The Multicore-aware Data Transfer Middleware (MDTM) Project

Wenji Wu (FNAL), Dantong Yu (BNL)
wenji@fnal.gov, dtyu@bnl.gov

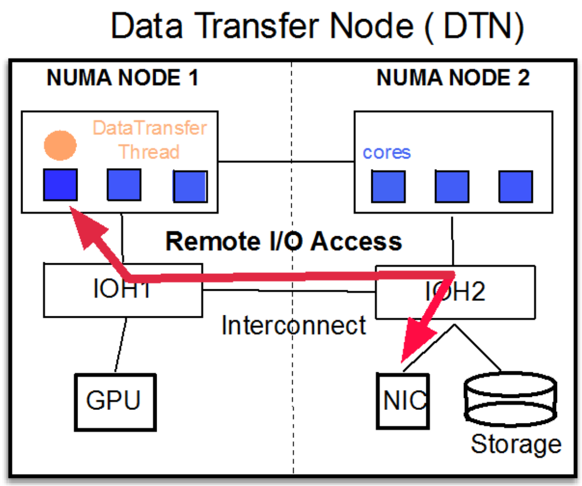
ASCR Next-Generation Network for Science (NGNS)
Principal Investigators' (PI) Meeting
Rockville, MD September 16-17, 2014

Problem Space

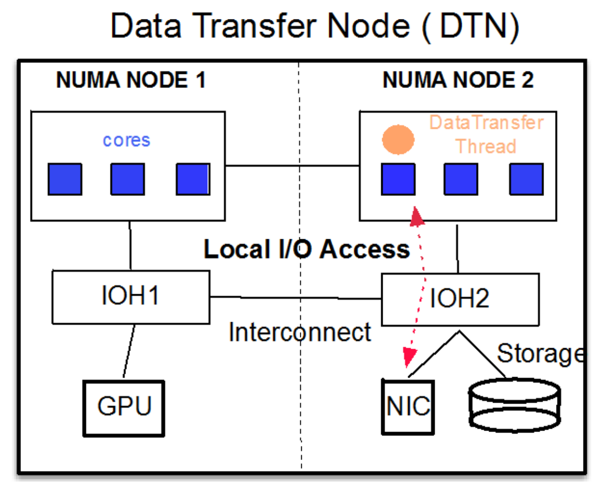
- Multicore/manycore has become the norm for high-performance computing.
- Existing data movement tools are still limited by major inefficiencies when run on multicore systems
  - Existing data transfer tools can't fully exploit multicore hardware, especially on NUMA systems
  - Disconnect between software and multicore hardware renders I/O processing inefficient
  - Performance gap between disk and network devices difficult to narrow on NUMA systems

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

A simple inefficiency case …



Scheduling without I/O locality



Scheduling with I/O locality

General-purpose OSes have only limited support for I/O locality!

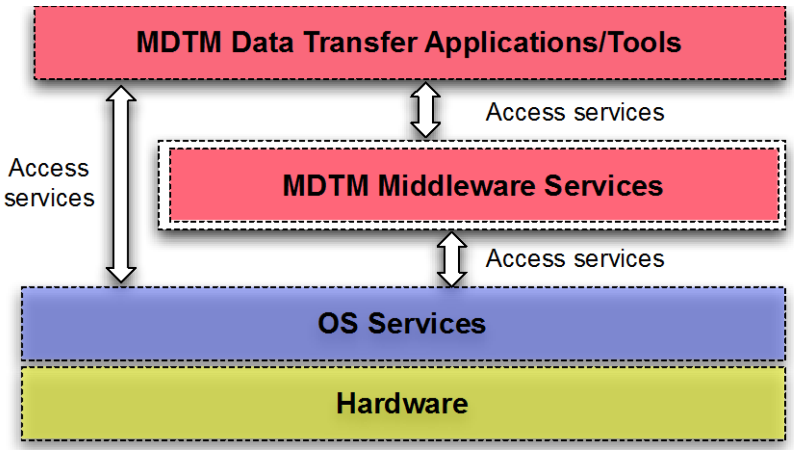
How can we improve?

Our solution

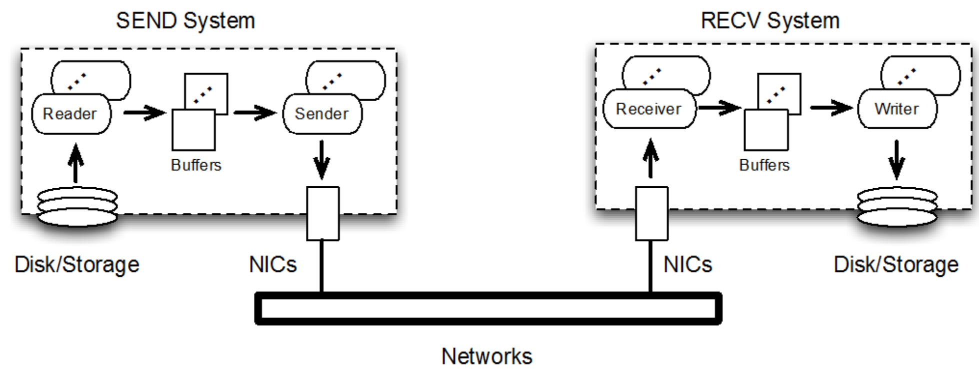
- **The Multicore-aware Data Transfer Middleware (MDTM) Project**
  - Collaborative effort by Fermilab and Brookhaven National Laboratory
  - Funded by DOE's Office of Advanced Scientific Computing Research (ASCR)

MDTM aims to accelerate data movement toolkits on multicore systems

MDTM Architecture



MDTM Architecture



MDTM Data Transfer Model

MDTM consists of two components:

- **MDTM data transfer application (BNL)**
  - Adopts an I/O-centric architecture that uses dedicated threads to perform network and disk I/O operations
- **MDTM middleware services (FNAL)**
  - Harness multicore parallelism to scale data movement toolkits on host systems

MDTM Architecture (cont.)

I/O-Centric architecture

Parallel data transfer

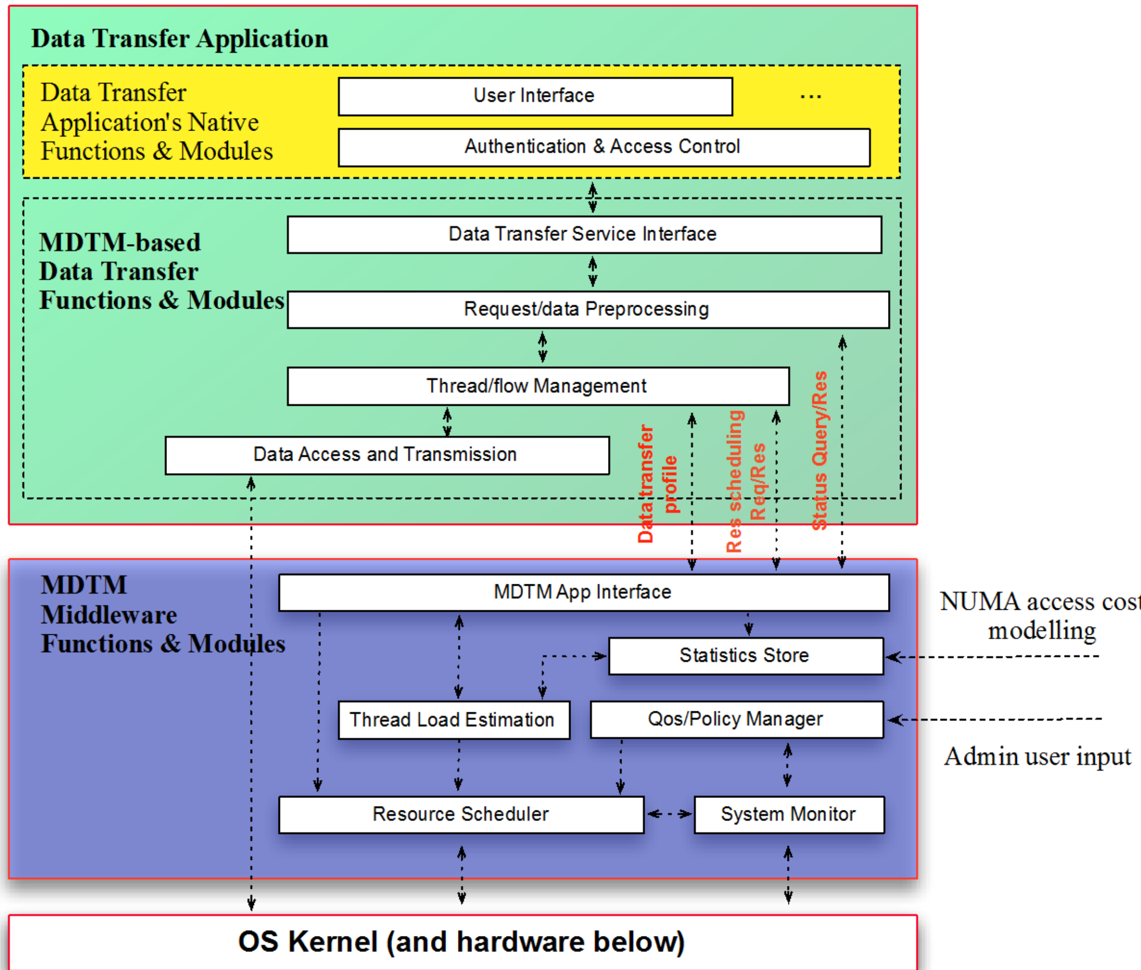
Data layout preprocessing

Data flow-centric scheduling

NUMA-awareness scheduling

I/O locality optimization

Maximizing parallelism



MDTM Software Logical Functions and Modules

Office of Science U.S. DEPARTMENT OF ENERGY   Fermilab   BROOKHAVEN NATIONAL LABORATORY

How does MDTM works?

A MDTM application spawns three types of threads

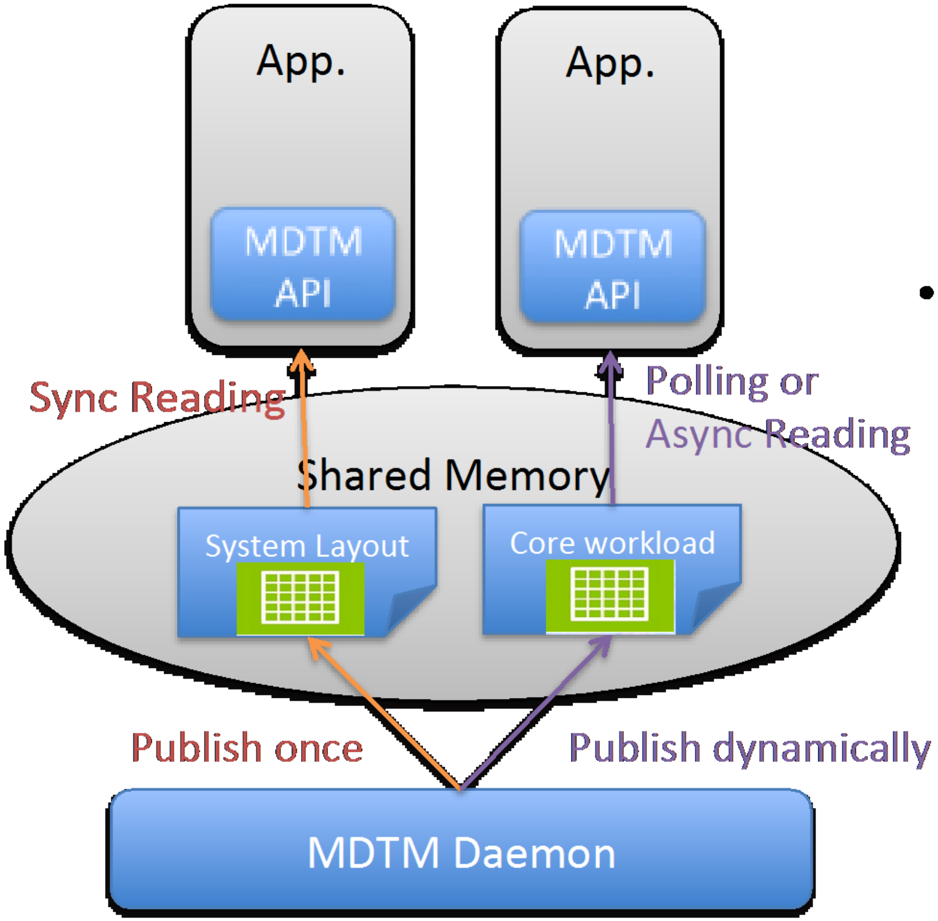
- Management threads to handle user requests and management-related functions
- Dedicated disk/storage I/O threads to read/write from/to disks/storages
- Dedicated network I/O threads to send/receive data

A 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 allow MDTM APP to access system configuration and status
- Scheduling service assigns system resources based on requirements of data transfer applications

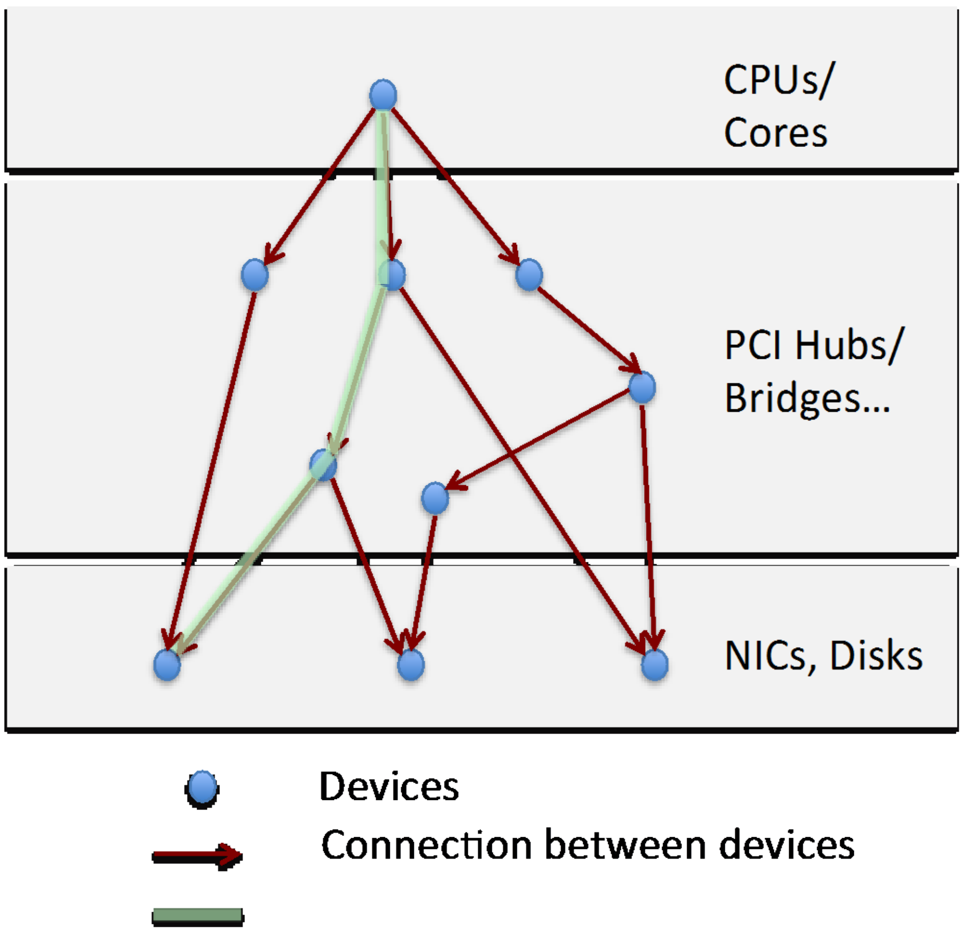
How does MDTM work? (Interaction)



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.

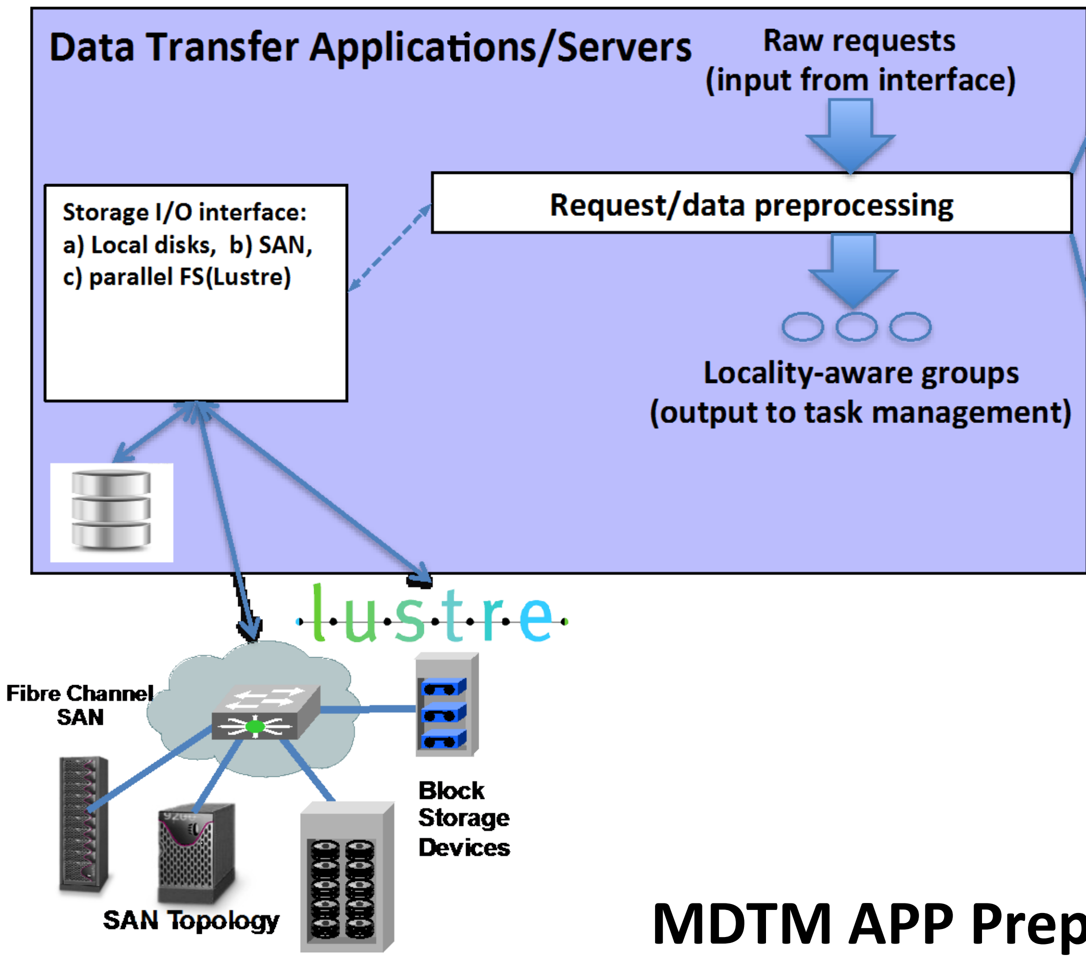
How does MDTM work? (Middleware)



- Each connection associated with a cost value which reflects scheduling factors like distance, traffic throughput and etc.
- Each node contains a cost table to its neighbors
- Applying Dijkstra's Algorithm to find the lowest cost path from CPU node to the NIC/Disk node in question
- pick up the core associated to the lowest cost path
- Pros and Cons
  more extensive system picture; scalable; dynamic; more complicated data structure

MDTM Middleware Scheduling

How does MDTM work? (MDTMApp)

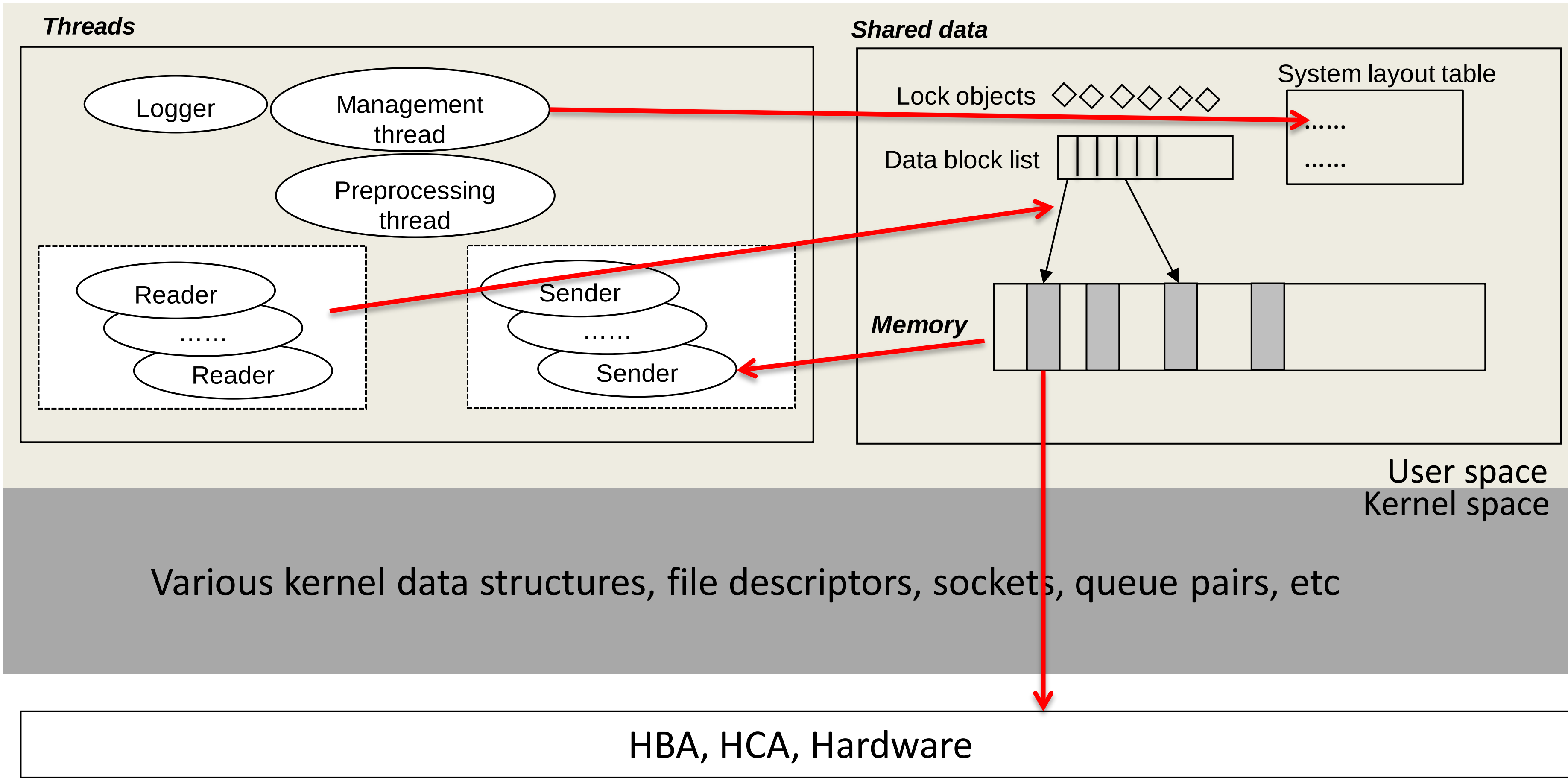


Key techniques:

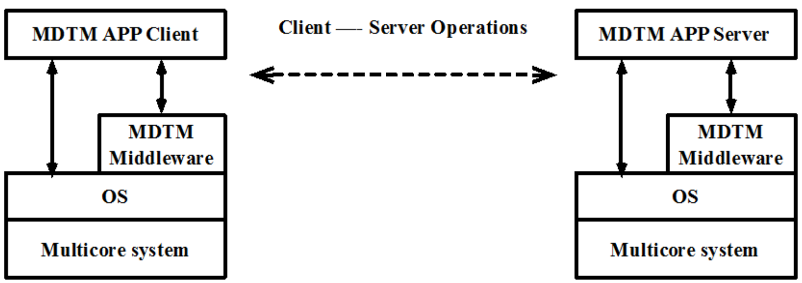
- Metedata access
  - Automatic preprocessing for various type of storages
  - Knowledge on storage system performance via test
- Obtain knowledge on system layout (cores, disks, NICs, etc)
- File grouping, sorting, load balancing
- Interface: file systems, storage, MDTM for layout
- Data structures: lists, sets, layout table, various statistics
- Communication: none

MDTM APP Preprocessing module

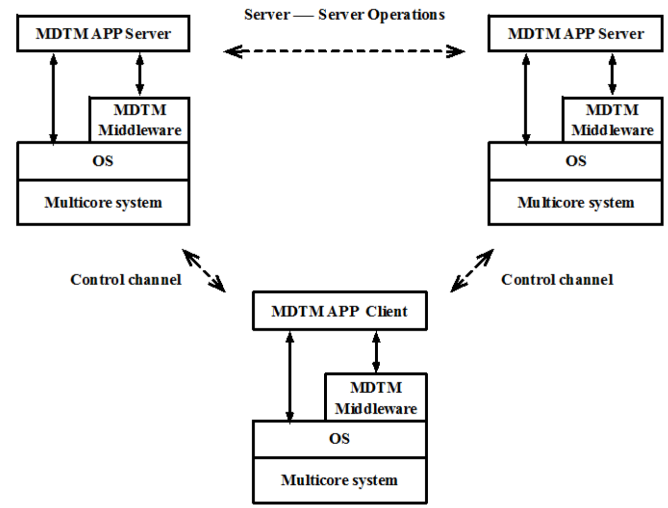
Data Access/Transmission Logic (Application memory layout)



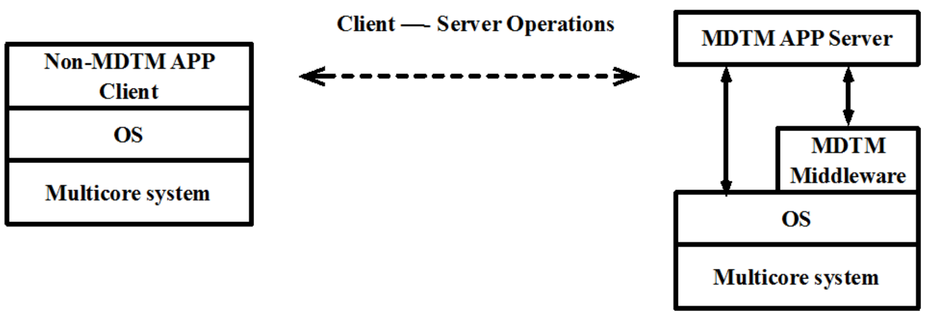
MDTM deployment



A. MDTM client – server data transfer



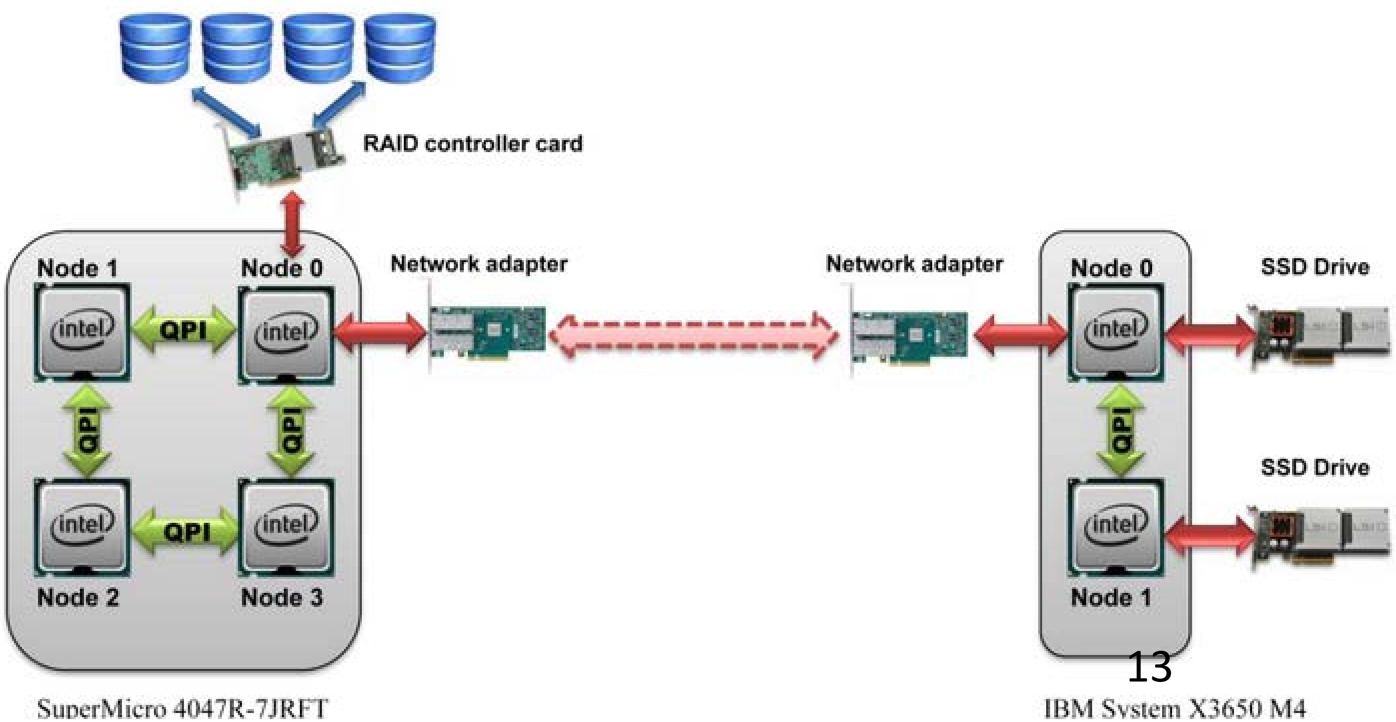
B. MDTM third party data transfer



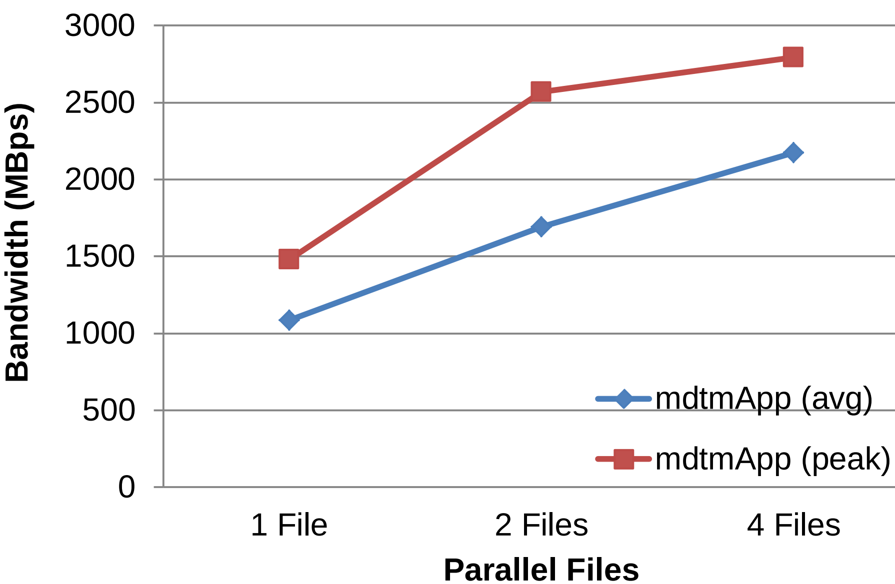
c. An MDTM server works with a standard FTP client

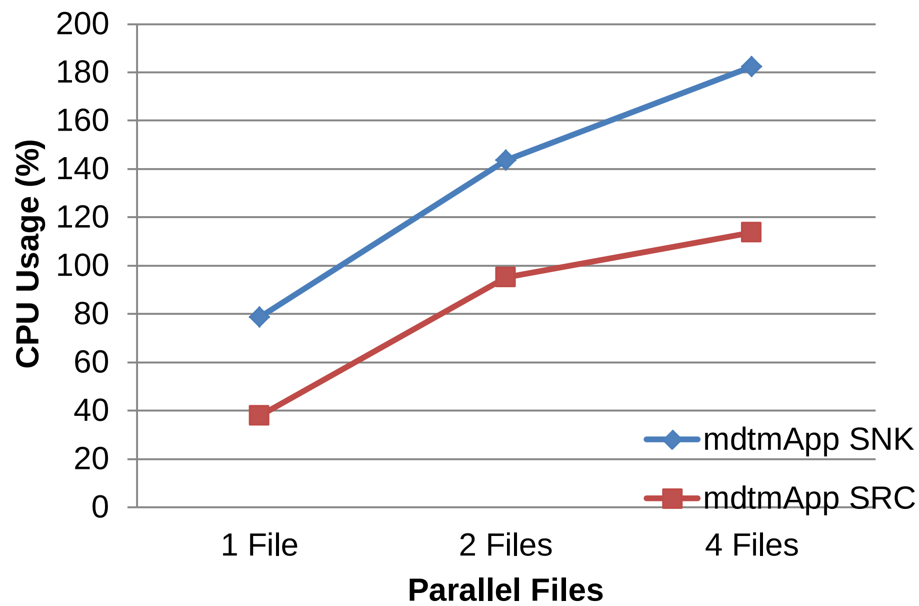
Initial Tests – Large Files

- Parallel streams from 2 SSDs at source host to 4 RAIDs at destination host
- Techniques used: locality-aware binding, grouping, parallel I/O of disk and network, sequential writing



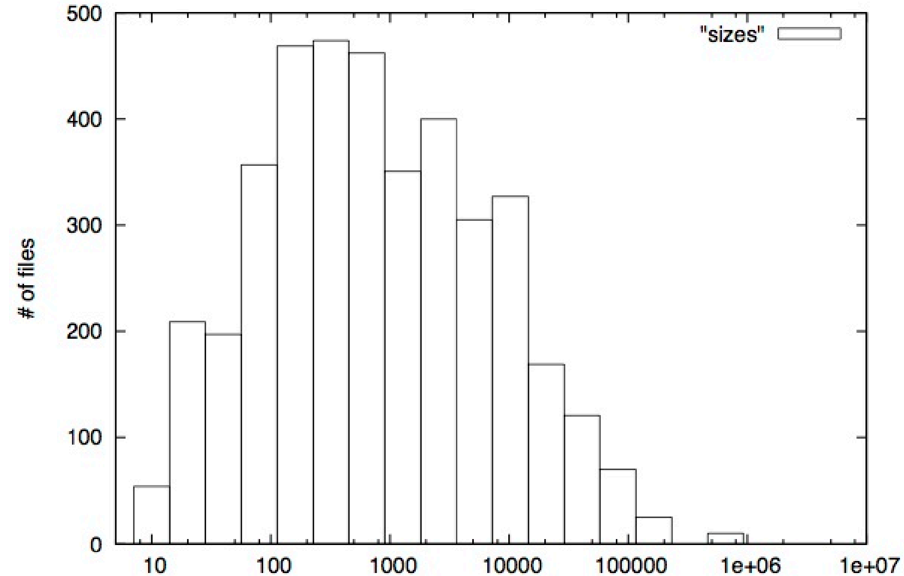
Initial Tests – Large Files

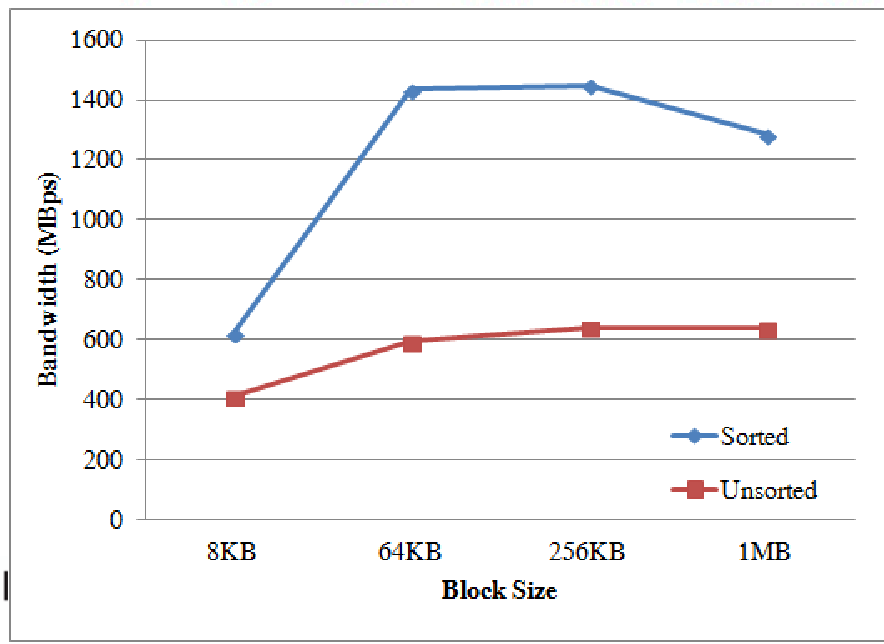




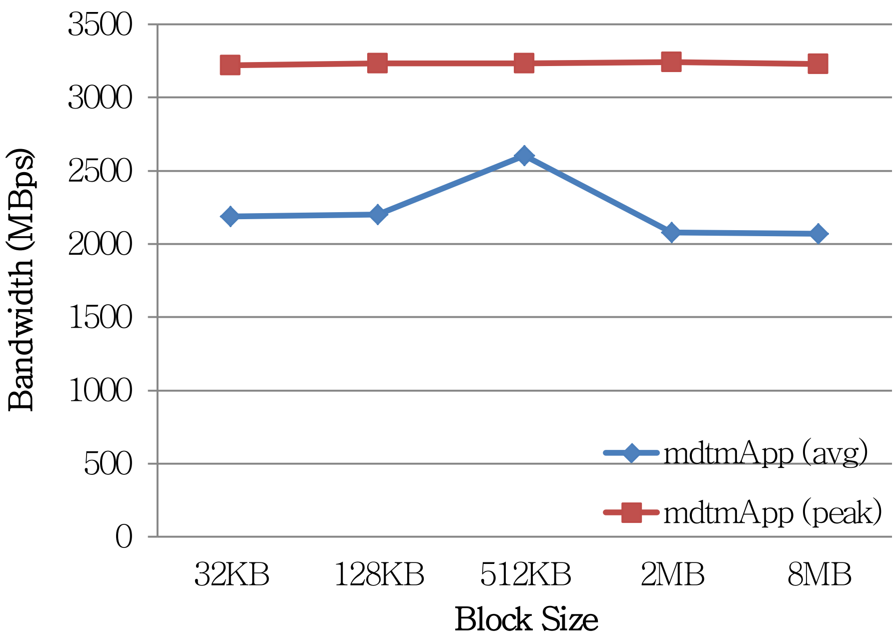
Initial Tests – Small Files

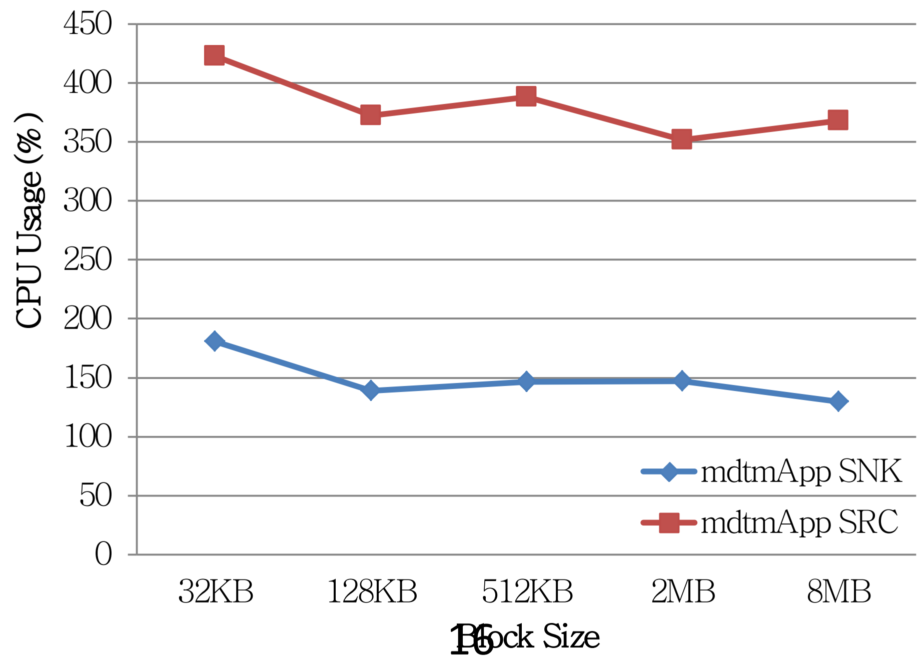
- Parallel streams from 4 RAIDs at source to /dev/null at destination, total 4,000 files with log-normal size distribution
- Techniques used: locality-aware binding/grouping, sorting, pipelining





Initial Tests – Small Files





Current Status

- We are on schedule, with both the application and middleware teams achieving their year-1 milestone goals.
- Major modules have been implemented
  - Thread/flow management, request preprocessing, and various data access/transmission methods. (by BNL)
  - Multicore system profiling, topology-based resource scheduling, interrupt affinity for network I/O, and web-based monitoring and management (by FNAL)
- What questions now to ask?
  - With new Intel Knight landing architecture and external NUMA-link by SGI, NUMA expands horizontally among clusters and vertically with the intra node level, Is there any standard middleware, API, library to support intelligent scheduling?
  - Asynchronous event-driven model and synchronous parallel threads for end-to-end data transfer flows.

Future Work

- MDTM middleware future work
  - Web-based remote monitoring capability
    - Online and real-time monitoring of specific data transfer's status and progress
    - Online and real-time monitoring of data transfer node system status
  - Web-based remote management capability
  - Support core affinity on disk I/O
  - Support QoS mechanisms for differentiated data transfer
- MDTM application future work
  - Load balancing among task groups
  - Dynamic and flexible allocation of resources such as thread pool and buffer to accommodate dynamic user loads.
  - Client-server interaction
  - Performance monitoring and reports to users
  - Intensive tests on large-scale testbeds

Questions?

Demo

http://mdtm-server.fnal.gov:1337