The Multicore-Aware Data Transfer Middleware Project

L. Zhang, P. DeMar, W. Wu
Fermilab
{liangz, demar, wenji}@fnal.gov

Tan Li*, Y. Ren*, S. Jin*, D. Yu+
*Stony Brook University, +BNL
{tan.li, yufei.ren, shujin}@stonybrook.edu , dtyu@bnl.gov

***Abstract*— Existing data movement tools are still bound by major inefficiencies when running on multicore systems. To address these inefficiencies and limitations, DOE's Advanced Scientific Computing Research (ASCR) office has funded Fermilab and Brookhaven National Laboratory to collaboratively work on the Multicore-Aware Data Transfer Middleware (MDTM) project. MDTM aims to accelerate data movement toolkits on multicore systems. A prototype version of MDTM is currently undergoing evaluation and enhancement.**

Index Terms— **Multicore; network; NUMA**

I. Problem Space and Significance

Multicore and manycore have become the norm for high-performance computing. These new architectures provide advanced features that can be exploited to design and implement a new generation of high-performance data movement tools. To date, numerous efforts have been made to exploit multicore parallelism in order to speed up data transfer performance: *At the application level*, various data movement tools have been developed, such as TCP-based GridFTP [1] and BBCP [2]. Parallel data transfer technologies are widely used in bulk data movement and provide significant improvement in aggregate data transfer throughput. These data transfer tools typically employ a multi-threaded architecture. Multiple threads are utilized, with each thread handling one or multiple flows, depending on the runtime environments. *At the OS level,* major OSes (e.g., Windows, and Linux) have been redesigned and parallelized to utilize multicore platforms more effectively [3]. *At the hardware level,* new multi-queue NIC technologies have been introduced, and the use of NUMA (non-uniform memory access) systems is on the rise. Due to the scalability advantage of NUMA architecture over UMA (uniform memory access) architecture, high-performance data transfer systems are typically NUMA based and feature several nodes distributed across the system.

Although these parallelization efforts have improved data transfer performance significantly, existing data movement tools are still bound by major inefficiencies when running on multicore systems. While there are numerous reasons for these inefficiencies, the inefficiencies fall into two general problem areas: (1) existing data transfer tools are unable to fully and efficiently exploit multicore hardware under the default OS support, especially on NUMA systems; and (2) the disconnect between software and multicore hardware renders network I/O processing on multicore systems inefficient [4]. These inefficiencies are fundamental and common problems that data movement tools will inevitably encounter when running on multicore systems. Ultimately, these inefficiencies result in performance bottlenecks on the end systems. In turn, such end system performance bottlenecks also impede the effective use of advanced networks. The U.S. Department of Energy (DOE) has a strategic goal of deploying terabit networks in support of distributed extreme-scale data movement. Existing backbone networks are already built upon ultra-scalable 100-Gigabit line rate technologies. Resolving performance issues within end systems is now becoming the critical element within the end-to-end loop of distributed extreme-scale data movement. Terabit networks need terabit-capable end systems to efficiently move data on and off of the network.

II. The MDTM Solution

To address these inefficiencies and limitations, DOE's Advanced Scientific Computing Research (ASCR) office has funded Fermilab and Brookhaven National Laboratory to collaboratively work on the Multicore-Aware Data Transfer Middleware (MDTM) project. MDTM aims to accelerate data movement toolkits on multicore systems. Essentially, MDTM consists of two major components (Figure 1):

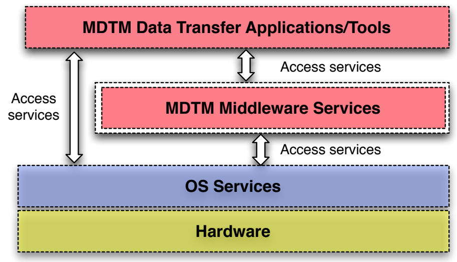

Figure 1: MDTM Architecture

- MDTM data transfer applications (client or server): An MDTM application performs data transfer tasks. It adopts an I/O-centric architecture that uses dedicated threads to perform network and disk I/O operations. In addition, it uses MDTM middleware services to fully utilize the underlying multicore system.
- MDTM middleware service: The middleware will hardness multicore parallelism to scale data movement toolkits on host systems. It provides generic services and functions that can be called by an MDTM

application to ensure efficient resource utilization on the host systems. MDTM middleware schedules and assign system resources based on the needs and requirements of data transfer applications (i.e., data transfer-centric scheduling). It also takes into account other factors, including NUMA topology, I/O locality, and QoS.

In the MDTM project, we plan to achieve several goals: (a) develop new software tools for extreme-scale data movement; (b) investigate, design, and implement generic middleware mechanisms to enable extreme-scale data movement tools to exploit multicore hardware fully and efficiently, particularly on NUMA-based platforms; and (c) integrate the solutions developed in (a)-(b) to provide a unified solution to support extreme-scale data transfer toolkits. To the best of our knowledge, the scope of this work is unique.

III. Current State, Initial Results, and Future Directions

The development and implementation of this project has proceeded on schedule, with both the middleware and the application teams have achieving their year-1 milestone goals. Specifically, the application team implemented key modules such as thread/flow management, request preprocessing, and various data access/transmission methods. The middleware team has implemented key modules such as multicore system profiling, topology-based resource scheduling, and interrupt affinity for network I/O. With this progress, the teams have been able to successfully integrate the middleware and application. The teams are now conducting initial function and performance tests.

In one set of initial tests, we evaluated and compared the performance of MdtmAPP and GridFTP. In our test configuration, we used one source server with 2 NUMA nodes, 32 cores, and with SSDs to store 8 large files. We used one destination server with 4 NUMA nodes, 64 cores, and software RAIDs to store the transferred files. Maximum network bandwidth between the two servers is 40Gbps. Our MdtmApp used 8 parallel groups of storage/network threads to handle the file transfers. For fair performance comparison, we ran 8 parallel and independent instances of GridFTP. Performance results were captured using the Ganglia tools, and the pictures below show the bandwidth throughput (Figure 2), and both source/destination CPU consumption (Figures 3a & 3b).

The near term focus of MDTM development will be on continued evaluation and enhancement of the software on 40GE-attached host systems, with particular emphasis in long distance (high RTT), high volume data transfer environments. Over the intermediate term, the project team intends to expand the evaluation of the product across a wider spectrum of multicore/manycore platform types. A key longer term objective is to include an external MDTM resource scheduling capability, enabling end system CPU resources to become a reservable resource for application-driven end-to-end path reservations.

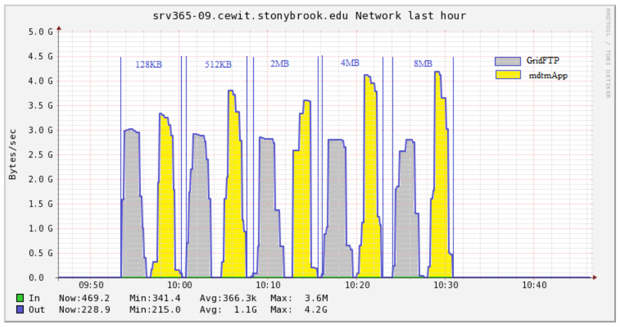

Figure 2: Throughput

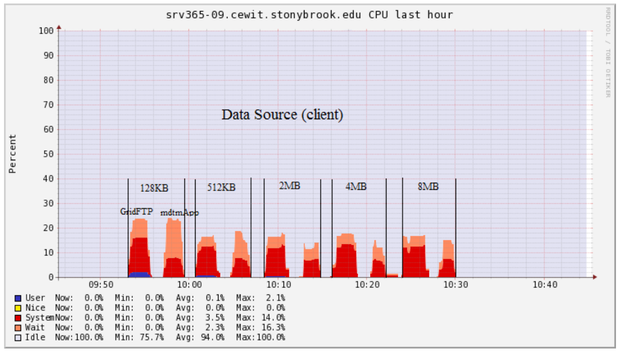

Figure 3a: CPU Utilization (source)

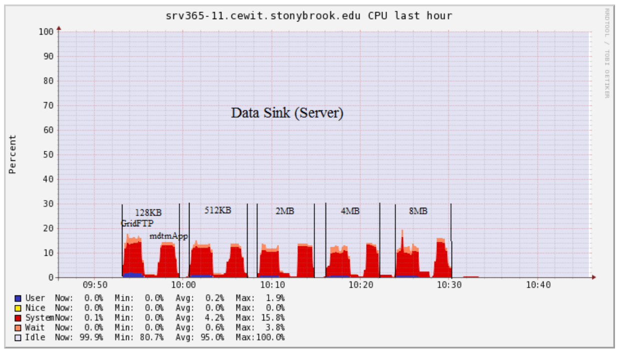

Figure 3b: CPU Utilization (sink)

References:

[1] W. Allcock, J. Bester, J. Bresnahan, A. Chervenak, L. Liming, and S. Tuecke, “GridFTP: Protocol Extension to FTP for the Grid,” Grid Forum Internet-Draft, Mar. 2001.

[2] BBCP, http://www.slac.stanford.edu/~abh/bbcp/

[3] P. Willmann et al., “An Evaluation of Network Stack Parallelization Strategies in Modern Operating Systems,” In Proc. USENIX Annual Technical Conference, pp. 91–96, 2006.

[4] W. Wu, P. DeMar, M. Crawford, “A Transport-Friendly NIC for Multicore/Multiprocessor System,” IEEE Transactions on Parallel & Distributed Systems, Volume 23, Issue 4, pp. 607-615, 2012.