Multicore-aware Data Transfer Middleware (MDTM)

L. Zhang P. DeMar, W. Wu (PI)
Fermilab

T. Li, Y. Ren, S. Jin, D. Yu (Co-PI)
BNL

Enhancing Big Data Science Through Optimization of Network I/O on Multicore Systems

Problem Space

Multicore/manycore has become the norm for high-performance computing. However, existing data movement tools are still limited by major inefficiencies when run on multicore systems:

- **Existing data transfer tools can't fully exploit multicore hardware, especially on NUMA systems**
- **Disconnect between software and multicore hardware renders network I/O processing inefficient**
- **Performance gaps between disk and network devices difficult to narrow on NUMA systems**
- **Data transfer tools receive only best-effort handling for their process threads**

These inefficiencies will ultimately result in performance bottlenecks on end systems. Such bottlenecks also impede the effective use of advanced high-bandwidth networks

Our Solution: The Multicore-aware Data Transfer Middleware (MDTM) Project:

- **Collaborative effort by Fermilab and Brookhaven National Laboratory**
- **Funded by DOE's Office of Advanced Scientific Computing Research (ASCR)**

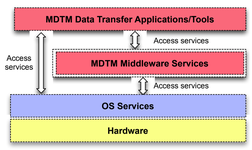
MDTM aims to accelerate data movement toolkits on multicore systems

MDTM Architecture

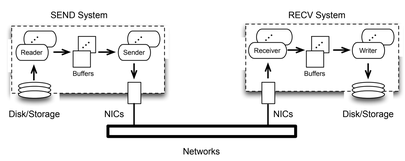
MDTM consists of two components:

- **MDTM data transfer applications (client or server)** - adopts an I/O-centric architecture that uses dedicated threads to perform network and disk I/O operations
- **MDTM middleware service** - harness multicore parallelism to scale data movement toolkits on host systems

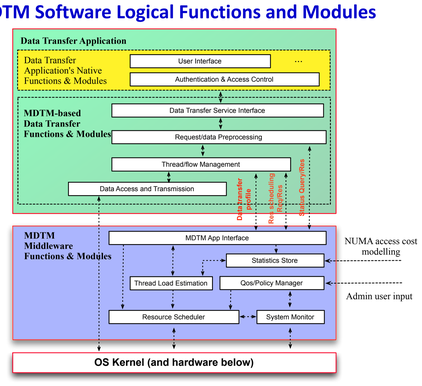
MDTM Architecture



MDTM Data Transfer Model



MDTM Software Logical Functions and Modules



I/O-centric architecture
Parallel data transfer

Data flow-centric scheduling
NUMA-awareness scheduling
I/O locality optimization
Maximizing parallelism

- **Data transfer application's native functions**
- **Data transfer service interface**
- **Request/data preprocessing**
- **Thread/flow management**
- **Data access and transmission**
- **App interface**

- **System profiling and monitoring**
- **Thread load estimation**
- **Resource scheduler**
- **QoS/Policy manager**

How does MDTM work?

An MDTM application spawns three types of threads:

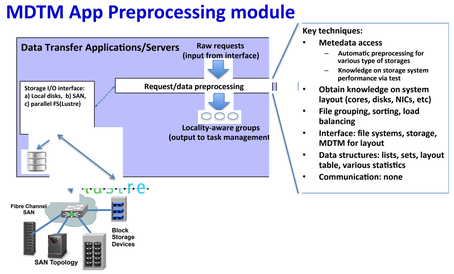
- **Management threads to handle user requests and management-related functions**
- **Dedicated disk/storage I/O threads to read/write from/to disks/storages**
- **Dedicated network I/O threads to send/receive data**

An MDTM data transfer application accesses MDTM middleware services explicitly via APIs.

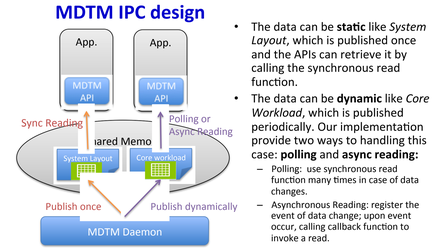
In operation, an MDTM middleware daemon will be launched. It will support two types of services

- **Query service allows MDTM APP to access system configuration and status**
- **Scheduling service assigns system resources based on requirements of data transfer application(s)**

MDTM App Preprocessing module

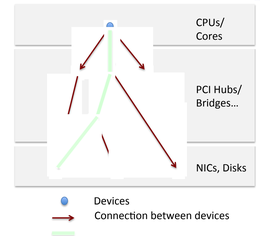


MDTM IPC design



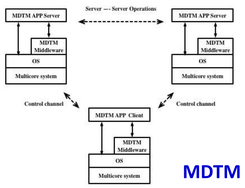
- The data can be **static** like *System Layout*, which is published once and the APIs can retrieve it by calling the synchronous read function.
- The data can be **dynamic** like *Core Workload*, which is published periodically. Our implementation provide two ways to handling this case: **polling** and **async reading:**
  - Polling: use synchronous read function many times in case of data changes.
  - Asynchronous Reading: register the event of data change; upon event occur, calling callback function to invoke a read.

MDTM Middleware Scheduling



MDTM deployment scenarios:

MDTM Client – Server data transfer



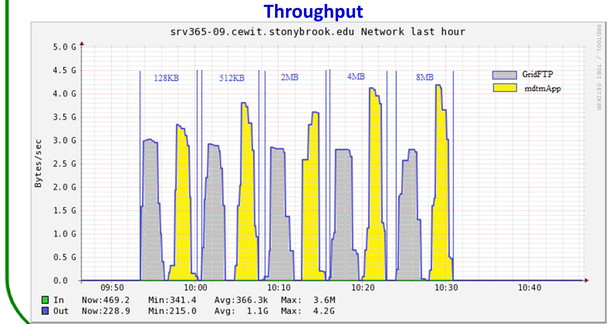
MDTM thirty party data transfer

MDTM server works with a standard FTP client

Initial Results (Year-1 prototype)

Test environment: **2 HPC servers connected at 40GE**
MdtmApp server – 8 parallel groups of network/storage threats
GridFTP server - 8 parallel, independent instances of GridFTP

Throughput



FTP CPU Consumption