NUMA-Aware Thread and Resource Scheduling for Terabit Data Movement

Taeuk Kim[1], Awais Khan[1], Youngjae Kim[1], Sungyong Park[1], Scott Atchley[2]

[1]Sogang University
Seoul, Republic of Korea

[2]Oak Ridge National Laboratory
Oak Ridge, TN USA 37831

ABSTRACT

The technology advancement has introduced several storage and network level changes to existing data transfer tools for efficient data sharing among different storage centers. LADS, an end-to-end layout-aware data transfer tool optimized for terabit network atop of parallel file system cannot utilize the NUMA equipped nodes, as it cannot prevent remote NUMA node memory accesses. We cater this issue by implementing memory-aware thread and resource scheduling at memory level in existing LADS framework.

1. INTRODUCTION

With the continuous expansion of big data, there is a growing demand for data sharing and collaboration in scientific, research and computing facilities. This data sharing and collaboration requires a tremendous amount of data transfer between geographically dispersed data centers. The Brookhaven National Lab (BNL) cooperates with European Large Hadron Collider (LHC) in the ATLAS experiment in which more than 3,000 scientists participate in producing petabytes of simulation and analytical data facilitating collaboration. Given this environment, the optimization of end-to-end data transfer tools between data centers is also important.

The three major contentions can occur in end-to-end data transfer between data centers; (i) network, (ii) storage, and (iii) memory. In an environment where each organization is connected to a terabits network, such as ESNet [1] the network is not considered as a candidate contention [2]. The storage bottleneck occurs due to I/O contention of threads in the object storage servers (OSSs) and targets (OSTs), when thread count exceeds the service rate of the OSS or when multiple threads access the same OST. LADS [2], a high-speed end-to-end data transfer tool between data centers, minimizes this I/O contention by being aware of the layout of data chunks and scheduling threads based on it. However, in the NUMA node environment, when a thread accesses memory on a remote NUMA node, memory contention occurs. If the buffer that LADS uses to transfer data is allocated in another NUMA node, remote NUMA node access is required. But currently, LADS is unaware of these NUMA architectures and cannot address these problems. We solve the memory bottleneck problems by using Memory-aware Thread Scheduling (MTS) in the LADS framework.

2. DESIGN AND IMPLEMENTATION

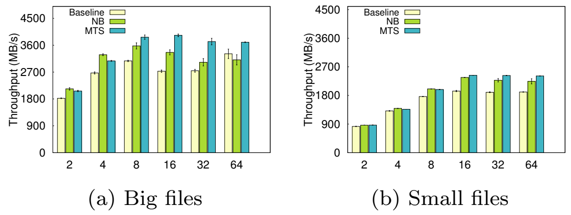
We address the remote memory access problem by providing two mechanisms. First, we propose an RMA buffer partition technique for each CPU socket. Second, we introduce *Memory-aware Thread Scheduling* (MTS) to solve the remote memory access problem. In current LADS architecture, three types of threads participate in data transmission; Master thread (MT), Communication thread (CT), I/O thread. In MTS, MT schedules I/O threads by considering data chunk's layout in PFS. CT keeps the RDMA connection between source and sink side. I/O thread loads data chunks to RMA buffer from PFS in source side and stores data chunks to PFS from RMA buffer on sink side. Also, MTS divides the RMA buffer per CPU socket and makes each CPU socket have a connection with opposite side, so MT, CT and I/O thread manage connection per every socket.

Socket-based MTS (SMTS): Distributing RMA buffer into all CPU sockets cannot perfectly prevent I/O thread from accessing to remote socket's RMA buffer though it can reduce the probability of remote socket's memory access. To deal with such cases, we designed SMTS which schedules all I/O threads to access local socket's RMA buffer.

NUMA-based MTS (NMTS): The recent CPU architecture often has more than one NUMA node per CPU socket. In this case, it is needed to balance all threads load into all NUMA nodes in socket. Also, MT and CT interact with each other using their own work queues, so they should be aligned to same NUMA node. For this purpose, we propose NMTS that ensures the load balancing of threads to NUMA nodes and schedules highly related threads to same NUMA node's cores.

3. EVALUATION

To evaluate the proposed ideas, we conducted preliminary experiments using a memory-based file system installed on each source and sink host. In our experiments, the overhead of PFS was excluded. We mount NFS on memory file system (tmpfs) and used 8 big and 24,000 small files workload, with each total size of 24GB. In Figure 1, baseline is throughput of LADS in NUMA architecture, and NB is LADS with NUMA binding I/O threads in node with RMA buffer, and MTS is the memory-level optimization with both SMTS and NMTS. When the I/O threads count is low, NUMA binding shows the best performance, in average 23% as compared to baseline. But when the thread count gets bigger, the memory-level optimization outperforms in both workloads, from 21.7% to 44% in comparison to baseline.



(a) Big files (b) Small files

Figure 1: Performance comparison for MTS with baseline. The x-axis shows the I/O thread count.

4. REFERENCES

[1] ESnet. Energy Sciences Network (ESnet).

[2] Youngjae Kim, Scott Atchley, Geoffroy R. Vallée, and Galen M. Shipman. LADS: Optimizing Data Transfers Using Layout-aware Data Scheduling. In *Proceedings of the 13th USENIX Conference on File and Storage Technologies*, FAST'15, 2015.

This work was supported by Institute for Information & communications Technology Promotion (IITP) grant funded by the Korea government (MSIT) (No.2015-0-00590).