Porting the EOS from X86 (Intel) to aarch64 (ARM) architecture

Yaosong Cheng[1,*], *Yujiang* Bi[1,3,**], *Yaodong* Cheng[1,2,3], *Haibo* Li[1,2,3], *Wang* Lu [1,2], *Minxing* Zhang[1,2]

[1]Institute of High Energy Physics, CAS, 100049 Beijing, China

[2]University of Chinese Academy of Sciences, 100049 Beijing, China

[3]Tianfu Cosmic Ray Research Center, 610213 Chengdu, China

Abstract. With the advancement of many large HEP experiments, the amount of data that needs to be processed and stored has increased significantly, so we must upgrade computing resources and improve the performance of storage software. This article discusses porting the EOS software from the x86_64 architecture to the aarch64 architecture, with the aim of finding a more cost-effective storage solution. In the process of porting, the biggest challenge is that many dependent packages do not have aarch64 version and need to be compiled by ourselves, and the assembly part of the software code also needs to be adjusted accordingly. Despite these challenges, we have successfully ported the EOS code to the aarch64. This article discusses the current status and plans for the software port as well as performance testing after porting.

1 Introduction

EOS [1] is the storage technology developed by CERN to support Large Hadron Collider (LHC) activity, and is also the major storage solution adopted by many experiments and projects at Institute of High Energy Physics (IHEP), CAS. It provides a service for storing large amounts of physics data and user files, with a focus on interactive and batch analysis. In our previous work at IHEP, EOS provided efficient and stable services, with peak throughput exceeding 50GB/s, managing more than 18 PB of scientific research data from 2016 to 2020. We expect to have more than 26 PB of data in 2021. In order to cope with the substantial data increase, we need a more cost-effective solution to expand storage space. Therefore, we explored the porting of EOS from Intel x86_64 to ARM aarch64 and tested the related performance.

We chose EOS_4.7.7 for porting, because it was the most stable version when we started porting. In section 2, we briefly introduce the ARM architecture and its role in HEP experiment. Section 3 introduces our porting work and the main difficulties we've encountered. The performance test results and discussions of our EOS are presented in

[*] e-mail: chengys@ihep.ac.cn

[**] e-mail: biyujiang@ihep.ac.cn

section 4. And finally, we summarize the EOS porting work and briefly introduce plans for future work.

2 The ARM architecture and its role in HEP experiment

The ARM is a widely use RISC architecture, and CPUS based on ARM are commonly used in smart phones, laptops, tablets, embedded systems, and so on. Compared with Intel's x86 architecture, it is characterized by small size, low power consumption, and low cost. These features give ARM a certain advantage in the high-performance computing field that requires a lot of power costs. With the introduction of the new generation of ARMv8 architecture, more and more manufacturers have begun to develop ARM-based server-level chips and applied them in scenarios such as distributed storage, big data analysis, and cloud computing. For example, Cavium ThunderX in the United States [2], Fujitsu A64FX in Japan [3], Huawei Kunpeng 920 in China, etc. At the same time, some researchers also discussed the performance differences between the ARMv8 and x86 series from different perspectives.

Laurenzano et al. tested and evaluated the performance of X-Gene [4], which is a 64-bit ARMv8 processor. The results show that the performance of X-Gene is roughly on par with an Intel Atom and that the energy consumption is roughly on par with an Intel Sandy Bridge. Yichao Wang et al. used the thread binding method to compare the performance of the ARM processor under a single Socket with the mainstream commercial processor Intel Xeon [5]. The experimental results show that the software CloverLeaf and TeaLeaf based on the Stencil method have achieved basically the same operating performance on the ARM processor as the Intel XeonHaswell architecture processor. Combining the low power consumption characteristics of ARM processors, ARM server-level processing has the ability to challenge mainstream x86 processors in memory-access restricted applications.

As a classic application scenario of high-performance computing, high-energy physics experiments pay great attention to the software ecology based on the ARM architecture. Many commonly used softwares have ported from x86 to aarch64, such as LHCb stack, LCG software stack and ATLAS software stack. These works are of great significance to the development of software in the field of high energy physics. Marek et al. ported the LCG software stack to aarch64, and then used the program "multicore/mtbb201_parallelHistoFill.C" in ROOT tutorials for testing [6]. The result shows that although single-core ARM machines do not achieve as good results as x86_64 machines, in a multi-core benchmark, as the number of cores increases, the CPU time of X86 and ARM tends to be similar. And the advantage of ARM processor lies mainly in the good cost performance, rather than high performance. In [7] Laura Promberger et al. work on porting LHCb stack from x86 to aarch64 and ppc64le. At the same time, they analyzed and proved the importance of cross-platform support for vectorization. The above results inspired us to port EOS to the aarch64 architecture.

3 Porting to aarch64 (ARM)

Our main work is to port EOS and QuarkDB to the ARMv8 architecture. In the process of porting, we solved three problems. First, because EOS and QuarkDB have many dependency packages, some of them do not have ready-made aarch64 versions, such as *grbc, eos-folly, protobuf, rocksdb, isa-l, isa-l_crypto*, etc. We need to find the correct version of their source code, and then manually compile and generate the corresponding RPM package. Second, some libraries such as *rocksdb* and *folly* will have errors during static linking. These errors may be related to the ARM architecture itself. After changing to dynamic link libraries, no errors will be reported. Therefore, we compiled all libraries with similar problems into dynamic libraries and added them to EOS. Third, we need to port inline assembly code manually. There is a small amount of Intel assembly code related to *crc32* function in EOS, and the assembly instructions need to be replaced with the corresponding ARM assembly instructions. In order to maintain programming consistency, we redefined the assembly instructions for the ARM architecture in the header file *common/crc32c/crc32.h* as following:

```
#ifdef __aarch64__
#define __builtin_ia32_crc32di __builtin_aarch64_crc32cx
#define __builtin_ia32_crc32si __builtin_aarch64_crc32cw
#define __builtin_ia32_crc32hi __builtin_aarch64_crc32ch
#define __builtin_ia32_crc32qi __builtin_aarch64_crc32cb
#endif // GCC_AARCH64_H
```

In addition, we have written an additional compilation switch based on the original software. Its function is to automatically identify the architecture type used by the local machine's chip after turning on the switch. In this way, we do not need to manually select the correct version when installing EOS on a new machine.

After successfully porting EOS to aarch64, we built EOS and generated RPM installation packages. By installing and running EOS on another new server with aarch64 architecture, we proved the success of our porting. In addition, we used two machines to build a distributed storage system for the evaluation of file read and write performance.

4 Performance

The speed of file reading and writing, in practical applications, are important indicators for storage software. Therefore, after installing the aarch64 version of EOS on new arm machines, we tested file reading and writing performance of the local hard disks mounted to EOS and the data transmission speed of the server in the distributed environment. Table 1 contains detailed information about the machines used for benchmarking.

Table 1. Machines used for the benchmark

Items	Parameter
Central Processing Unit	HiSilicon Kunpeng 920-6426 CPU @ 2600MHz 64cores * 2EA
Network Interface Card	25 Gbps
Storage device	SSD drives with SAS interface (SATA 3.2, 6.0 Gb/s)
Operating System	CentOS Linux release 7.6.1810
Operating System Kernel Version	4.14.0-115.10.1.el7a.aarch64
EOS version	4.7.7.aarch64

It is worth noting that our Network interface card support 25 Gbps, so the theoretical data transmission speed is 3.125 giga byte per second. We expect the actual data transmission speed is within 10% of the theoretical value.

4.1 Performance of SSD

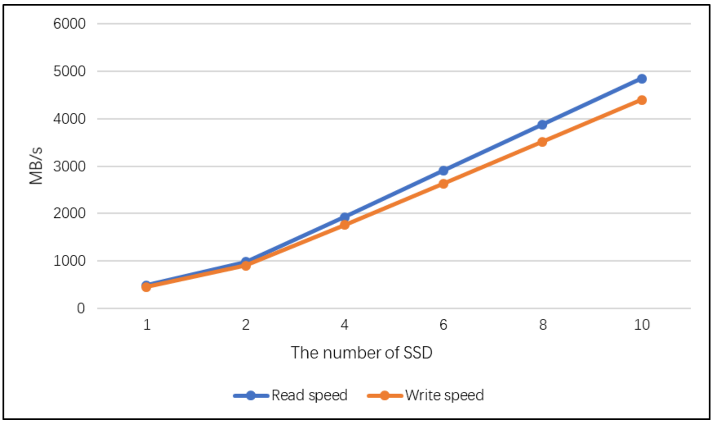

Fig. 1. Average read or write speed of multiple SSDs

We use the “*dd*” command to test the disks mounted to EOS. Some examples are as following:

Write test command:

dd if=/dev/zero of=$dirn/localtest/f$i bs=1024M count=10 ***oflag=direct***

Read test command:

dd of=/dev/zero if=$dirn/localtest/f$i bs=1024M count=10 ***iflag=direct***

In order to make the results more accurate, we set the arguments “*oflag*” and “*iflag*” to “*direct*”, which means that we turn off system cache during the tests. Then we test the average speed when reading or writing multiple SSDs at the same time. The test results are shown in Figure 1. The results show that the average read speed of a single SSD is 511 MB/s, and the average write speed is 421 MB/s.

It can be seen that with the increase of the number of SSDs, the average reading and writing speed of the system are also increasing, and the relationship between the two is almost linear. We can better evaluate the performance of data transmission of EOS by testing the performance of SSD.

4.2 Performance of data transmission

For a complete EOS system, metadata entries for files and directories are cached in the MGM namespace server. The data storage service FST provides a plug-in infrastructure for file layouts and storage protocols. The main layouts are replication and erasure encoding of files. The replica mode means how many copies of the file are stored in EOS.

We tested the data transmission performance of EOS with different replica layout configurations. Firstly, we evaluated the performance of single replica mode with 2 FSTs, and the development is shown in Fig. 2.

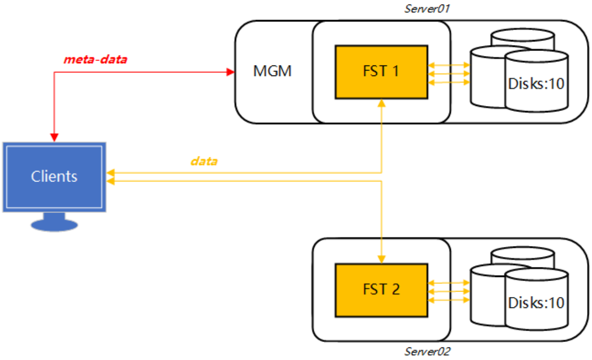

Fig. 2. The development of 2 FSTs in single replica mode

We use the XRootD client "*xrdcp*" to test the writing performance, and the reading performance is tested by using a script "*xrdread*" written by ourselves, the main part of which is to read a file in the storage system through the XRootD protocol. We take the total number of network received and sent data per second provided by the "*dstat*" command as the performance indicator. To test the maximum data transfer speed, we run multiple writing or reading instances simultaneously on each client. Specifically, we set the file size involved in "*xrdcp*" to 16GB, and executed a total of 180 identical commands on all clients, so that the amount of data sent to the EOS server in real time has exceeded the limit of the network bandwidth. For the "*xrdread*" script, we made the same configuration. Some examples are as following:

Write test command:

```
xrdcp /localdata/write/f$i   root://eos.mgm.test.com//eos/data/
```

Read test command:

```
./xrdread root://eos.mgm.test.com///eos/data/f$i   /localdata/read/
```

Based on the above configuration, we test the data transfer speed. the results show that the maximum total receive data speed of EOS is 5.868 GB/s and the maximum send data speed is 5.620 GB/s.

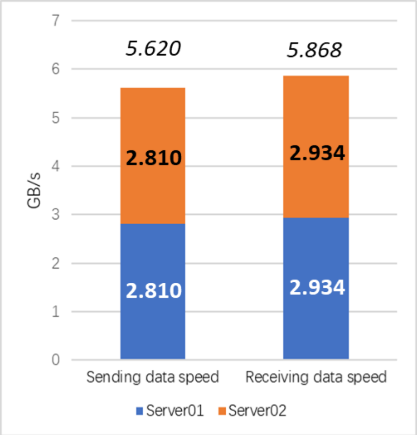

Fig. 3. Maximum transferring data speed

The data transmission speed of each server is in line with our expectation, that is the difference between the theoretical value and the actual value is within 10%. The reason is that EOS will perform verification when sending or receiving data. In addition, sending data is lower than receiving data in that the sent data is read by the SSD, and the received data is written into the cache first. The performance difference between the cache and the SSD causes this phenomenon.

Then, we tested the data transmission performance difference between 2 FSTs and 6 FSTs in two-replica mode. The system developments are shown in Figure 4.

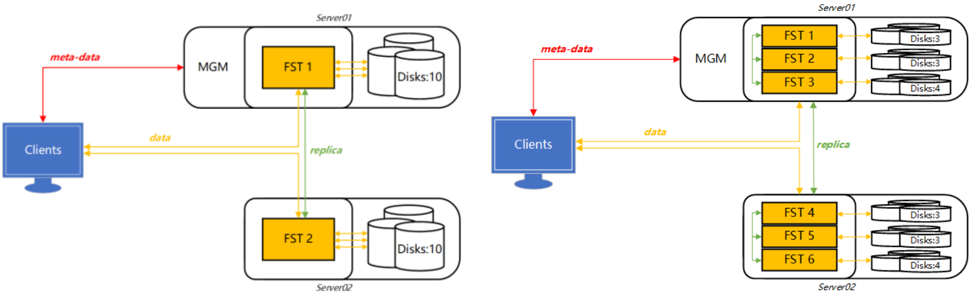

Fig. 4. The development of 2 FSTs and 6 FSTs in two-replica mode

When there are 2 FSTs in two-replica mode, one FST is deployed on each server. For the second case, we set up 3 FSTs on each server. The test results are shown in Figure 5 and Figure 6.

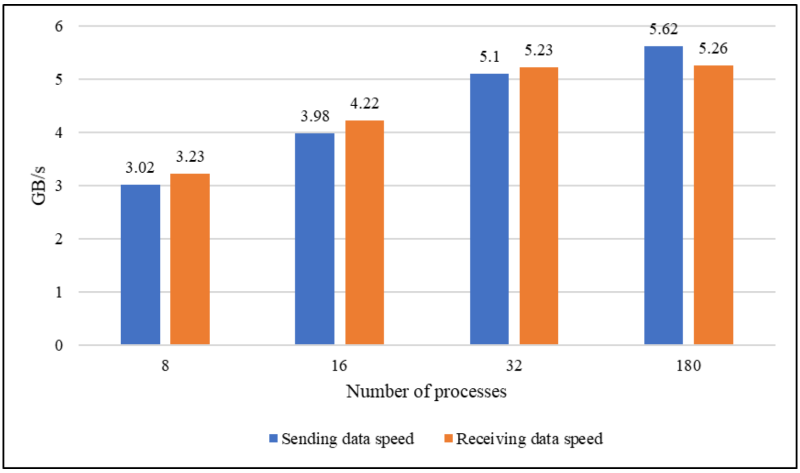

Fig. 5. maximum read or write speed of 2 FSTs in two replica mode

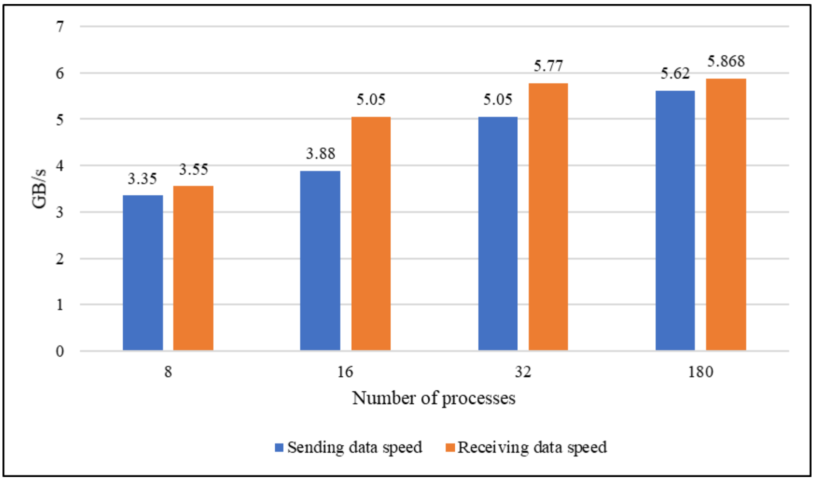

Fig. 6. maximum read or write speed of 6 FSTs in two replica mode

We can conclude from the test results that when the network bandwidth does not reach the maximum limit, the sending data speed is always lower than the receiving data speed. And the reason is the same as that of single replica. It should be noted that when the network bandwidth is full, the speed of sending data of 2 FSTs is greater than the speed of receiving data. This is because when FST receives data, it needs to send the data to another FST to complete the backup at the same time which occupies part of the bandwidth. When there are 6 FSTs, the speed of sending data is lower than the speed of receiving data. This is because there are multiple FSTs on the same machine, and the backup may not need to go through the network.

These tests show that the main functions of EOS can run normally on aarch64 for the first time, and the performance is in line with our expectations, that is, the actual data transmission speed is within 10% of the theoretical speed.

5 Future work

Our future goal is to have our arm branch of EOS been merged into EOS upstream in the CERN, and to organize and maintain the EOS dependent libraries based on the aarch64 architecture, and provides unofficial support for these libraries on ARM platform., By this more people or organizations can benefit from EOS on ARM platform and provide us with valuable feedback, which may promote the progress and development of EOS in turn. Finally, we also plan to conduct a detailed analysis of the cost-effectiveness of different architectures, which will help us choose more appropriate hardware resources.

6 Conclusion

Our work has broadened the application scenarios of the EOS distributed storage system, so that it can work on ARM servers as well as x86 machines. This provides us with more options when purchasing hardwares. During the whole research process, firstly we tested and proved the availability of EOS based on aarch64machines. Then, through the test of single and double copy mode, the EOS performance on aarch64 machines has reached our expectation. We have confidence that following work based on this will benefit us in providing better services for various experiments.

7 Acknowledgments

This work is supported by the National Natural Science Foundation of China (No.12075268).

This work is partly supported by the Science and Technology Innovation Project of Institute of High Energy Physics, Chinese Academy of Sciences (No.E15451U2)

References

1. Peters, A. Joachim, E.A. Sindrilaru, and G. Addel, EOS as the present and future solution for data storage at CERN. JPCS, **664** (2015)
2. S.M. Smith, J. Price, T. Deakin, A. Poenaru, A performance analysis of the first generation of HPC-optimized Arm processors. CCPE, **31** (2019)
3. T. Yoshida, Fujitsu high performance CPU for the Post-K Computer. HCS (2018)
4. M.A. Laurenzano, A. Tiwari, A. Cauble-Chantrenne, A. Jundt, L. Carrington, Characterization and bottleneck analysis of a 64-bit ARMv8 platform. ISPASS, **36**, (2016)
5. M. Marek, G. Ganis, P. Mato, I. Razumov, Cern students note, Porting the LCG software stack to the ARM architecture. September 2019, https://cds.cern.ch/record/2690233
6. L. Promberger, M. Clemencic, B. Couturier, A.B. Iartza, N. Neufeld, Porting the LHCb Stack from x86 (Intel) to aarch64 (ARM) and ppc64le (PowerPC). EPJ, **214**, 05016 (2019)