is a particle accelerator
- It collides beams of protons at an energy of 14 TeV
- It has a circumference of 27km, is located 100mt underground
- It has four major detectors: ALICE, ATLAS, CMS, LHCb

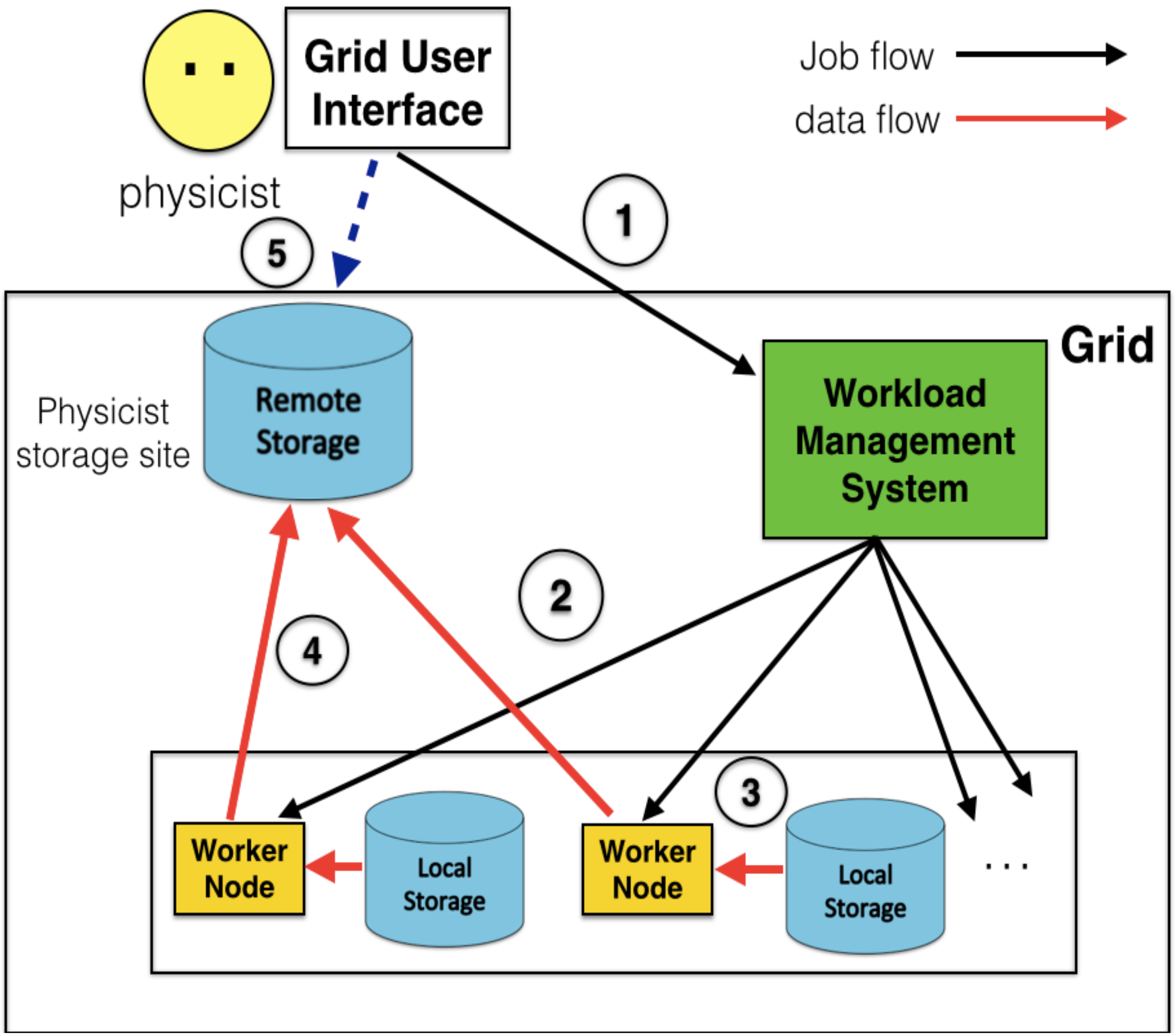


WLCG: Worldwide LHC Computing Grid



Use-case: Distributed data analysis in CMS

- 1000 individual users per month
- More than 60 sites
- 20k jobs/hour
- Typically 1 file/job
  - Files vary in size
- 200k completed jobs per day
- Minimal latencies
- Chaotic environment



Outline

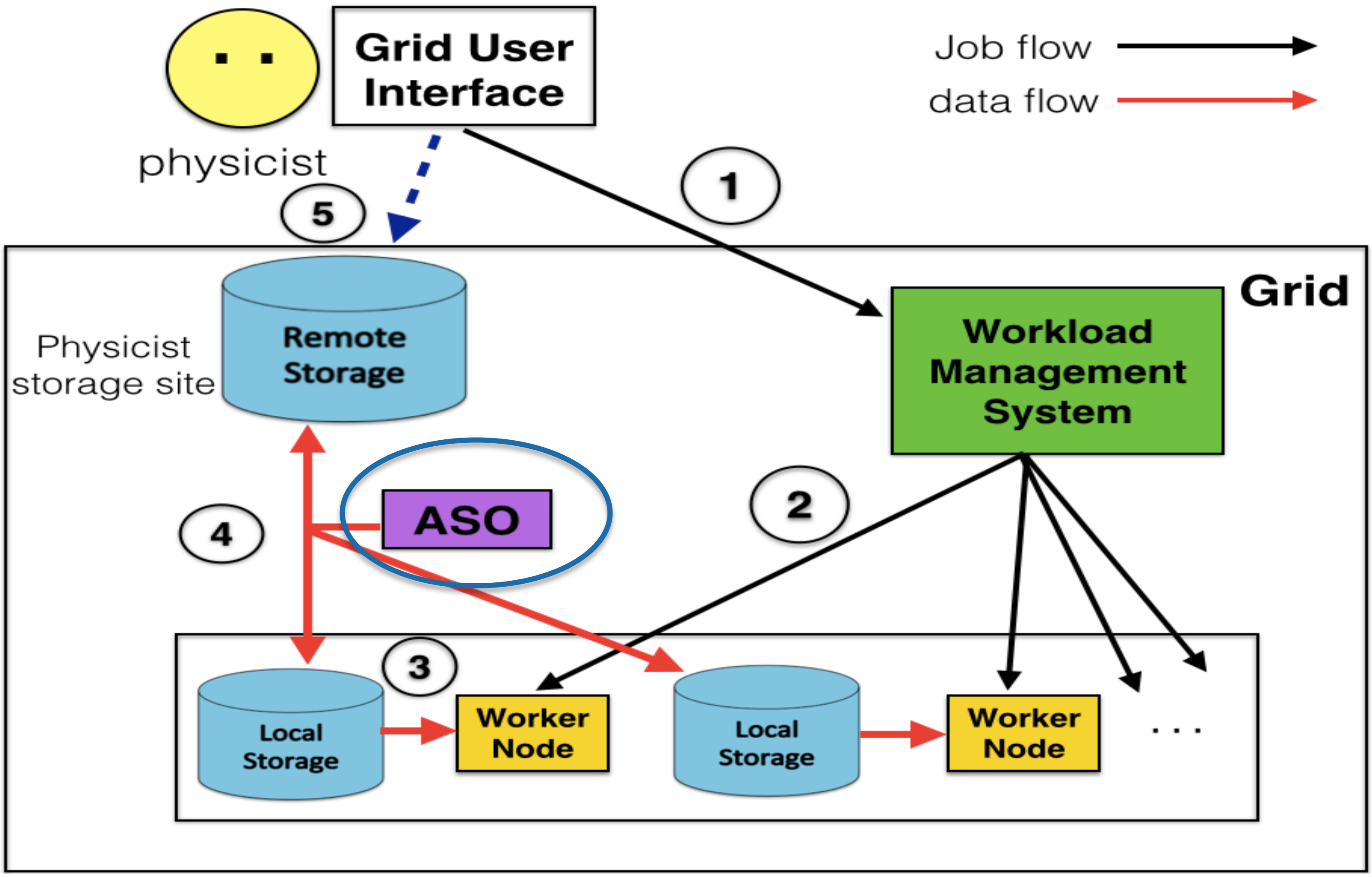
- Context
- **Problematic and strategy**
- System architecture
- Integration and deployment models
- Experience and lessons learnt

Problematic

- 15% to 20% of the jobs fail and about 30 to 50% of the failures are due to the jobs not being able to upload their output data to a remote disk storage
  - Between 5% and 10% of jobs fail in the remote copy of outputs
  - the overall CPU loss is even higher than 5-10% since those jobs fail at the end of the processing
  - often it results in DDoS to CMS Tier-2 storage systems

AsyncStageOut (ASO) is implemented to reduce the most common failure mode of analysis jobs

Asynchronous stage-out strategy



ASO algorithm

1. The analysis jobs copy locally the outputs to the temp area of the local storage
2. The transfer requests are uploaded into ASO from a data source (Worker Nodes, Workload Management system…)
3. The ASO tool:
   1. Creates, schedules and manages jobs to transfer the user files from the local storage to the target destination
   2. Manages the publication of the transferred files into experiment's data catalogue
   3. Updates the status of the file
4. The output is available to the user

Outline

- Context
- Problematic and strategy
- **System architecture**
- Integration and deployment models
- Experience and lessons learnt

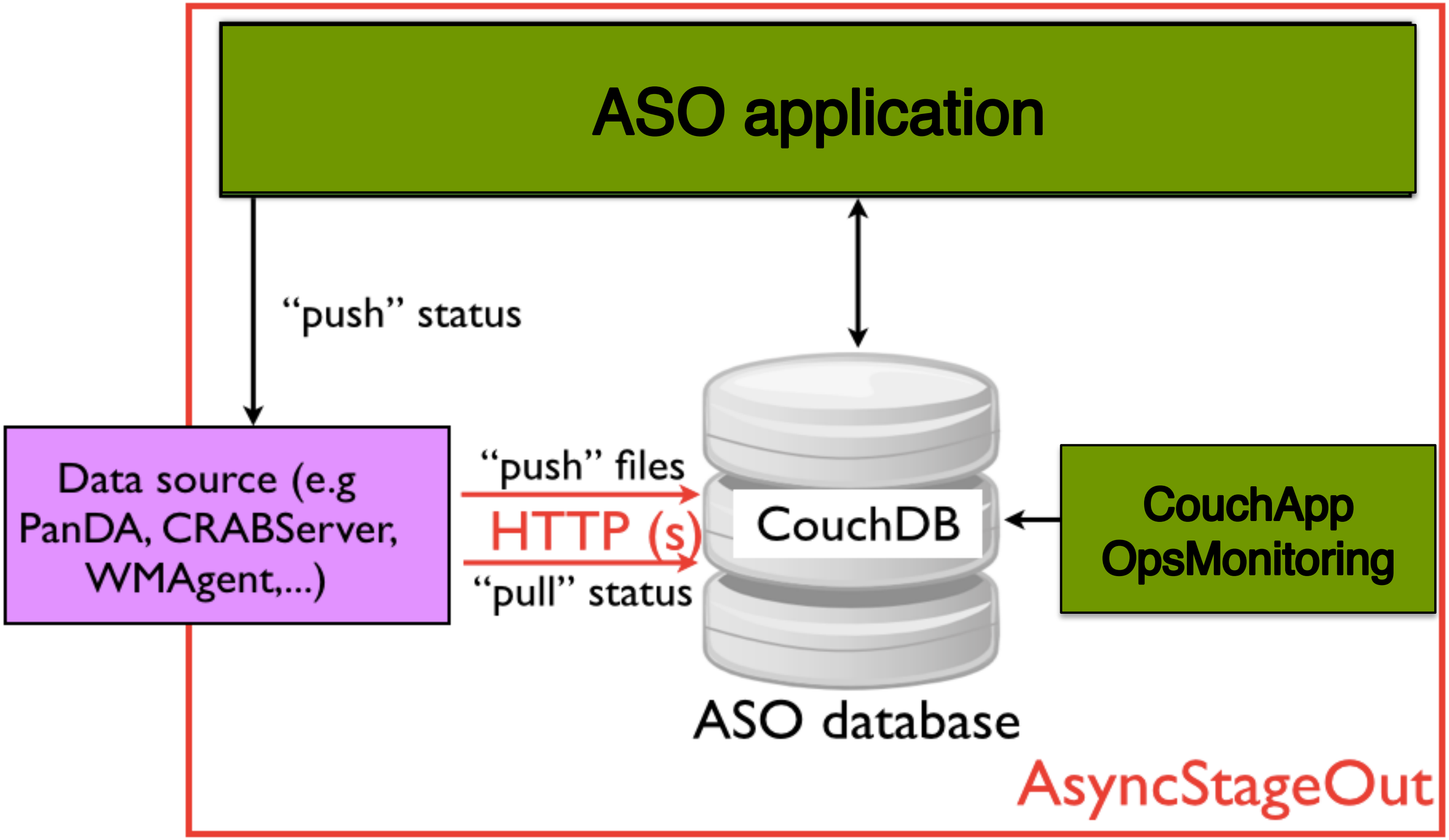
Why CouchDB?

- Fast prototyping of new systems thanks to the schema-less nature of CouchDB
- Fast implementation of the Web monitoring
  - No particular deployment of the monitoring is required since it is encapsulated into CouchDB
- Rapidly incorporate new types of data
- Easy communication with external tools across the CouchDB REST interface
- The easy replication and the integrated caching of CouchDB should provide a highly scalable and available system to face the new challenges

Implementation and technologies

- Implemented in Python as a standalone tool with modular approach
- Organized as set of components loosely coupled and communicating across a database
- Rely only on CouchDB as input and data storage
- Highly configurable tool: max_transfer_retry, max_files_per_transfer, data_source, …
- Plugin-based architecture: data placement and bookkeeping
  - Independence of Grid/Experiment technologies

Architecture



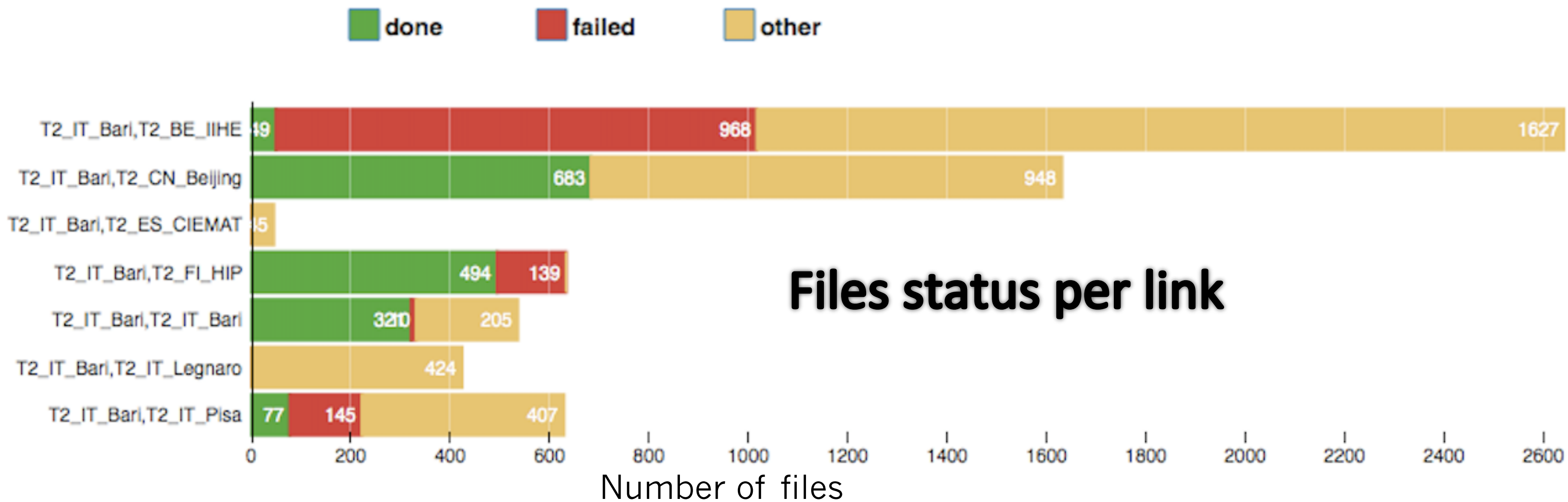
Transfer document

_id	"0000090220b097c8d212e7a29900b45082c5794d6506c59110cd7622"
_rev	"5-1f52d1bdb79f432af8182ad0ea1728fd"
_attachments	
checksums	
destination	"T2_IT_Legnaro"
inputdataset	"/QCD_Pt-1000to1400_Tune4C_13TeV_pyt castor_PU_S14_POSTLS170_V6-v1/AODSI
lfn	"/store/user/riahi/Hbb_104583.root"
publication_state	"not_published"
size	3105000
source	"T2_FR_GRIF_LLR"
start_time	"2014-10-15 19:21:43"
state	"done"
type	"output"
user	"riahi"

Monitoring implementation

- The Map/Reduce views of CouchDB are visualized across Protovis
  - Migration to D3.js is on-going
- The monitoring application is encapsulated into CouchDB server as CouchApp

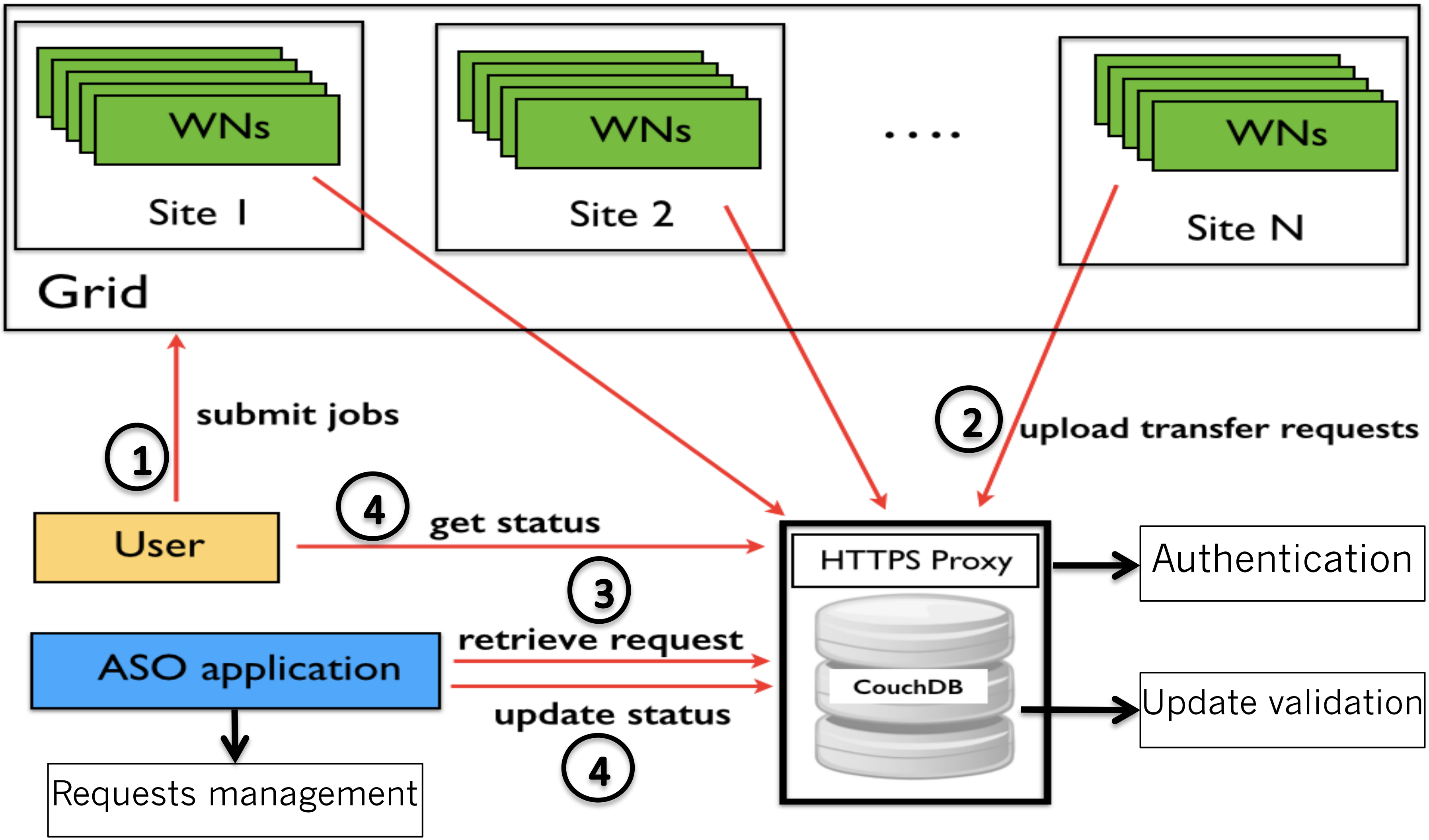
Some monitoring plots



Outline

- Context
- Problematic and strategy
- System architecture
- **Integration and deployment models**
- Experience and lessons learnt

Integration



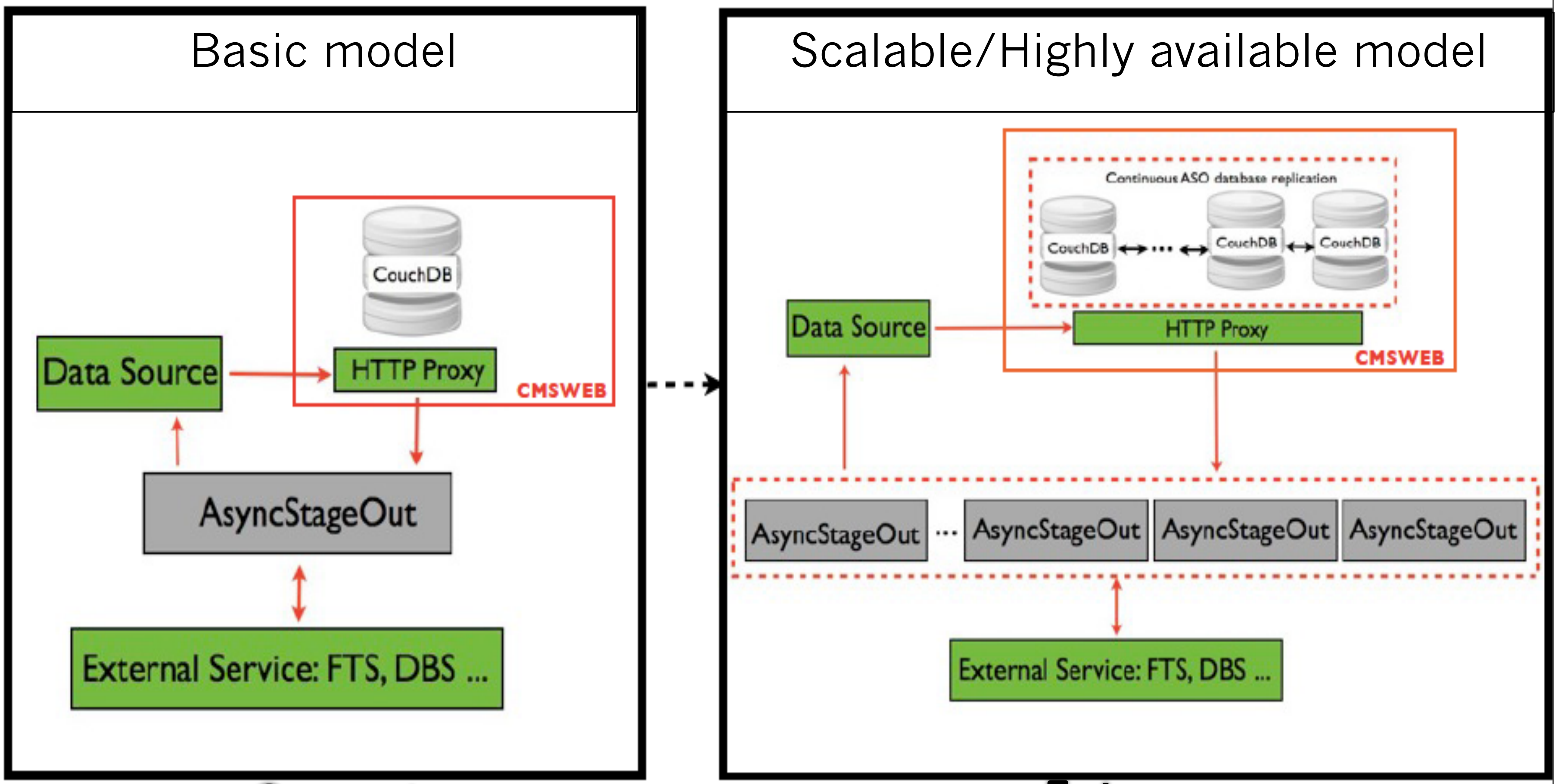
Authentication/Validation

- The authentication with CouchDB is performed using the X509 Proxy Certificate
  - Using custom authentication handler
- Document update validation:

```
// The following rule aplies for all operation types
var allowed = isGlobalAdm ||
              matchesRole("operator", "group:aso") ||
              matchesRole("web-service", "group:facops") ||
              ((newDoc._deleted === true || newDoc.user === userCtx.name) &&
              (!oldDoc || oldDoc.user === userCtx.name));

// Throw if user not validated
if (!allowed) {
    log(toJSON(userCtx));
    throw {
        forbidden : "User not authorized for action."
    };
}
```

Deployment models



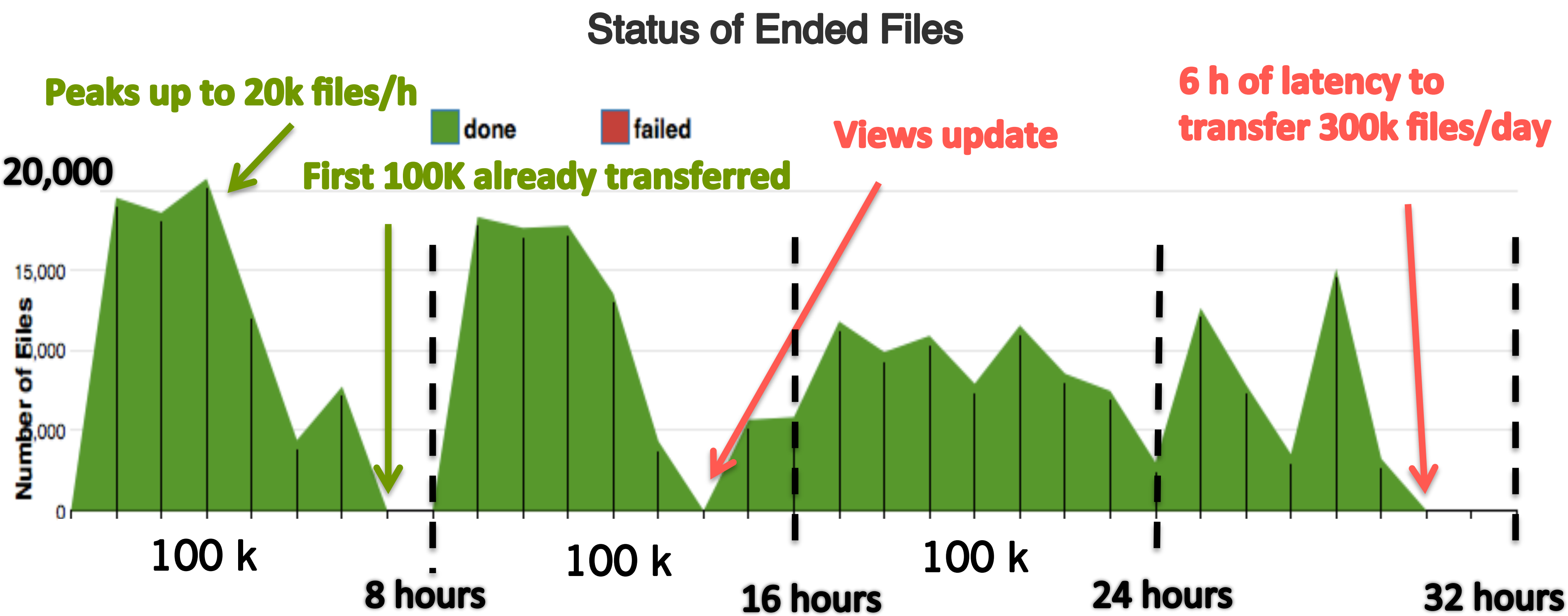
Current **Future**

Outline

- Context
- Motivations and strategy
- System architecture
- Integration and deployment models
- **Experience and lessons learnt**

Commissioning tests

- Application and CouchDB were deployed in **1 VM** with 8 VCPU, 15 GB of RAM and 200 GB of disk storage
- Scale up to **1.5** the **production** load (20 k files/h - 200 k completed files/day)
  - **300k files/day ➡ inject ~ 100k files each 8 hours**

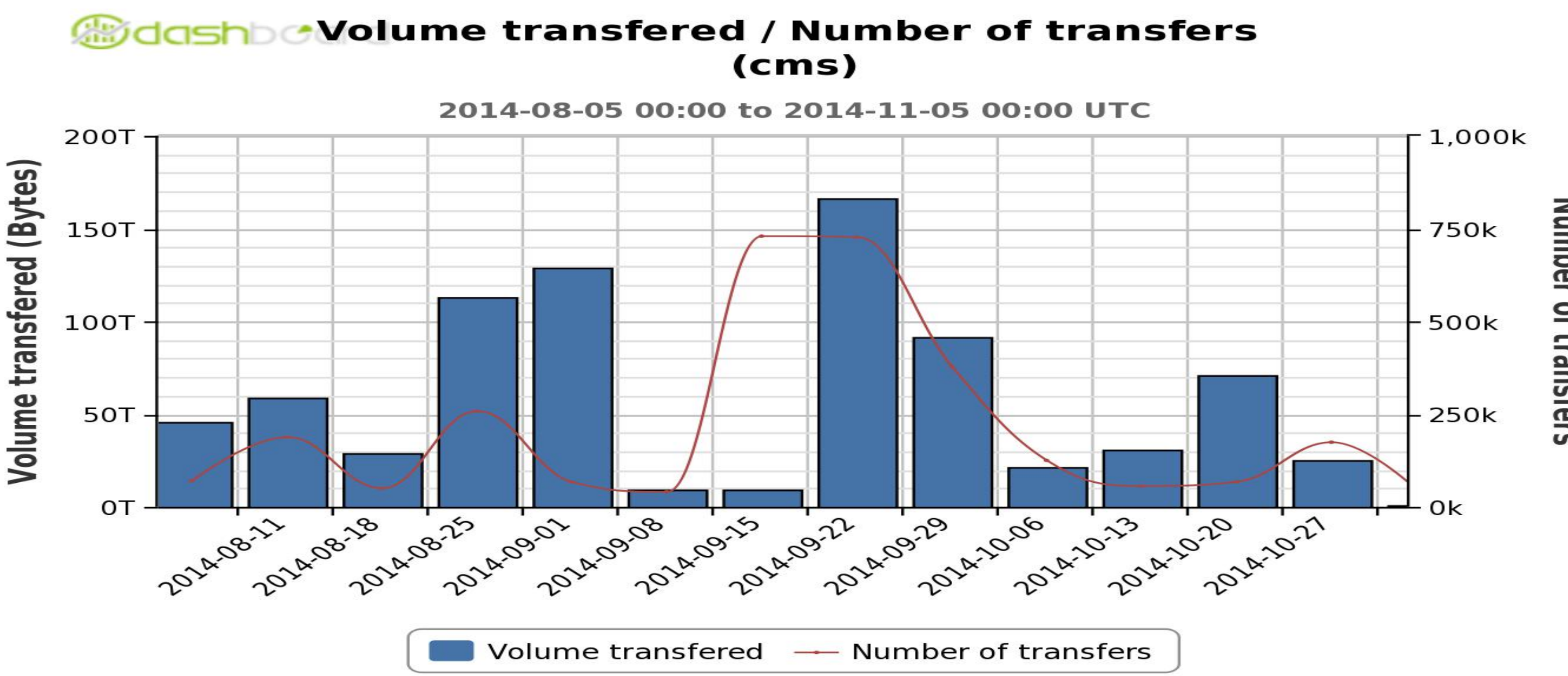


Commissioning experience

- ASO can manage load peak of more than **20k files/h** without critical error or crashes
- Nearly **60 GB of disk storage** were consumed by the CouchDB
  - More than **90 %** has been used for **views caching**
- The average **CPU idle time** was almost stable at **90%**
- The **RAM** was always almost **fully consumed** during the processing time
  - Most probably the delays seen would be reduced by increasing the RAM for accessing the cached views

Production results

- ASO is in production since June 2014
  - More than 800 TB transferred during the last 3 months
    - Peak of 750k files per week



Production environment

- Hardware
  - 2 physical nodes (migration to VMs is ongoing)
  - CouchDB: 8 Cores and 24 GB RAM
  - ASO application: 24 Cores and 32 GB RAM
- ASO Database

Size	Operation
▪ Average database size: 20 GB ▪ 3 Design docs: 33 views ▪ Average number of docs: 800k	▪ Upgrade: 1 time/month ▪ Compaction: 2 times/day

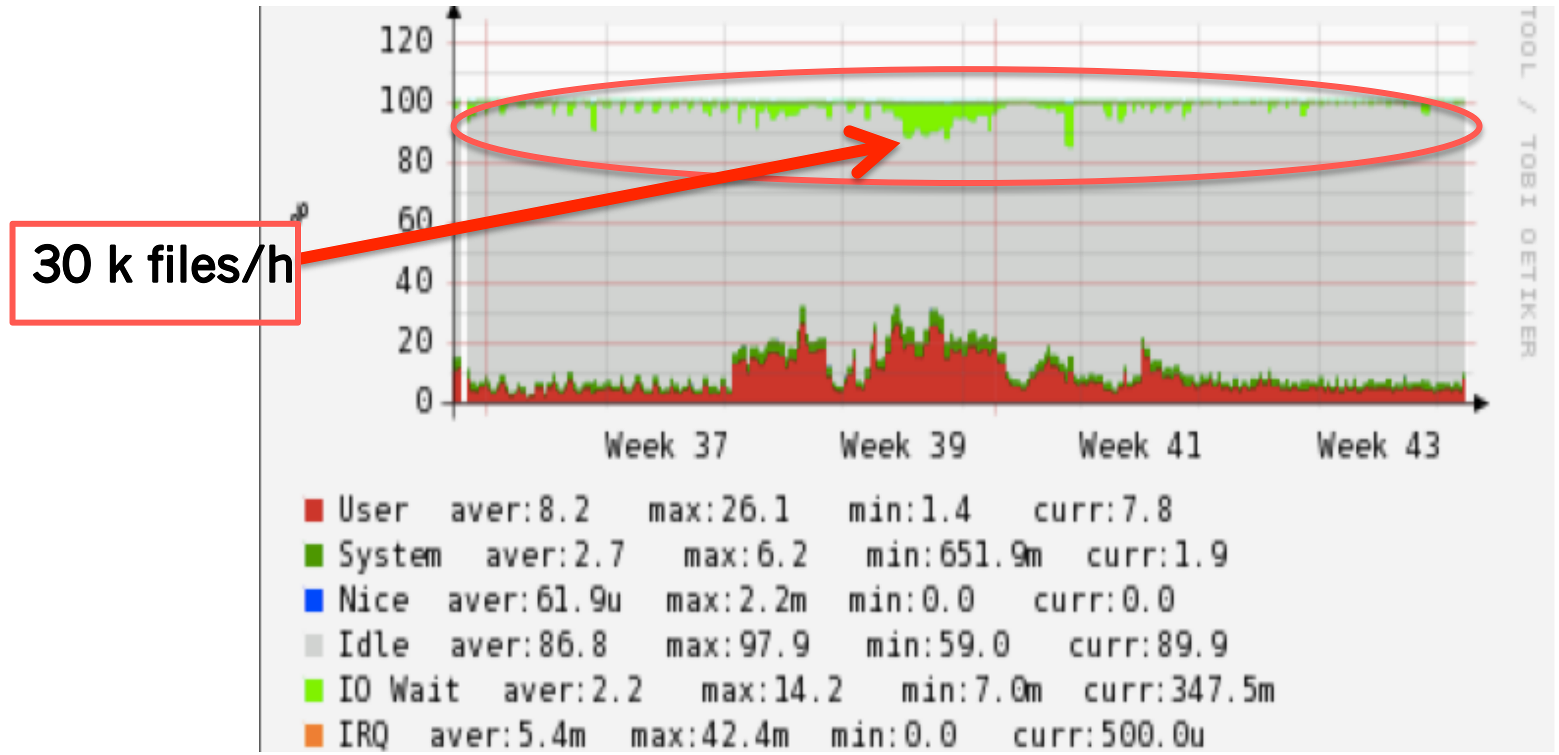
Problem 1: Database upgrade

- During this first phase of production we need to upgrade the database once per month to include new features
- CouchDB spends more than 24 hours for views index generation → ASO is off during this operation
- CMS users cannot perform physics analysis for more than 1 day

Solution

1) Start a fresh CouchDB instance at each upgrade while keeping the old one running
   - Requires development to support load balancing over separated couch instances
   - Increases CouchDB operation efforts
2) Upload the new design document in a replicated database, trigger the view index generation offline and switch once it is completed

Problem 2: Compaction



Often I/O wait time is close to 10 % in 8 cores → frequent I/O bottlenecks

Map function improvement

```
function(doc) {
        ...
        emit([day, hours, minutes], {"state": doc.state, "size": doc.size});
}
```

```
function(doc) {
        ...
        emit([day, hours, minutes, doc.state], doc.size);
}
```

```
function(doc) {
        ...
        emit([day, hours, minutes, doc.state], 1);
}
```

Reduce function improvement

```
function(keys, values, rereduce) {
    var output = {'done': {'njobs': 0, 'size': 0}, 'failed': {'njobs': 0, 'size': 0},
                  'killed': {'njobs': 0, 'size': 0}, 'resubmitted': {'njobs': 0, 'size': 0}};
    var keys = ['done', 'failed', 'killed', 'resubmitted'];
    var values = ['njobs','size'];
    ...
        for (var v in values) {
            var value = values[v];
            var state = value['state'];
            output[state]['njobs'] += 1;
            output[state]['size'] += value['size'];
        }
    }
    return output;
}
```

→ _sum

Results

	Time for Index generation	Total number of views	Views size
Initially	**28 hours**	33	**30 GB**
After views clean up	17 hours	25	17 GB
After views code improvement	**25 minutes**	30	**15 GB**

Conclusions

- Fast system and Web monitoring prototyping
- The system has shown satisfactory performances
- Database operation issues are understood
  - They are mainly addressed by views code improvement
- Promising technology for other applications (data analytics, data mining...)
- Looking forward to your feedbacks and suggestions!

References

- CERN: http://home.web.cern.ch/
- CMS: http://cms.web.cern.ch/
- ASO: https://github.com/dmwm/AsyncStageout
- Protovis: http://mbostock.github.io/protovis
- D3js: http://d3js.org/

Thank you for your attention!

hassen.riahi@cern.ch