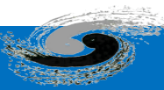


# A computational storage plugin implemented in EOS to support in-situ data processing on storage servers

CC-IHEP, CAS  
Yaodong Cheng  
chyd@ihep.ac.cn

# Outline

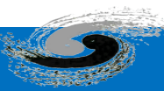
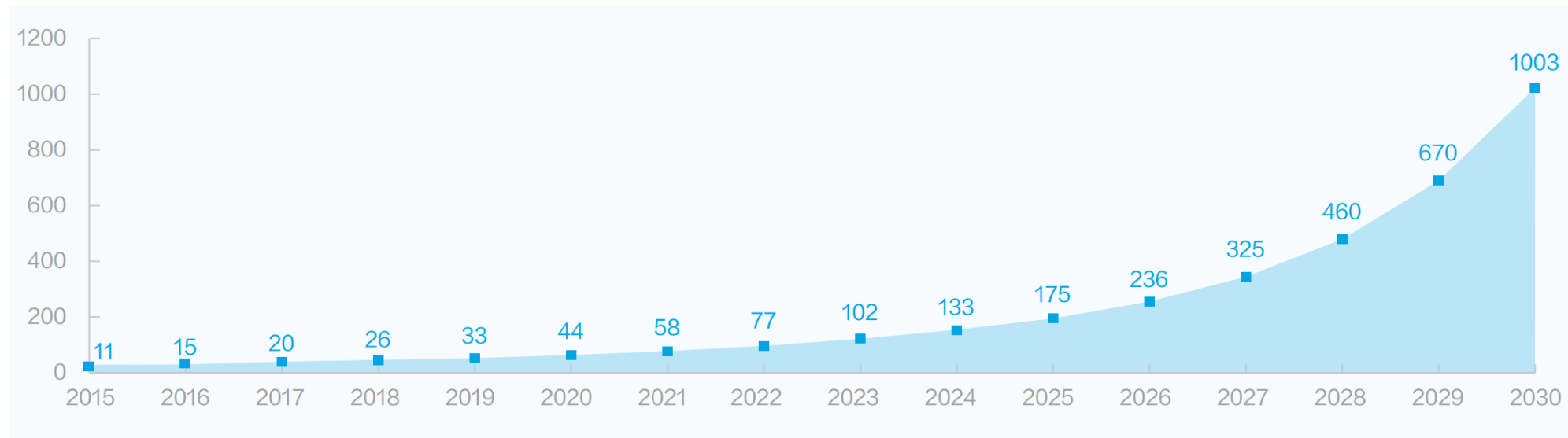
- Background and motivation
- Architecture design and implementation
- Deployment and usage
- Some use cases
- Conclusion



# Background - Data exploration

- The data generated worldwide will reach yottabyte (YB) every year by 2030
  - Driven by large scientific experiments, IoT, smart vehicle, biomedical, new energy, AIGC, ...
  - It is difficult to move data due to too large volume
- If the data move is reduced, it will
  - Save energy
  - Save network bandwidth
  - Reduce the load of host CPU

**Process data close/near/in storage**



# Computational Storage

- Move compute to the data instead of data to the compute

- Value

- Less data transferred on the network
- Faster response times (low latency)
- Improved security; Energy Efficiency

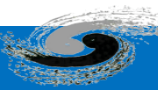
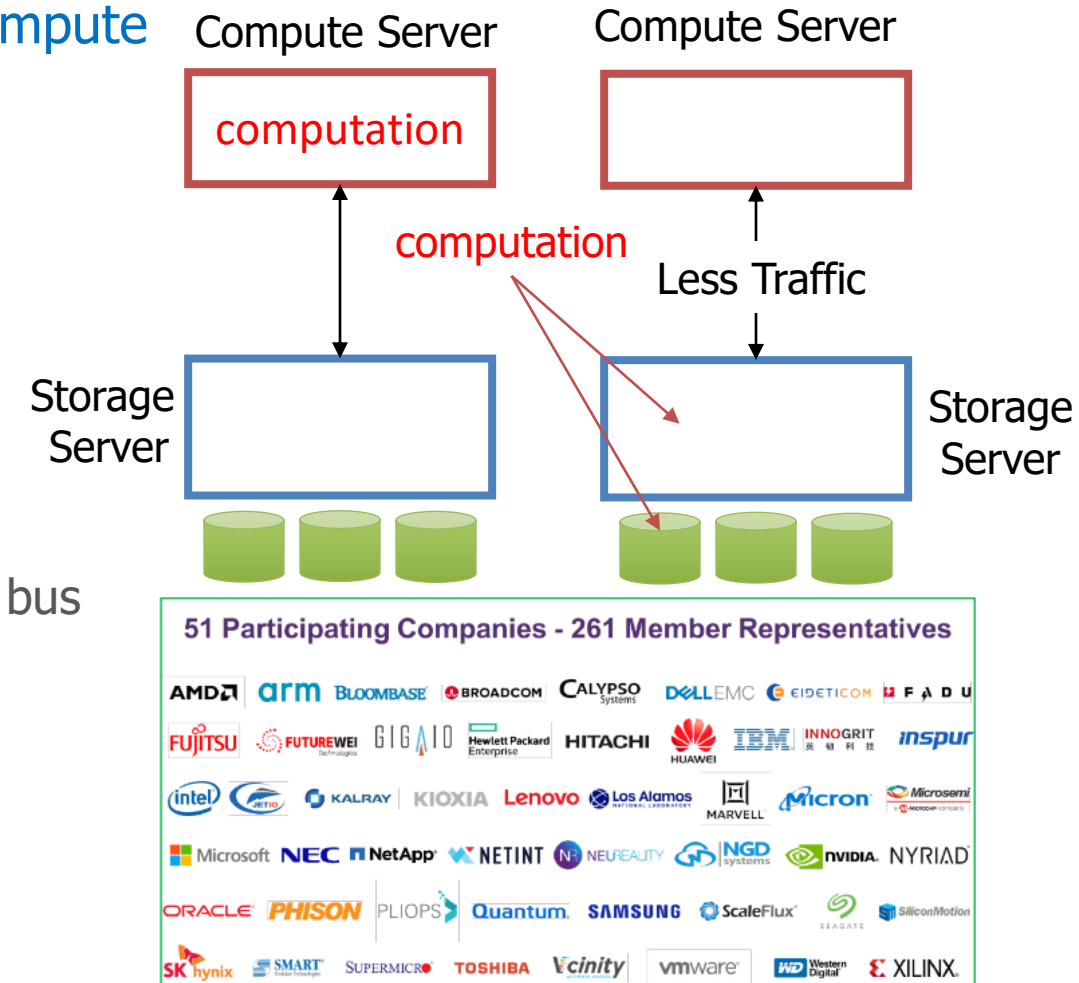
- Different Architectures

- **CSD**: Move compute into the drive
- **CSA**: Move compute into the storage array
- **CSP**: Compute platform on the PCI-E/NVMe/NVMeoF bus

- Standards

- SNIA TWG (Storage Networking Industry Association Computational Technical Work Group)

- **Computational Storage Architecture and Programming Model v1.0** published August 2022



# Some cases of computational storage

- SmartSSD

- Put FPGA into SSD, supporting compression, AI inferencing, ...

- Database acceleration

- Scans and aggregations close to data. **POLARDB** [FAST'20]



- File system offload

- Functions such as indexing and metadata operations in SmartSSD
- **KevinFS** [USENIX OSDI'21] (<https://github.com/dgist-datalab/kevin>)

- Computation offload from compute node

- push down structured queries from compute node to storage server
- **SkyhookDM** [FAST'20] (<https://iris-hep.org/projects/skyhookdm.html>)



SkyhookDM

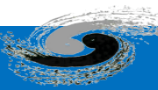
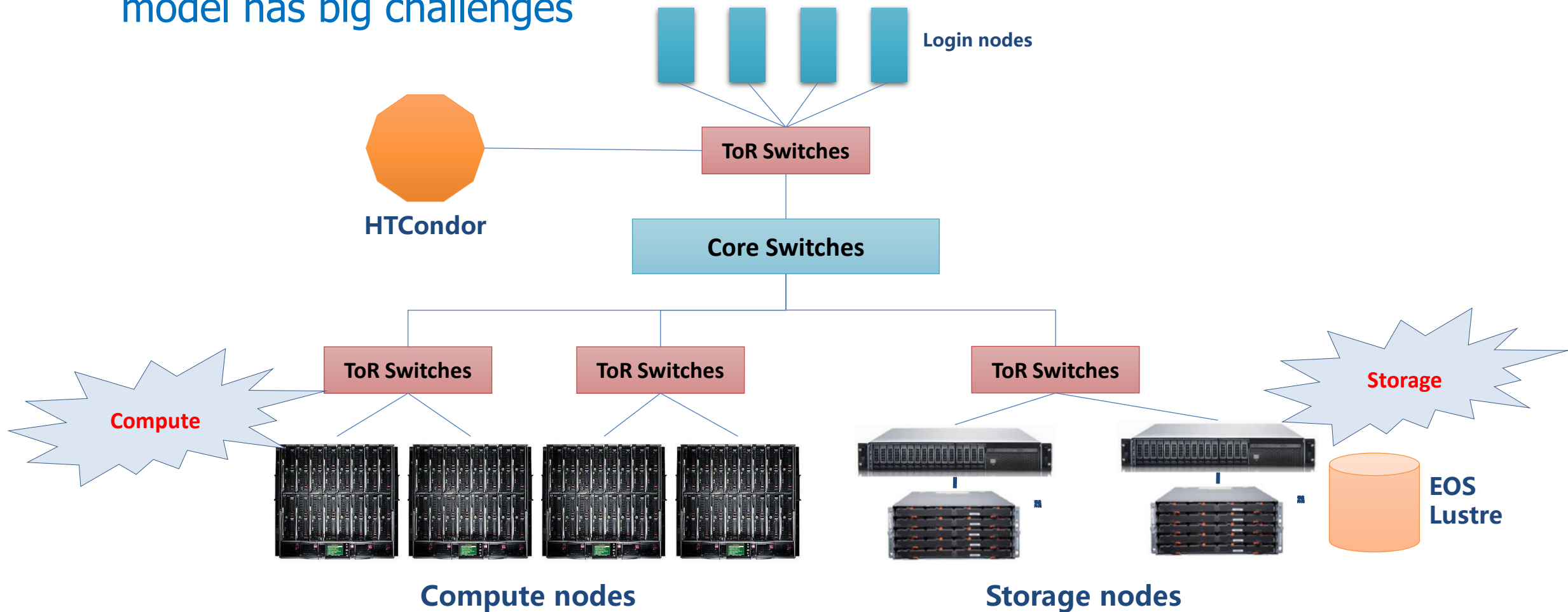
- Neural network acceleration

- HolisticGNN [Fast'22], RecSSD [ASPLOS'21], RM-SSD [HPCA'22]

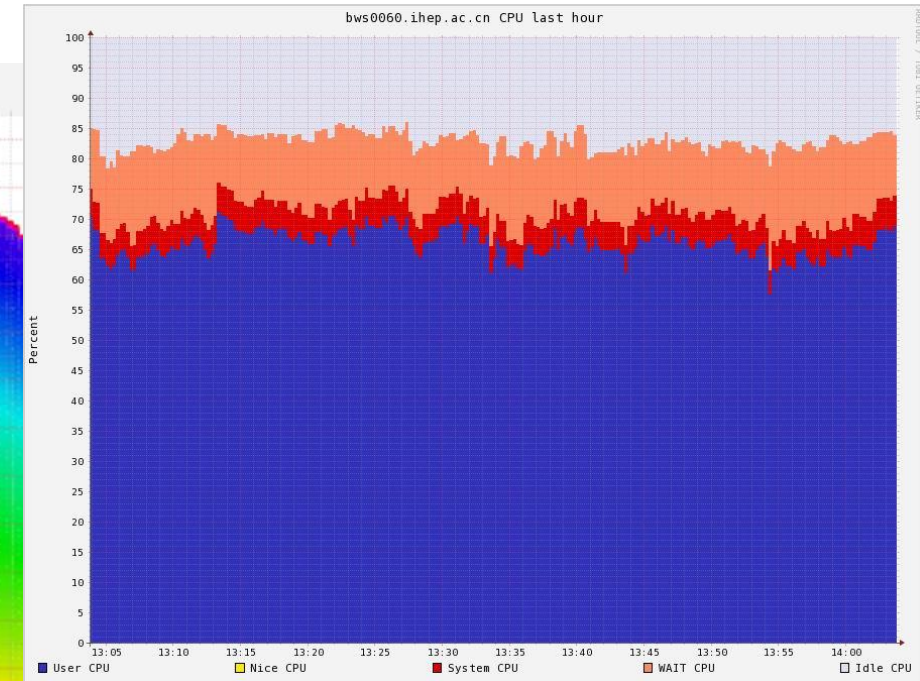
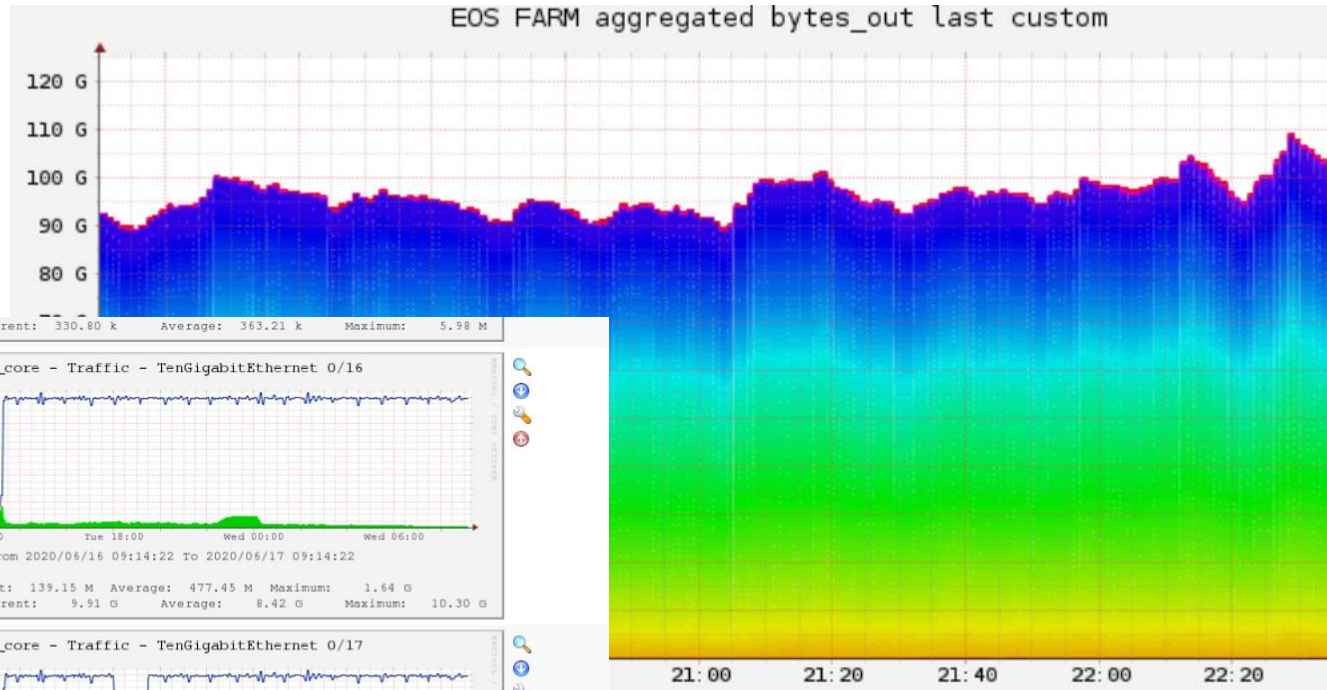


# Why do we need Computational Storage

- Our current network-centric architecture and compute-storage separation model has big challenges



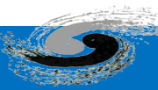
# Problems



**The over-crowded network brings instability and low performance of distributed file systems**

**CPU efficiency is low on compute nodes due to IO bottleneck**

**Overloading of switch ports leads to package loss**

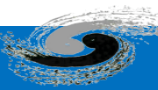
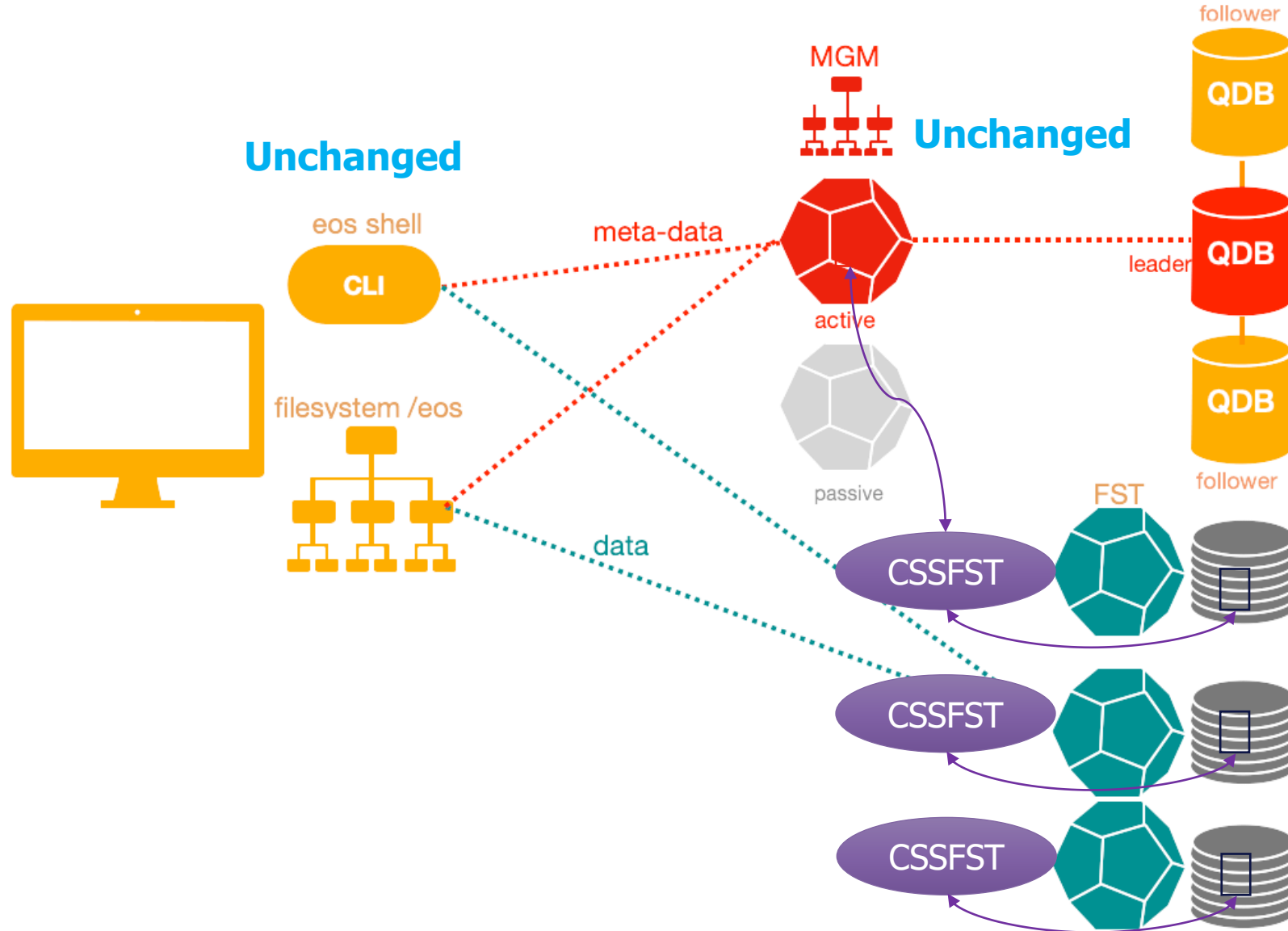


# Solution considerations

- Different computational storage solutions
  - **CSD:** many products available in the market, such as SmartSSD by Samsung, WD NGD, etc. High performance, but with Small capacity, high price.
  - **CSA:** Not yet find products
  - **CSP:** Use computing resources such as GPU, CPU/SoC, FPGA on storage node as a processor. Easy to expand storage capacity but still need exchange data between CSP and storage devices using PCIe.
- Computational storage solution based on EOS
  - The CPU or other resources (GPU, FPGA, etc) in storage server work as **CSP**
  - The name of task to be executed on storage server is passed through **XRootd client**  
*Open("root://eos01/eos/data.txt") → Open("root://eos01/eos/data.txt?css=sort")*
  - The task running on **FST** node reads or writes local disk and update **MGM** information after it finishes

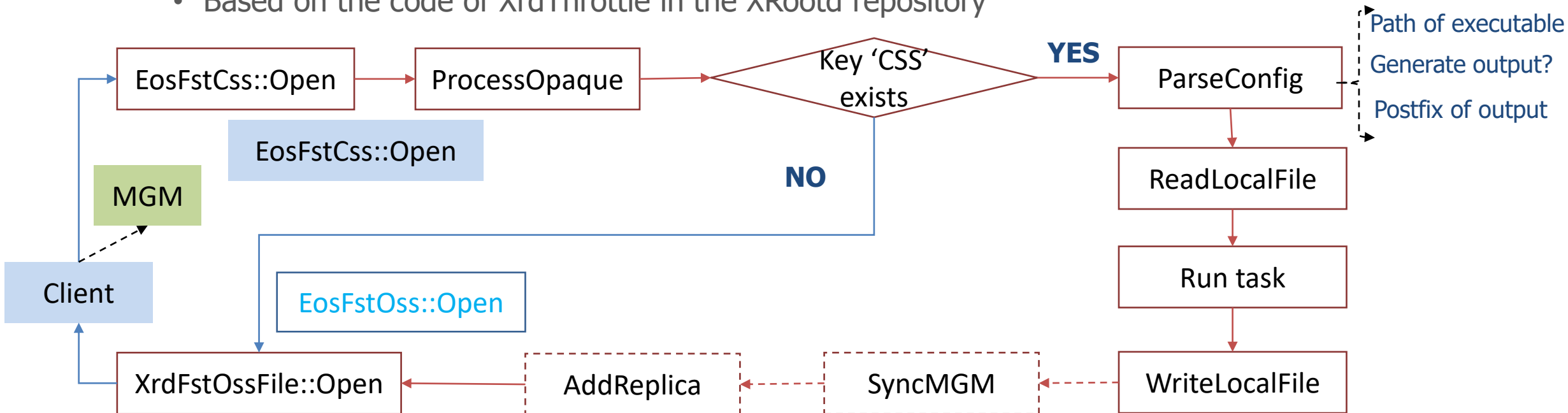


# Architecture



# Implementation

- Write a plugin EosFstCss for FST, which doesn't need to modify any code of FST
  - Based on the code of XrdThrottle in the XRootd repository



- If the task doesn't generate output file, such as 'sort' function, SyncMGM and AddReplica are not necessary, and stdout&stderr will be written into the local file, then READ operation will get them
- The naming of the local output file is based on EOS rule, ie. fid/10000
- The code is hosted on IHEP GitLab, and has been tested with EOS4.8 and EOS5.1

<https://code.ihep.ac.cn/storage/eoscss/cssfst.git>



# Deployment in FST

- Install cssfst rpm package in FST server
- Modify xrd.cf.fst configuration file

```
xrootd.fslib -2 libXrdEosFst.so
```

```
xrootd.fslib -2 libEosFstCss.so -2 libXrdEosFst.so
```

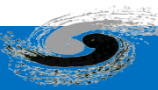
- Edit /etc/eoscscs.conf and customize computational storage functions

```
{ "sort" : {  
  "name" : "sort",  
  "path" : "/usr/local/libexec/cssfst/sort.sh",  
  "out" : false },  
"km2a_decode" : {  
  "name" : "km2a-decode",  
  "path" : "/usr/local/libexec/cssfst/km2a-decode.sh",  
  "out" : true,  
  "postfix" : "root"},  
"zstd": {  
  "name": "zstd",  
  "path" : "/usr/local/libexec/cssfst/zstd.sh",  
  "out" : true,  
  "postfix" : "zst" } }
```

- The executable shell is written and deployed by administrator, which can use container (docker, singularity, ...)
- All the executable shell should given one input file and one output file, and return "EXEC\_SUCCESS" or EXEC\_FAILED

```
#cat sort.sh  
/usr/bin/sort -n $1 > $2  
if [ $? -eq 0 ];then  
  echo "EXEC_SUCCESS"  
exit 0  
echo "EXEC_FAILED"  
exit 1
```

```
#cat km2a-decode.sh  
container=/usr/local/libexec/cssfst/km2adecode.sif  
apptainer exec --bind $dirn $container  
/root/km2a/km2a-decode/decode_sort $1 $2  
if [ $? -eq 0 ];then  
  echo "EXEC_SUCCESS"  
exit 0  
echo "EXEC_FAILED"  
exit 1
```



# How to use

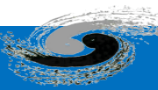
- Two methods to use it, xrdcp or a dedicated client cssclient
- 1) use xrdcp, appending CSS function name into file path  
xrdcp root://eosbak02.ihep.ac.cn//eos/user/chyd/data.txt?css=sort -
- 2) use cssclient tool, which is a wrapper of XrdPosixXrootd (Open, Read)  
export EOS\_MGM\_URL=root://eosbak02.ihep.ac.cn/  
cssclient -f /eos/user/chyd/data.txt -c sort

```
[root@eosbak02 css]# xrdcp root://eosbak02.ihep.ac.cn//eos/user/chyd/data.txt -
1-01, 266.0
1-06, 145.9
2-04, 188.8
1-03, 183.1
2-01, 122.2
2-03, 199.1
1-04, 119.3
1-01, 222.3
2-05, 183.6
1-05, 180.3
1-04, 155.8
1-02, 190.5
1-20, 223.5
2-02, 130.9
1-02, 189.1
1-03, 185.1
2-03, 199.2
[204B/204B][100%][=====][204B/s]
```

**Traditional mode, showing  
the content of the file**

```
[root@eosbak02 css]# xrdcp root://eosbak02.ihep.ac.cn//eos/user/chyd/data.txt?css=sort -
1-01, 222.3
1-01, 266.0
1-02, 189.1
1-02, 190.5
1-03, 183.1
1-03, 185.1
1-04, 119.3
1-04, 155.8
1-05, 180.3
1-06, 145.9
1-20, 223.5
2-01, 122.2
2-02, 130.9
2-03, 199.1
2-03, 199.2
2-04, 188.8
2-05, 183.6
[204B/204B][100%][=====][204B/s]
```

**Computational storage mode, showing  
the processed content of the file**



# One example of LHAASO decode

- LHAASO is a large-scale cosmic ray detector array located in southwest China at an altitude of 4410 meters, ~2000 KM away from Beijing
  - It generates 12PB of data annually, which is transferred to Beijing
  - Decode is process to convert raw detector binary data into ROOT file, which needs to read and write much data but consumes very little CPU power
- **Traditional computing mode:** a compute node reads raw data (.dat) from one FST server and then write output data (.root) into another EOS server
- **Computational storage mode:** any XRootd client can launch the decode function on FST server through XRootd Client or cssclient, which read and write data locally

```
bash-4.2$ time aptainer exec /home/chyd/km2adecode.sif /home/km2a/
decode_sort_xrootd.sh /eos/user/c/chyd/km2a/20220701003242.670.dat
/eos/user/c/chyd/km2a/20220701003242.670.dat.root
```

|      |           |   |
|------|-----------|---|
| real | 0m54.875s | <b>Traditional mode running on<br/>compute node</b> |
| user | 0m31.467s |   |
| sys  | 0m8.258s  |   |

```
bash-4.2$ time cssclient -f /eos/user/chyd/km2a/20220701003242.670.dat -c km2a_decode
file '/eos/user/chyd/km2a/20220701003242.670.dat.root' created
```

|      |           |
|------|-----------|
| real | 0m26.347s |
| user | 0m0.015s  |
| sys  | 0m0.015s  |

**Computational storage mode  
launched from a login node**

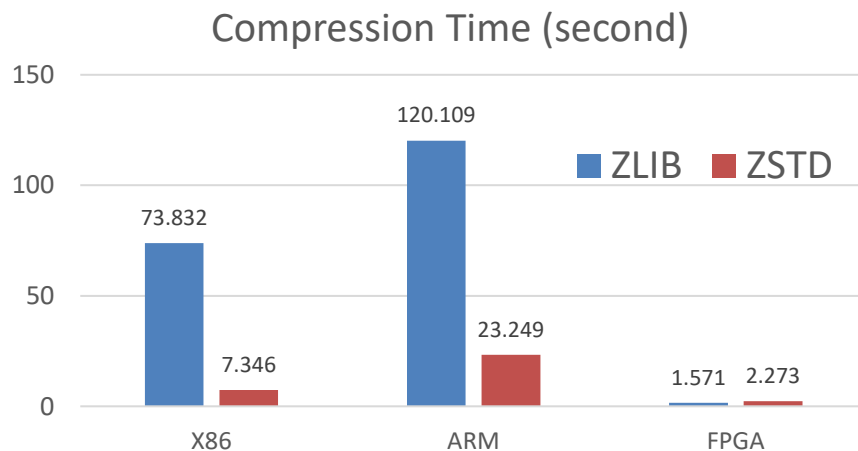
```
bash-4.2$ eos ls -l /eos/user/chyd/km2a
-rw-r--r-- 1 chyd u07 1001554470 Nov 14 21:22 20220701003242.670.dat
-rw-r--r-- 1 chyd u07 446552932 Apr 20 05:26 20220701003242.670.dat.root
```

The same program processes one same file, but the CSSFST only **took one half the time** of traditional mode



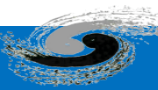
# More cases

- In addition to CPU power in EOS server, some heterogenous computing resources such as GPU, CPU/SoC, FPGA can also be used as computational storage accelerator
- Case 1:** We implemented Intelligent compression for synchrotron radiation source image [chep'21] based on neural network algorithm, but it is very slow. So we use GPU and FPGA to accelerate the process, **more than 340X faster than original method**



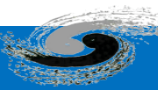
|                  | Original | GPU+FPGA | JPEG2000 | PNG  | ZSTD |
|------------------|----------|----------|----------|------|------|
| Compression rate | 2.08     | 1.78     | 1.26     | 1.43 | 1.13 |
| Time(s)          | 1281.4   | 3.7      | 0.8      | 0.6  | 0.6  |

- Case 2:** We have designed a low-power server that integrates an FPGA and an ARM chip on a single motherboard. First, we ported EOS to the AARCH64 architecture [chep'21] and then developed a compression algorithm based on FPGA, **more than 100X faster than ARM CPU, 50X faster than X86 CPU with ZLIB.**



# Conclusion

- Computational storage is an approach to exploit the computing resources in EOS server
- The plugin CSSFST implemented by IHEP is scalable, configurable and easy to deploy and use
- The heterogenous resources such as GPU, CPU/SoC, FPGA can be used in computational storage mode to accelerate the data processing
- Some known questions such as impossibility to reduce data move in RAIN (erasure code) mode
- We hope we can collaborate with the EOS community to enhance computational storage functionality, making it one of optional features of EOS



Thank you for your attention  
感谢您的聆听！

