Russian Data Lake Prototype as an Approach Towards National Federated Storage for Megascience

Aleksandr Alekseev, Andrey Kiryanov, Alexei Klimentov, Tatiana Korchuganova, Valery Mitsyn, Danila Oleynik, Sergey Smirnov, Andrey Zarochentsev

Background

HENP experiments are preparing for HL-LHC era, which will bring an unprecedented volume of scientific data. This data will need to be stored and processed by collaborations, but expected resources growth is nowhere near extrapolated requirements of existing models both in storage volume and compute power.

=> *Computing models need to evolve.*

This evolution includes multiple aspects:

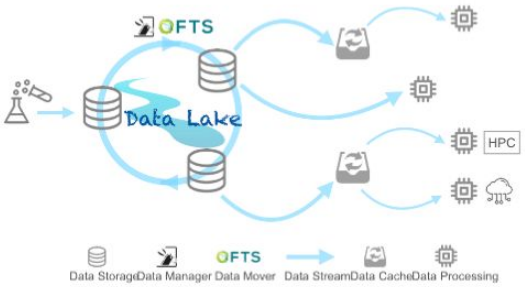
- Optimized data processing, squeezing the maximum from available CPU/GPGPU/FPGA resources
- Optimized data storage, reduction of the number of copies, different data access methods, full utilization of network resources
- Cost optimizations, no high-end expensive RAID setups, no underutilized CPUs on storage servers, no HDDs with 90% free space on the worker nodes
- Deployment optimizations, scalability and containerization with on-demand expansion into the cloud (both community and commercial)
- Operational cost optimization, more standardized solutions, lower requirements on unique Grid expertise

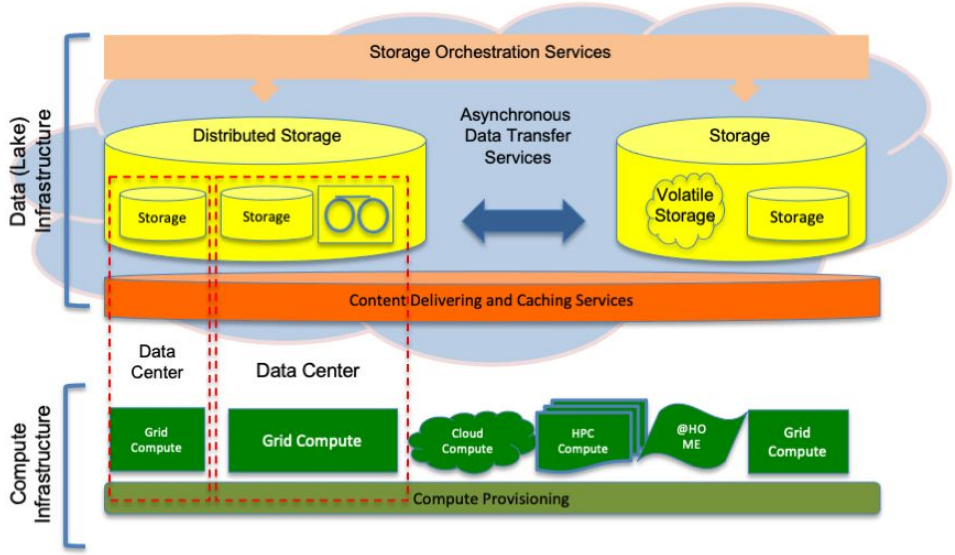
Why are we doing it?

Moving to a network-centric model

Datalake model:

Fewer number of facilities operating storage services, less data replication

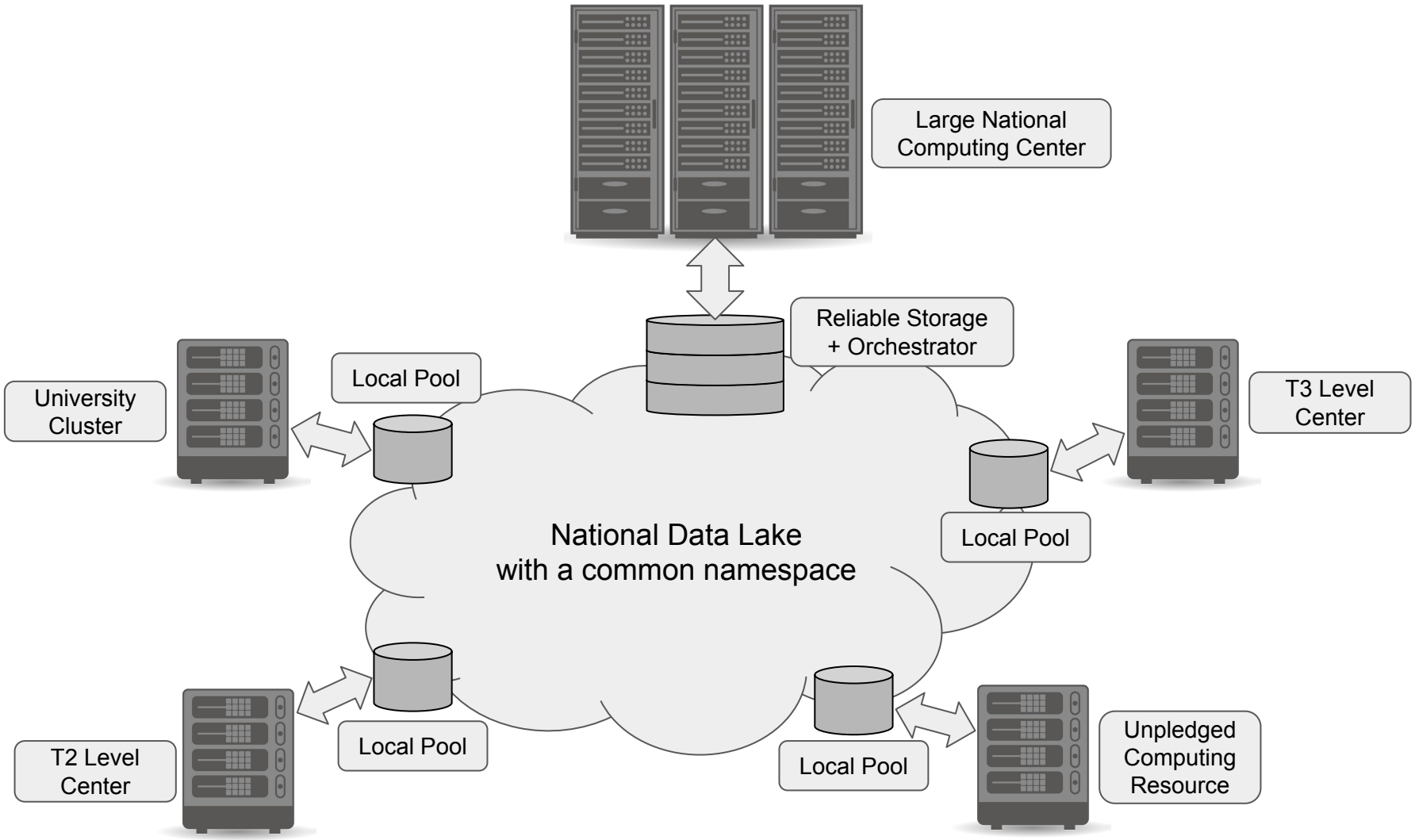




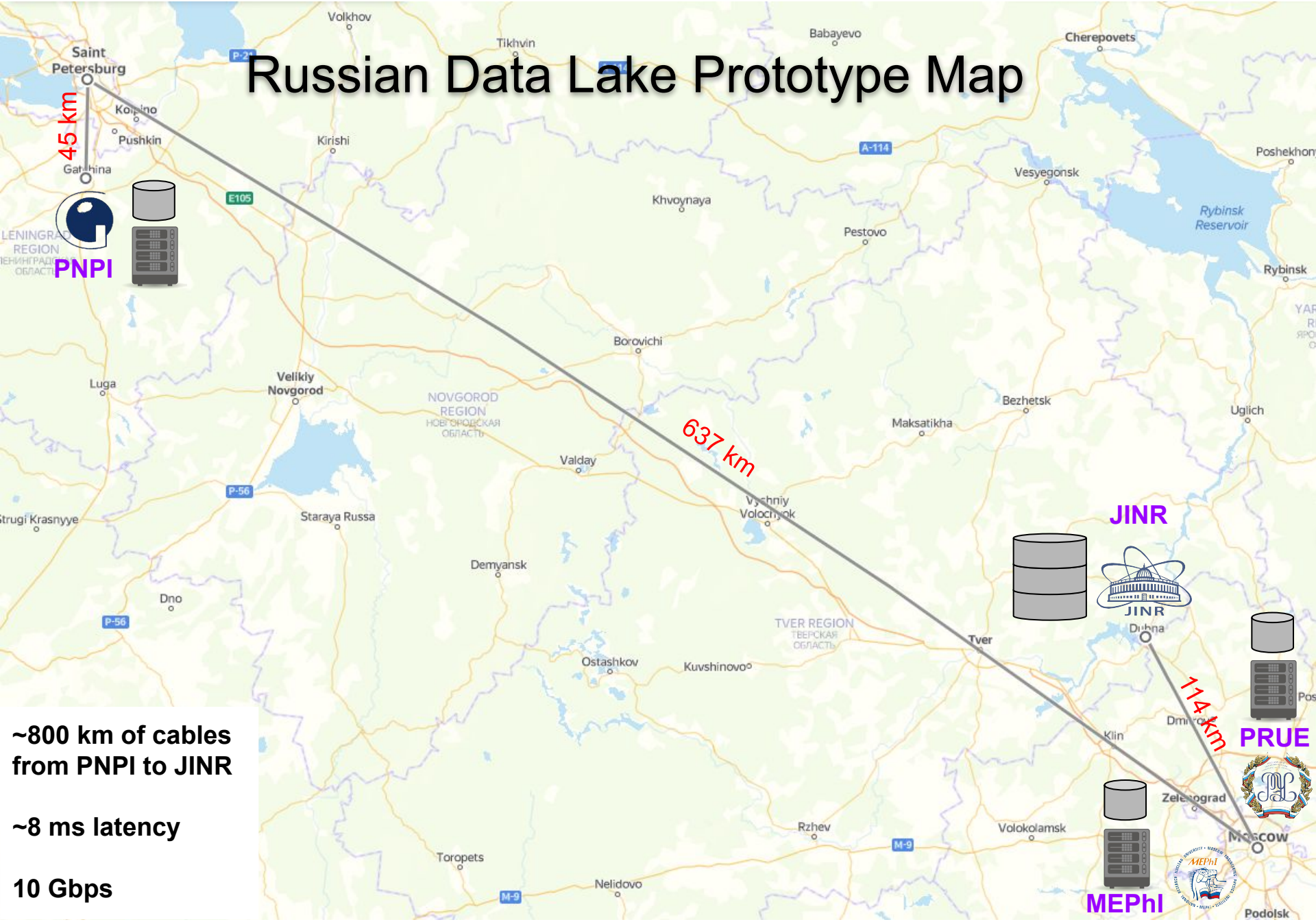
CPUs and storage not necessarily co-located: need to deliver the content over the WAN and/or cache it

From Simone Campana's talk on Tuesday

Data Lake with CPU-oriented smaller sites



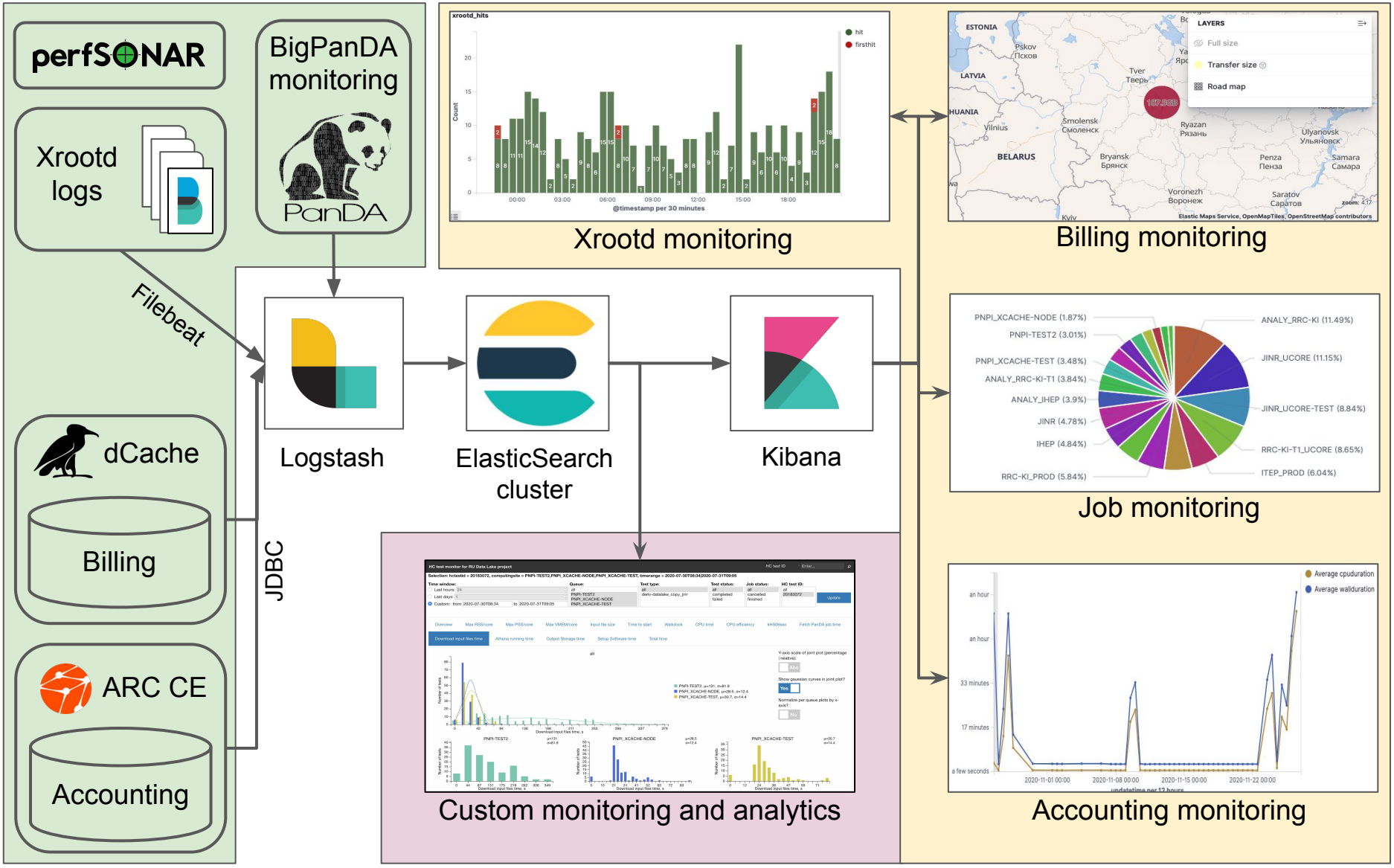
Russian Data Lake Prototype Map



Russian Data Lake Prototype Building Blocks

- Resources
  - Bare-metal at JINR and MEPhI
  - Virtualized at PNPI and PRUE
- Storage systems
  - EOS
  - dCache
  - XCache
- Payloads configuration, submission and testing
  - Custom synthetic tests
  - PanDA and ProdSys2 (ATLAS)
  - HammerCloud
  - CRIC (former AGIS)
- Monitoring infrastructure
  - perfSONAR
  - Logstash
  - ElasticSearch
  - Kibana
  - Custom web apps

Russian Data Lake Prototype Monitoring

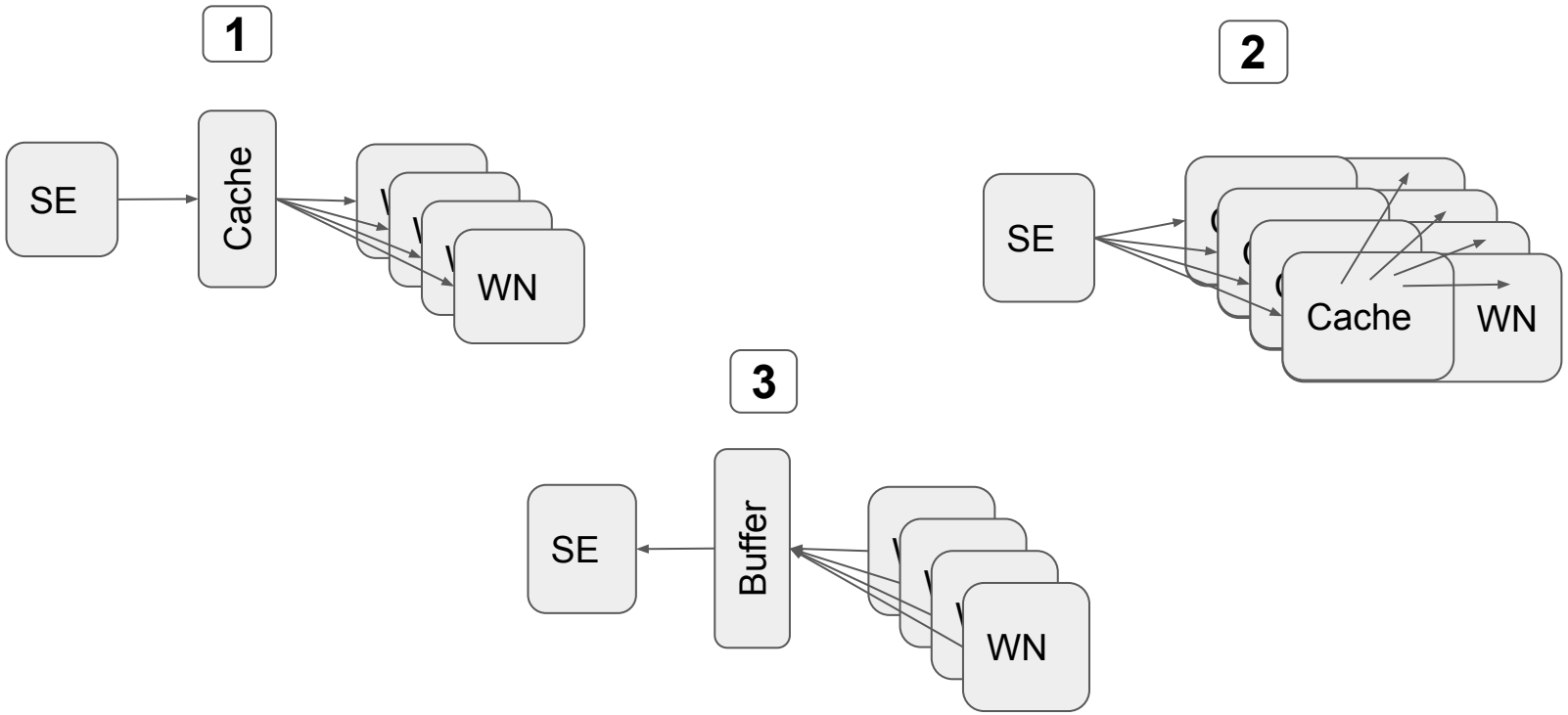


Custom monitoring and analytics

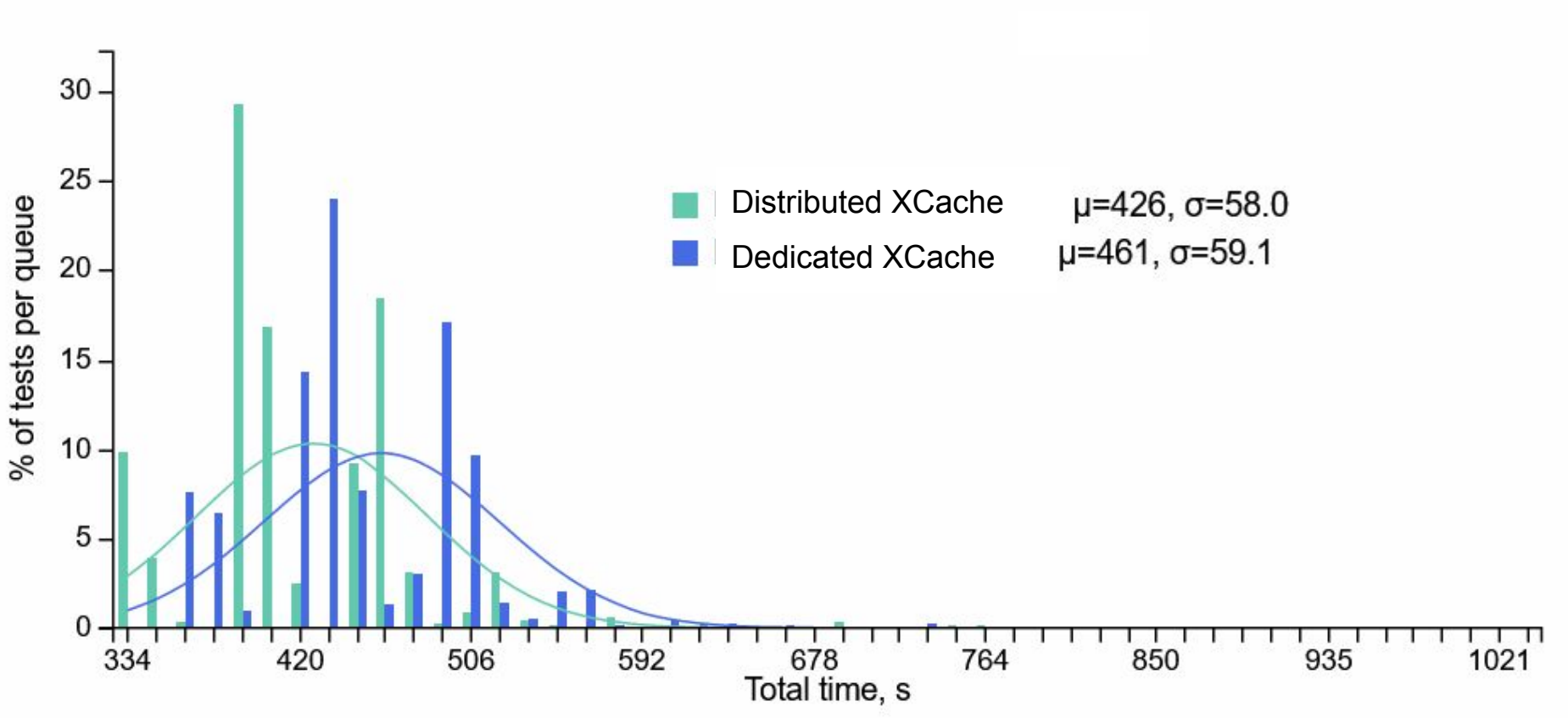
- A custom web application for HammerCloud monitoring (to get round Kibana limitations)
- Job selection by a rich spectrum of criterias

Data caching and buffering

1. Cache on a dedicated server (XCache)
2. Distributed cache on the worker nodes (XCache)
3. Buffer on a dedicated server (EOS)

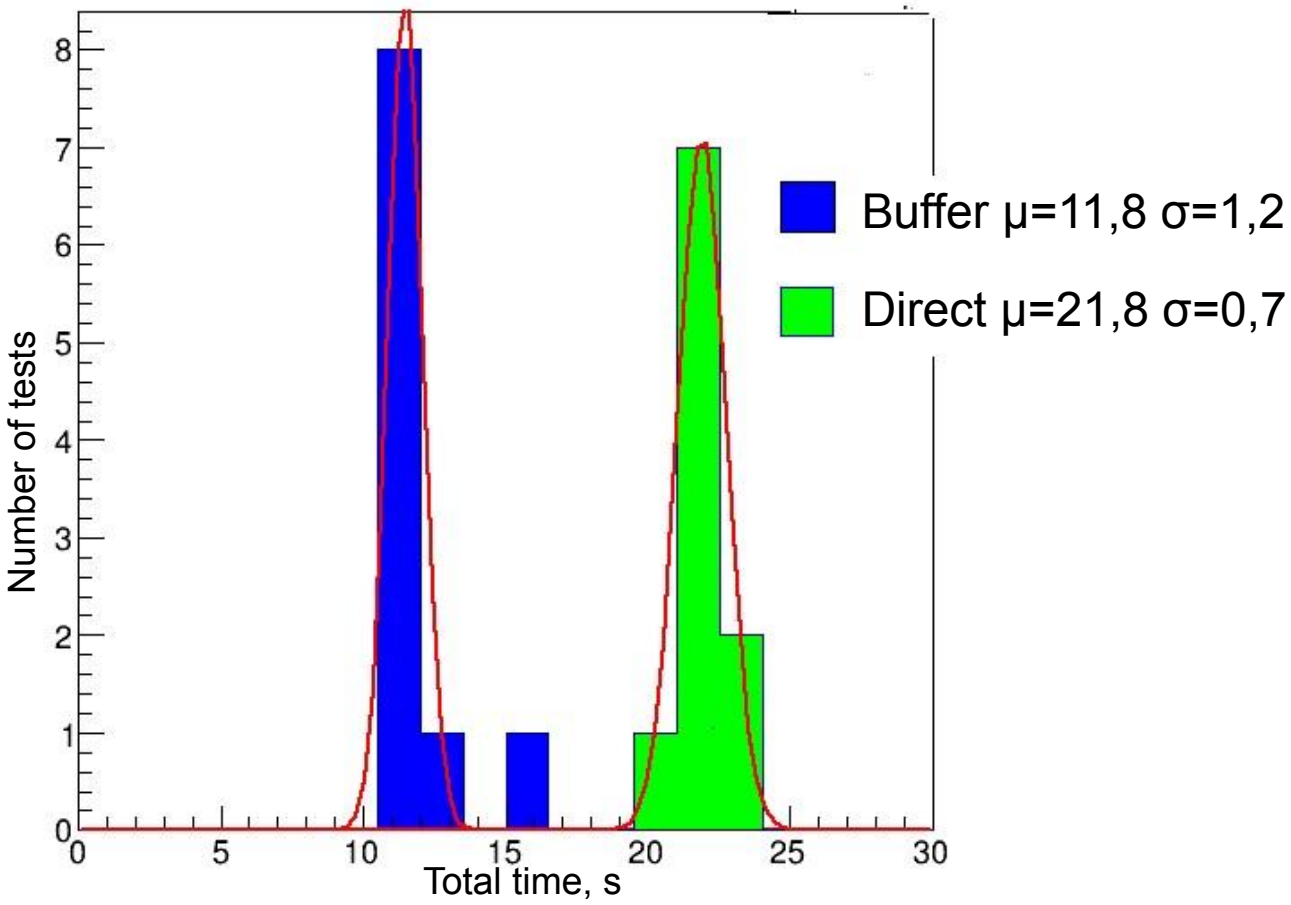


Caching for read-oriented workloads



Total time of tests with dedicated XCache vs distributed XCache
Difference is within the margin of error

Buffering for write-oriented workloads



Total time of tests with direct write vs buffered write
Using EOS LRU

Conclusions

- We have advanced in building a Russian Data Lake prototype using existing network infrastructure and production-grade resources
- We have deployed an extensive monitoring infrastructure that allows us to gather most of the necessary metrics
- We have conducted various types of tests on our way to measure the impact of caching and buffering on improving CPU efficiency
    - This is a work in progress, we have plans on expanding our tests to other real-life workloads
- We are working on extending our prototype to more Russian sites, but decent network connectivity remains a crucial requirement

Acknowledgements:

This work was funded in part by the Russian Science Foundation under contract No.19-71-30008 (research is conducted in Plekhanov Russian University of Economics)

Thank you!

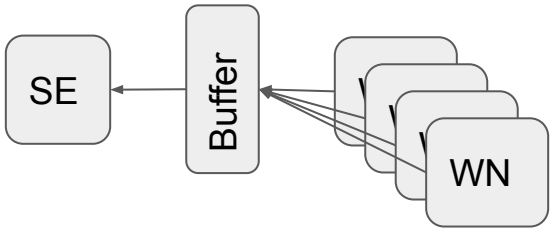
Backup

Write-oriented payloads: Buffering

In order to save CPU time we need to offload files to the remote storage in the background, and we've found a way to achieve this using built-in EOS features:

- We create a special directory in the global namespace
- Default placement policy for this directory is to store all files on the site-local pool
- EOS will pick up new files created in this directory and change their placement policy to a new one
- New placement policy will force flles to migrate to the remote pool

This way we offload all remote writes to the storage engine and free up CPU time on the worker nodes

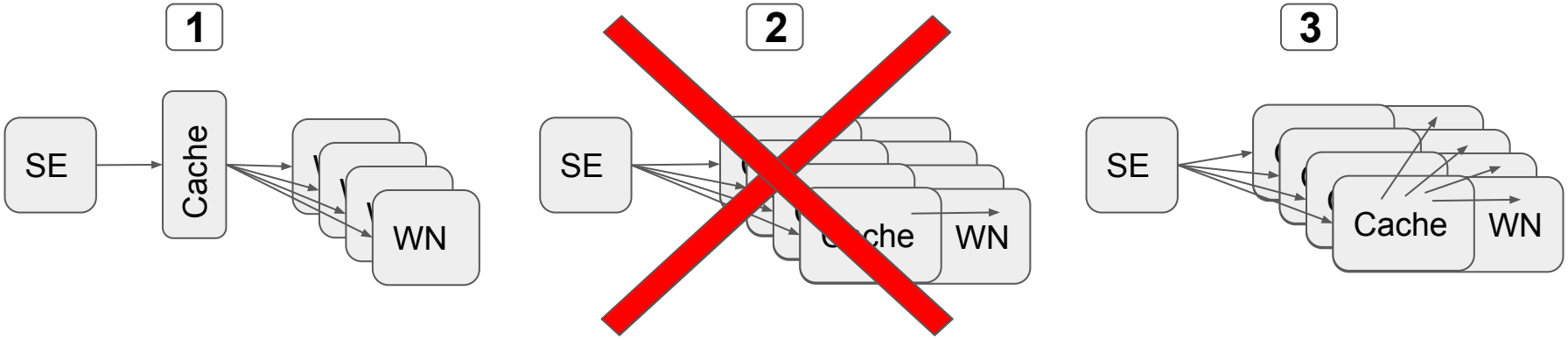


With our synthetic tests we've achieved up to 1.8x speedup

Read-oriented payloads: Caching

There's no one-size-fits-all solution because of hardware (especially network throughput and configuration) differences on different sites. We were considering three pretty obvious scenarios:

1. A single dedicated cache server (poor external network, good internal)
2. A local isolated cache on every worker node (good external network, poor internal) – *this scenario was dropped almost immediately*
3. A shared cache between worker nodes (external and internal networks of the same quality) – *requires some sort of service discovery*



With our synthetic tests we've achieved up to 2.4x speedup

With real-life ATLAS workloads we've achieved a 1.2x speedup