Single Run-Time Environment

Yutaka Ishikawa, Atsushi Hori, Hiroya Matsuba, Yoshikazu Kamoshida, Kazuki Ohta

(University of Tokyo)

Shinji Sumimoto (Fujitsu Laboratory)

Takashi Yasui (Hitachi)

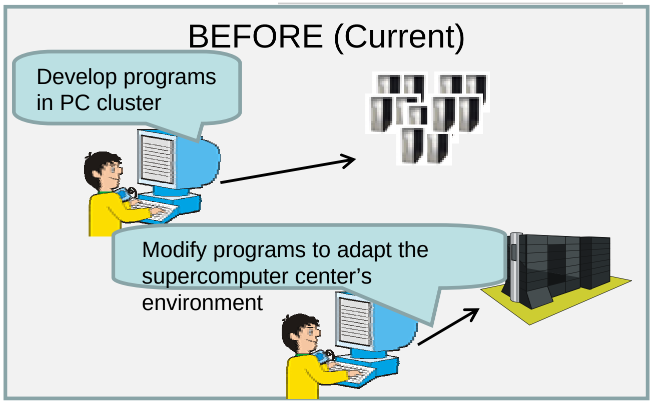
T2K Open Supercomputer Alliance

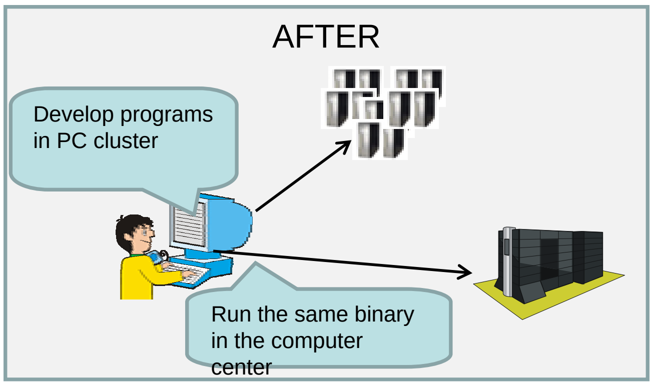
Motivations and Objectives

- Motivations

  Though the commodity clusters are built using x86 CPU and Linux, the application binaries developed in a machine environment could not run in other machine environments due to the following reasons:

  - Local disk usage
    - local disks may be used in the user cluster
    - the usage of local disks depends on the center policy
  - File system scalability
    - 1,000 processes or less in PC cluster
    - 10,000 or more processes in center machine
  - MPI standard does not specify the application binary interface
  - No standard of batch script
- Objectives
  - Single binary runs everywhere


BEFORE (Current)
Develop programs in PC cluster
Modify programs to adapt the supercomputer center's environment

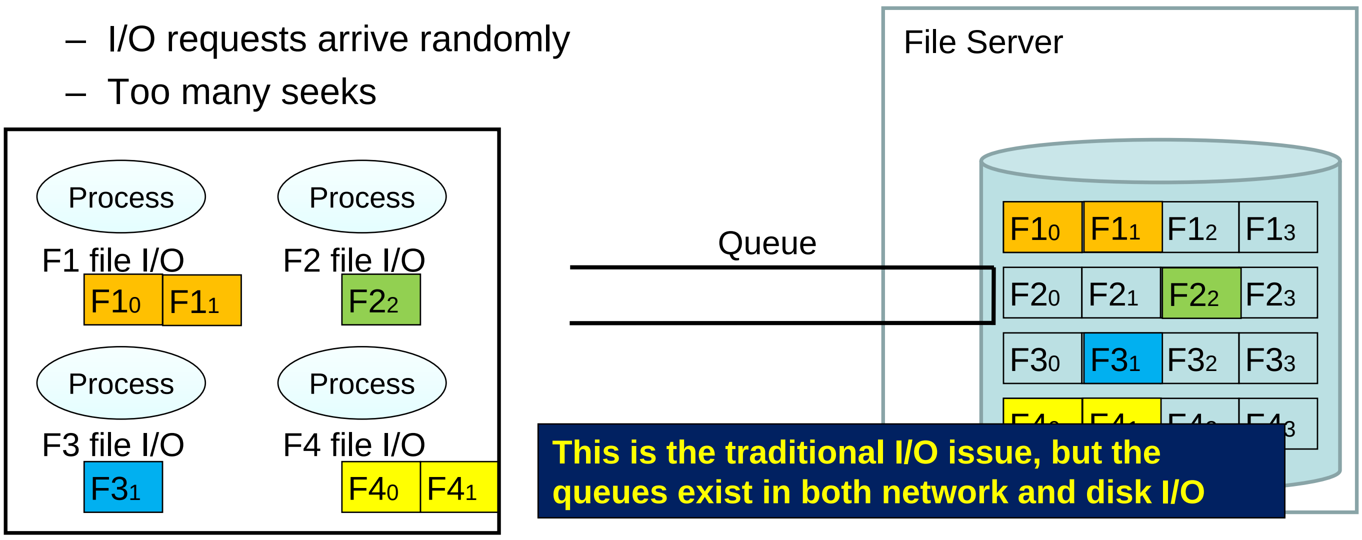

AFTER
Develop programs in PC cluster
Run the same binary in the computer center

Ongoing Research

- File System
  - pdCache [Kazuki Ohta]
    - File cache system
  - CatWalk [Atsushi Hori]
    - Transparent file staging system
  - STG [Hiroya Matsuba]
    - Portable high-performance file staging system
  - File Access Tracer [Takashi Yasui]
    - Understanding the application I/O behavior
- MPI-Adapter [Shinji Sumimoto]
  - Binary compiled under some MPI implementation may run under other MPI implementations

File System Issue: Seek

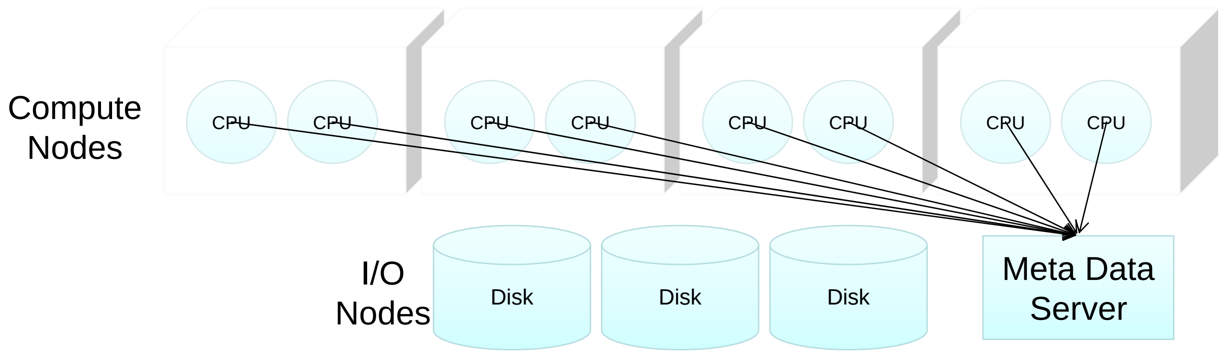
- Many Cores and File Accesses
  - Assuming that each process runs on each core
    - Eg., 4 processes runs on 1 node with 4 cores
  - Each process requests sequential access to a file on the file server
- Server side
  - I/O requests arrive randomly
  - Too many seeks



This is the traditional I/O issue, but the queues exist in both network and disk I/O

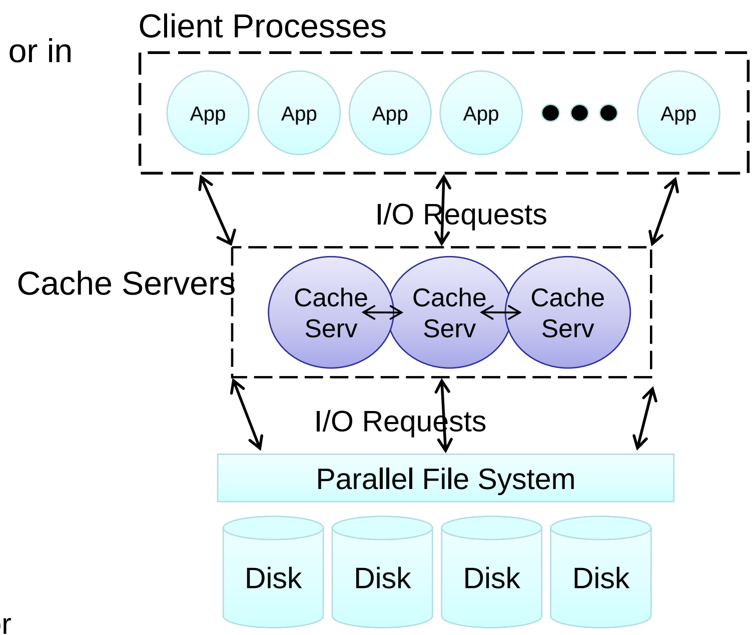
File System Issue: Meta Data Handling

- Meta data server



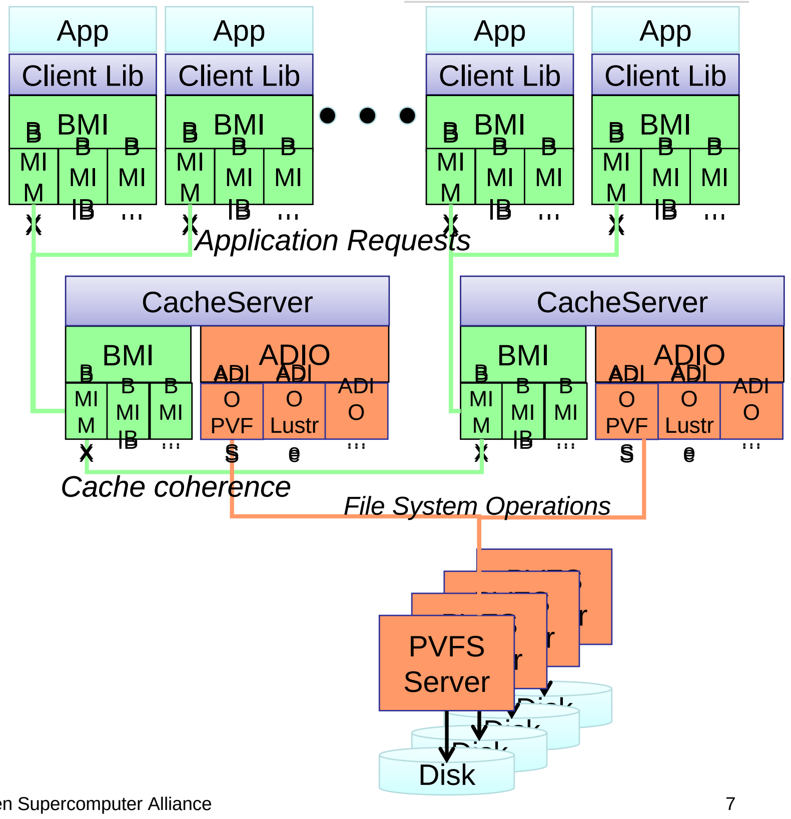
pdCache

- Cache Servers
  - May be located in compute nodes or in some independent nodes
  - cache file and meta data
  - Reduces
    - Disk Seeks
    - Disk I/O Requests
    - Meta data access
  - Handles client requests fairly
- Portability
  - Independent of file system
  - Cluster networks
- Related Work
  - ZOID: I/O Forwarding Infrastructure for Petascale Architectures [Iskra, PPoPP08]
  - Scalable I/O Performance through I/O delegate and Caching System [Nisar, SC08]



pdCache: Software Stack

- ADIO: Abstract Device Interface for I/O [Thakur96]
  - Is designed in ROMIO for MPI-IO
  - Supports most parallel file systems
- BMI: Buffered Message Interface [Carns05]
  - Is designed in the PVFS2 file system
  - Supports most cluster networks
- A remote procedure call mechanism is implemented in BMI
  - To handle application requests
  - To communicate with CacheServer's



pdCache: Evaluation

- Coming Soon ☺

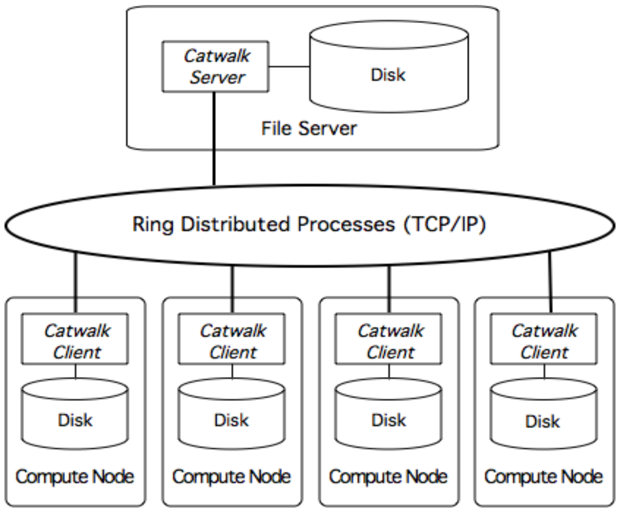
Catwalk: An Overview

- Transparent File Staging
  - The users do not take care of the file staging commands, but the Catwalk midleware takes care of it
    - At a file open, the Catwalk copies the file from the file server to the local disk if the file does not exist in the local disk
    - At a file close, the Catwalk copies the file from the local disk to the file server
- Assuming Environment
  - TCP/IP connection between the file server and the cluster
    - Requires some coordination of network traffic
  - No requirement of highly network bandwidth
  - No requirement of the administrator mode to install Catwalk

- New Oxford America Dictionary

  A ***narrow walkway*** or platform extending into an auditorium, esp. in an industrial installation, along which models walk to display clothes in fashion shows.
- http://en.wikipedia.org/wiki/Catwalk

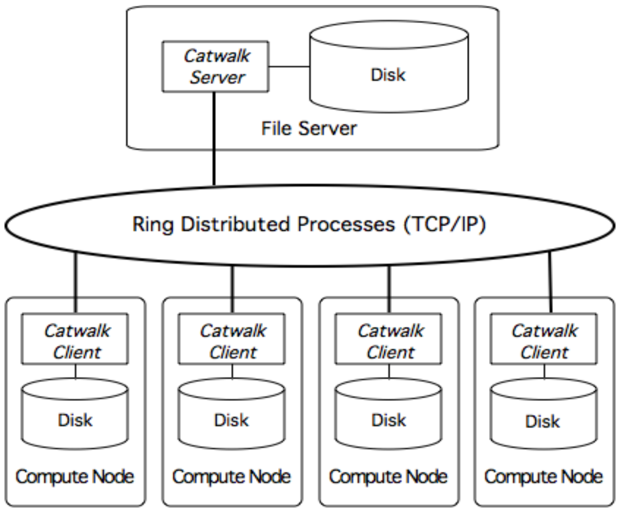
  Typically, catwalks are located in positions ***hidden from audience*** view or directly above an audience, and are considered "behind-the-scenes".



Catwalk: An Overview

- Transparent File Staging
  - The users do not take care of the file staging commands, but the Catwalk midleware takes care of it
    - At a file open, the Catwalk copies the file from the file server to the local disk if the file does not exist in the local disk
    - At a file close, the Catwalk copies the file from the local disk to the file server
- Assuming Environment
  - TCP/IP connection between the file server and the cluster
    - Requires some coordination of network traffic
  - No requirement of highly network bandwidth
  - No requirement of the administrator mode to install Catwalk

- Catwalk consists of
  - user library
  - Client process
  - Server process



Catwalk: Stage In

1. The open system call is intercepted
2. A Catwalk client sends the stage-in request to the Catwalk server
3. The Catwalk server receives the request and enqueues it to the stage-in queue

The Catwalk server

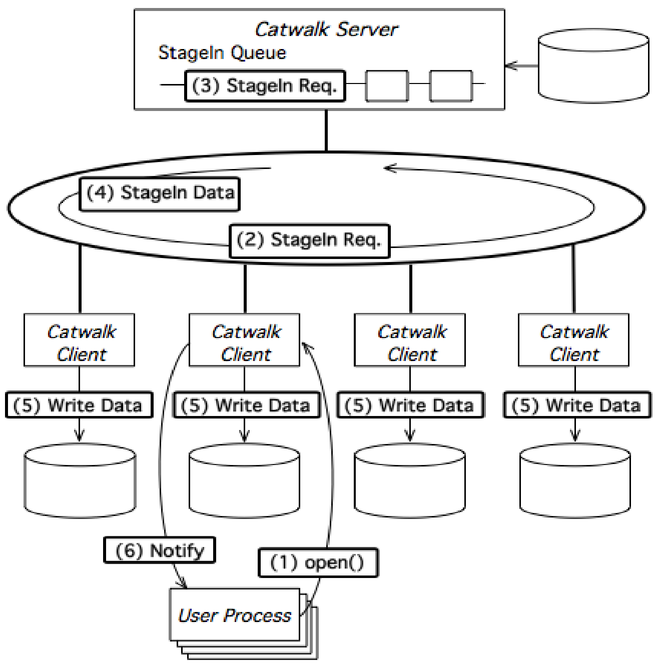
4. While the stage-in queue is not empty
   - ✓ dequeues a request from the stage-in queue
   - ✓ sends the requested file to a cluster node along with the ring topology

A Catwalk client

When the stage-in file arrives

- ✓ Sends the file to the next cluster node along with the ring toplogy

5. Writing the file to the local disk
6. Notifies the user process



Catwalk: Stage Out

1. The create system call is intercepted
2. A Catwalk client enqueues the stage-out request to its request
3. When a Catwalk client receives the signal from the user process at the process exiting, this event is sent to the Catwalk server
4. When the Catwalk server receives all the exiting events from the clients, the stageOut token is sent to the cluster node

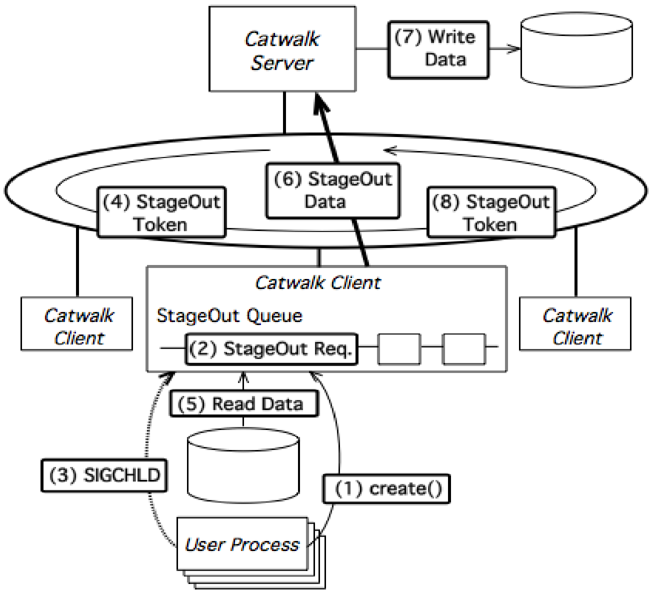
A Catwalk client

4. At receiving the stageOut token, the following procedures are performed until the stageout queue becomes empty:
   - ✓ dequeues a request from the stage-out queue
   5. Read the file, and 6. sends the file to the server
6. Sends the stageOut token to the next node in the ring topology

The Catwalk server

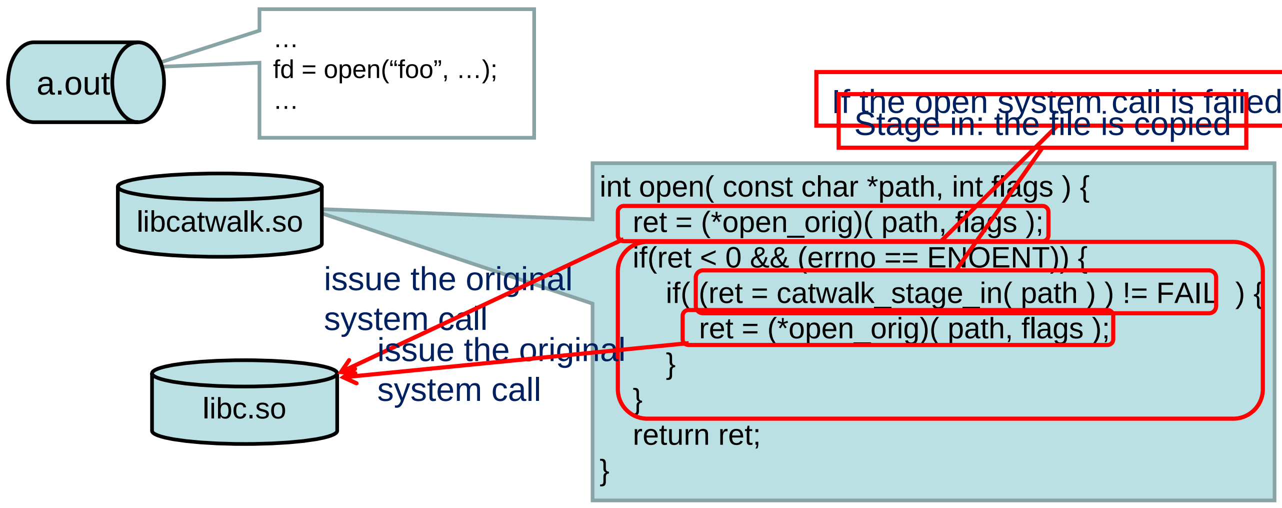
When the stage-out file arrives

7. Stores the file to the file system



CatWalk: User Library Implementation

- Hooking the open system call
  - Using the LD_PRELOAD feature of Linux
    - The dynamic library specified by the LD_PRELOAD environment variable is used prior to the system dynamic libraries


a.out
libcatwalk.so
libc.so
issue the original system call
issue the original system call
If the open system call is failed
Stage in: the file is copied

```
…
fd = open("foo", …);
…
```

```
int open( const char *path, int flags ) {
    ret = (*open_orig)( path, flags );
    if(ret < 0 && (errno == ENOENT)) {
        if( (ret = catwalk_stage_in( path ) ) != FAIL  ) {
            ret = (*open_orig)( path, flags );
        }
    }
    return ret;
}
```

CatWalk: Evaluation

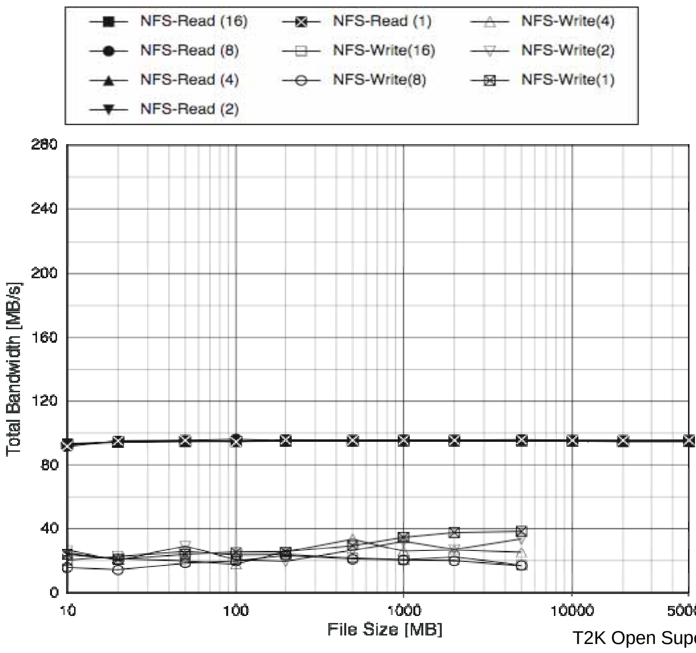
- T2K Open Supercomputer
- 17 nodes
  - One for file server and 16 for compute nodes
- Network
  - 1Gbps Ethernet

CPU	AMD Barcelona, 2.3GHz, 4x4 cores
Memory	32 GB
Local Disk	SATA
Network	Intel E1000, Myrinet 10G
OS	RHELS 5.1
File System	EXT3, NFS Ver.3

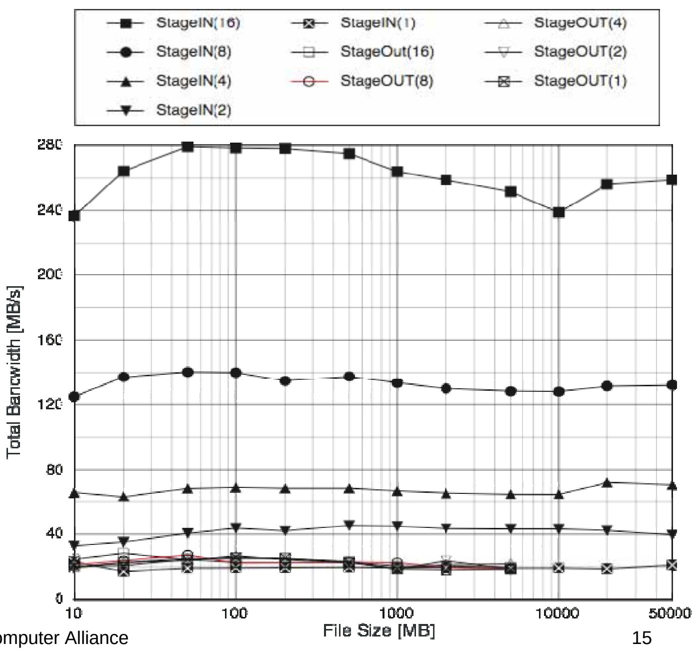
CatWalk: Evaluation

```
Stage in                                   Stage out
    CREATE_FILE                                MPI_Barrier();
    READ_32GB_FILE                             create();
    MPI_Barrier();                             while( ) write();
    open();                                    close();
    while( read() > 0 );      Time             if( CATWALK )            Time
    close();                                       force_stageout();
    MPI_Barrier();                             MPI_Barrier();
                                               SYNC_ON_SERVER
```

NFS



CatWalk

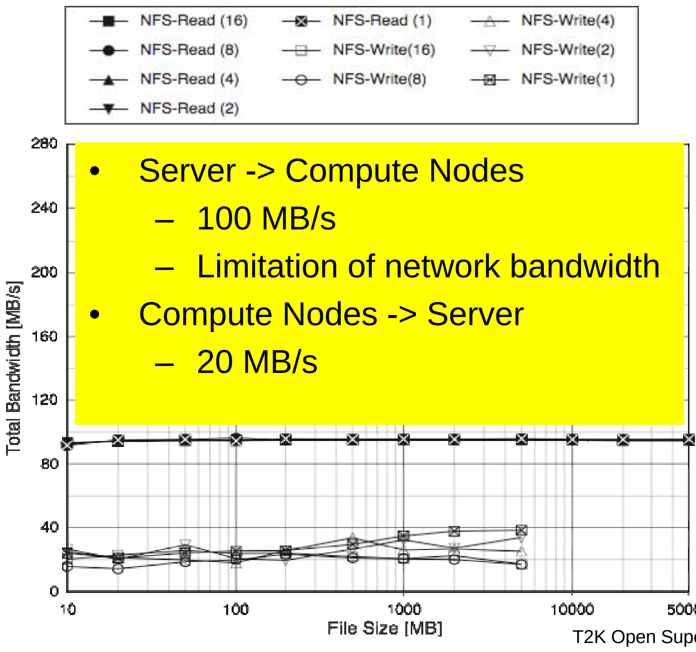


CatWalk: Evaluation

```
Stage in                                   Stage out
    CREATE_FILE                                MPI_Barrier();
    READ_32GB_FILE                             create();
    MPI_Barrier();                             while( ) write();
    open();                                    close();
    while( read() > 0 );      Time             if( CATWALK )            Time
    close();                                       force_stageout();
    MPI_Barrier();                             MPI_Barrier();
                                               SYNC_ON_SERVER
```

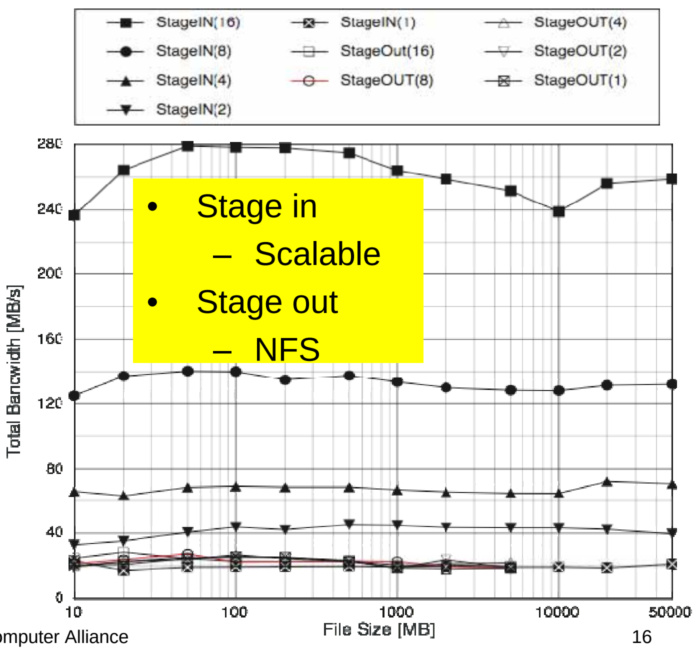
NFS

- Server -> Compute Nodes
  - 100 MB/s
  - Limitation of network bandwidth
- Compute Nodes -> Server
  - 20 MB/s



CatWalk

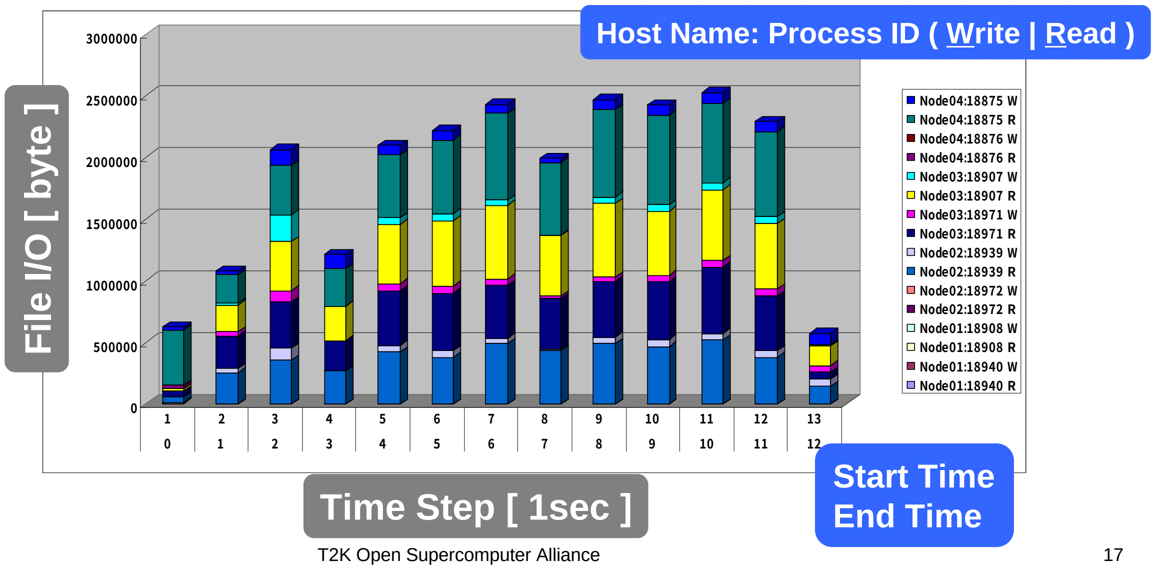
- Stage in
  - Scalable
- Stage out
  - NFS



File Access Tracer

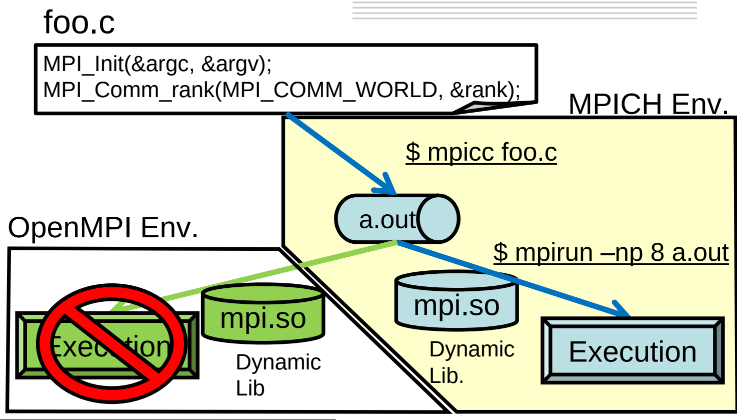
- To understand the application I/O behavior
- Hooking open/creat/read/write system calls to get the file access pattern
- Using LD_PRELOAD feature
- No recompilation

ProteinDF File I/O



MPI Portability Issue

- No ABI (Application Binary Interface)
  - Ex. MPI_Comm type is the address type in OpenMPI while the MPI_Comm type is 32 bit integer in other implementations

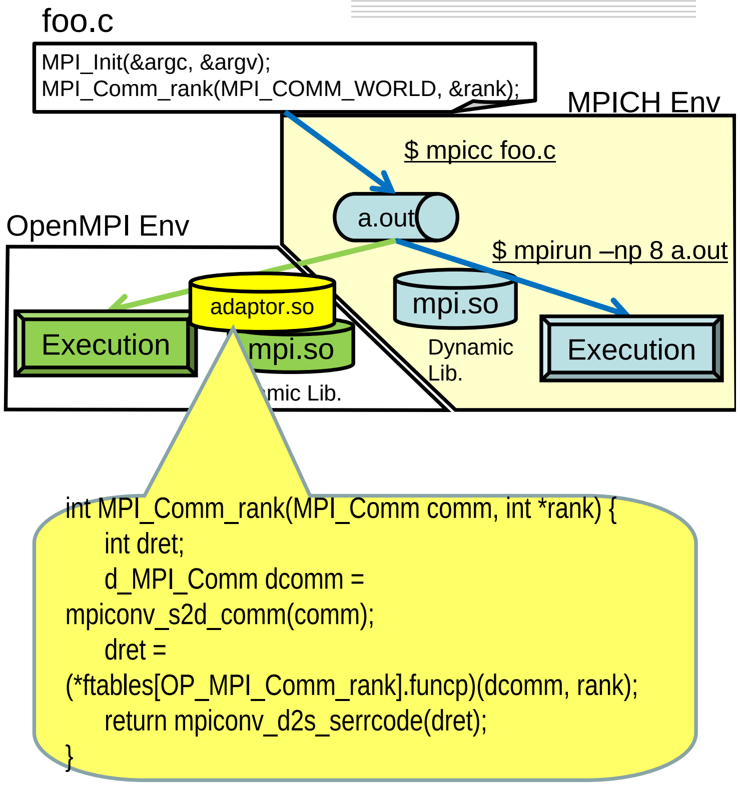


Constant	MPICH2	OpenMPI
MPI_COMM_WORLD	0x44000000	&ompi_mpi_comm_world
MPI_INT	0x4c000405	&ompi_mpi_int
MPI_INTEGER	0x4c00041b	&ompi_mpi_integer
MPI_SUCCESS	0	0
MPI_ERR_TRUNCATE	14	15
MPI_COMM_WORLD	0x44000000	0
MPI_INTEGER	0x4c00041b	&ompi_mpi_integer
MPI_SUCCESS	0	0
MPI_ERR_TRUNCATE	14	15

MPI-Adapter

- Adapter.so
  - The LD_PRELOAD feature is used
  - At the MPI_Init function,
    - The target MPI library is opened using *dlopen()*
    - All MPI function addresses defined in the target library are collected

Example
- The communicator in MPICH is converted to one in OpenMPI
- MPI_Comm_rank in OpenMPI is invoked
- The return value is converted to one in MPICH



```
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
```

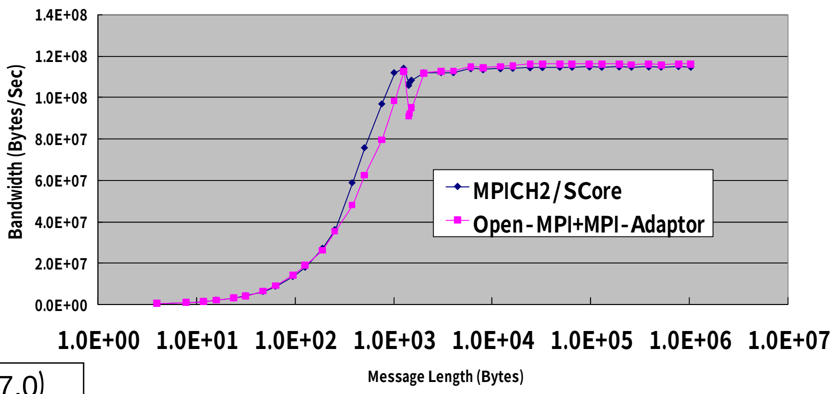
```
int MPI_Comm_rank(MPI_Comm comm, int *rank) {
    int dret;
    d_MPI_Comm dcomm =
mpiconv_s2d_comm(comm);
    dret =
(*ftables[OP_MPI_Comm_rank].funcp)(dcomm, rank);
    return mpiconv_d2s_serrcode(dret);
}
```

MPI-Adapter: Evaluation

- MPI-Pingpong(mpi_rtt)
- MPICH2/SCore
  - Compiled under the MPICH2/SCore environment
- OpenMPI+MPI-Adaptor
  - Compiled under the OpenMPI environment
  - Runs under the MPICH2/SCore environment with MPI-Adapter

	RTT(usec)	Ratio
MPICH2/SCore	43.328	100%
OpenMPI+ MPI-Adaptor	43.440	100.2%

MPI Burst Bandwidth



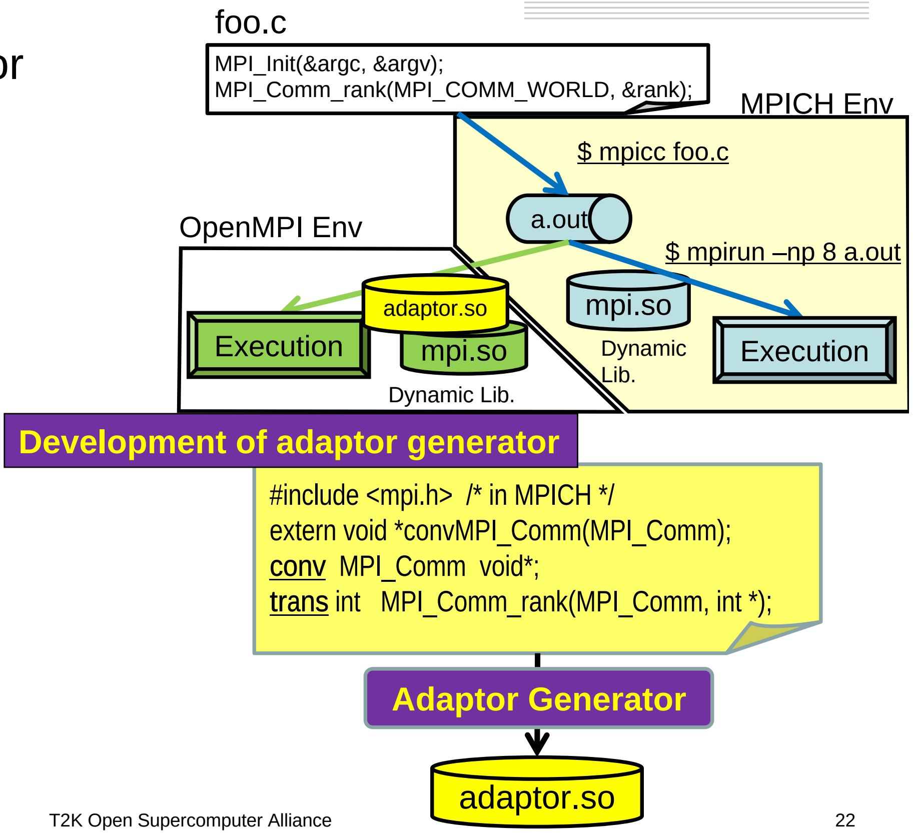
RX200S2 Cluster (Xeon 3.8GHz, SCore7.0)
Network: Intel E1000 NIC,
Netgear 48Port Switch
MPI MPICH2/SCore w/ PMX/Etherhxb

MPI-Adapter: NAS Parallel Benchmark IS

	Class A	Class B	Class C
MPICH2/SCore	45.90	52.27	70.20 Mops
OpenMPI+ MPI-Adaptor	46.10	49.77	70.02 Mops

MPI-Adapter: Future Work

- The adapter generator
  - Generates stub routines



```
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
```

```
#include <mpi.h>  /* in MPICH */
extern void *convMPI_Comm(MPI_Comm);
conv  MPI_Comm  void*;
trans int   MPI_Comm_rank(MPI_Comm, int *);
```

Concluding Remarks

- Single Runtime Environment
  - CatWalk, MPI-adaptor, File AccessTracer
    - Will be distributed with SCore version 7 at Q2 of 2009
    - Runs in any Linux cluster without root access rights
  - Portable File Staging System
    - Is also being developed
- High-level file I/O library
  - HDF (Hierarchical Data Format) ?
    - http://www.hdfgroup.org/
  - File Access Tracer is used to
    - Gather application file I/O access patterns to think of better file I/O library design