



Data Services and Trilinos

SAND2011-8799C

Addressing I/O Challenges for Exascale Applications

Approved for Public Release: SAND2011-XXXXP

SC11

Nov, 2011

Ron Oldfield

Sandia National Laboratories



Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.



I/O Challenges for Exascale

- Storage systems are the slowest, most fragile, part of an HPC system
- Current usage models not appropriate for Petascale, much less Exascale
 - Checkpoints are a **HUGE** concern for I/O...currently primary focus of FS
 - App workflow uses storage as a communication conduit
 - Simulate, **store**, analyze, **store**, refine, **store**, ... most of the data is transient
 - High-level I/O libraries (e.g., HDF5, netCDF) have high overheads
- Trios Data Services to the rescue!
 1. Reduce the “effective” I/O cost through data staging
 2. Reduce amount of data written to storage (integrated analysis, data services)
- Nothing comes for free...
 - We use additional compute and memory resources
 - Data services introduce issues with resilience (we’re addressing this)

Trios Data Services

I/O Software to Reduce I/O

Approach

- Leverage available compute/service node resources for I/O caching and data processing

Application-Level I/O Services

- First used for seismic imaging (mid 90s)
- PnetCDF staging service
- CTH real-time analysis
- SQL Proxy (for NGC)
- Interactive sparse-matrix visualization (for NGC)

Client Application
(compute nodes)

I/O Service
(compute/service nodes)

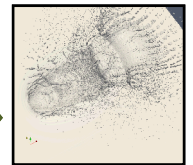
Raw Data

Processed Data

Cache/aggregate
/process



Lustre File System



Visualization Client

NETEZZA

LexisNexis®

Nessie (NEtwork Scalable Service InterfacE)

- Framework for developing data services
- Client and server libs, cmake macros, utilities
- Originally developed for lightweight file systems

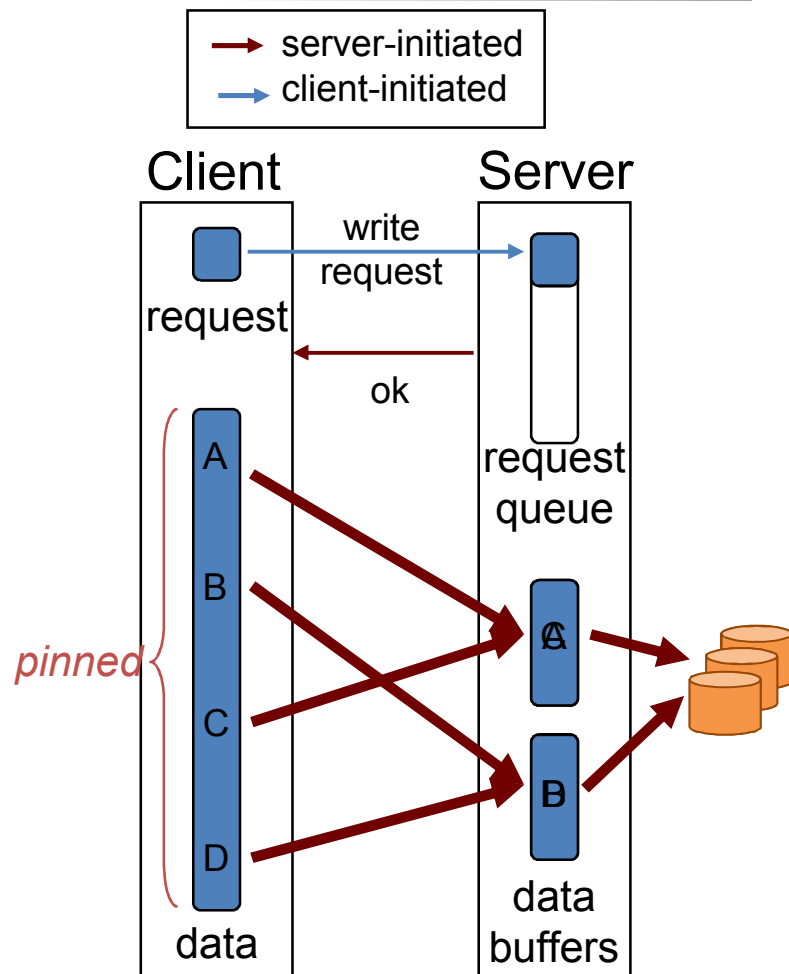


Network-Scalable Service Interface

Some Details on Nessie

Designed for Bulk Data Movement on HPC Platforms

- Goals of data-movement protocol
 - Low stress on servers (assume order of magnitude more clients than servers)
 - Efficient use of network (avoid copies, dropped messages, retransmissions, ...)
- Features of Nessie
 - Asynchronous, RPC-like API
 - User low-level RDMA transports
 - Portals, InfiniBand, Gemini
 - Small requests
 - Server-directed for bulk data
 - Writes: pull from client
 - Reads: push to client

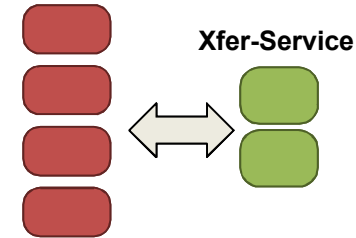


Example: A Simple Transfer Service

Trilinos/packages/trios/examples/xfer-service

- Used to test Nessie API
 - **xfer_write_encode**: client transfers data to server through RPC args
 - **xfer_write_rdma**: server pulls raw data using RDMA get
 - **xfer_read_encode**: server transfers data to client through RPC result
 - **xfer_read_rdma**: server transfers data to client using RDMA put
- Used for performance evaluation
 - Test low-level network protocols
 - Test overhead of XDR encoding
 - Tests async and sync performance
- Creating the Transfer Service
 - Define the XDR data structs and API arguments
 - Implement the client stubs
 - Implement the server

Client Application



```
/* Data structure to transfer */
struct data_t {
    int int_val;           /* 4 bytes */
    float float_val;      /* 4 bytes */
    double double_val;    /* 8 bytes */
};

/* Array of data structures */
typedef data_t data_array_t<>;

/* Arguments for xfer_write_encode */
struct xfer_write_encode_args {
    data_array_t array;
};

/* Arguments for xfer_write_rdma */
struct xfer_write_rdma_args {
    int len;
};

...
```

Transfer Service

Implementing the Client Stubs

- Interface between scientific app and service
- Steps for client stub
 - Initialize the remote method arguments, in this case, it's just the length of the array
 - Call the rpc function. The RPC function includes method arguments (*args*), and a pointer to the data available for RDMA (*buf*)
- The RPC is asynchronous
 - The client checks for completion by calling `nssi_wait(&req)`;

```
int xfer_write_rdma(  
    const nssi_service *svc,  
    const data_array_t *arr,  
    nssi_request *req)  
{  
    xfer_write_rdma_args args;  
    int nbytes;  
  
    /* the only arg is size of array */  
    args.len = arr->data_array_t_len;  
  
    /* the RDMA buffer */  
    const data_t *buf=array->data_array_t_val;  
  
    /* size of the RDMA buffer */  
    nbytes = args.len*sizeof(data_t);  
  
    /* call the remote methods */  
    nssi_call_rpc(svc, XFER_PULL,  
        &args, (char *)buf, nbytes,  
        NULL, req);  
}
```

Transfer Service

Implementing the Server

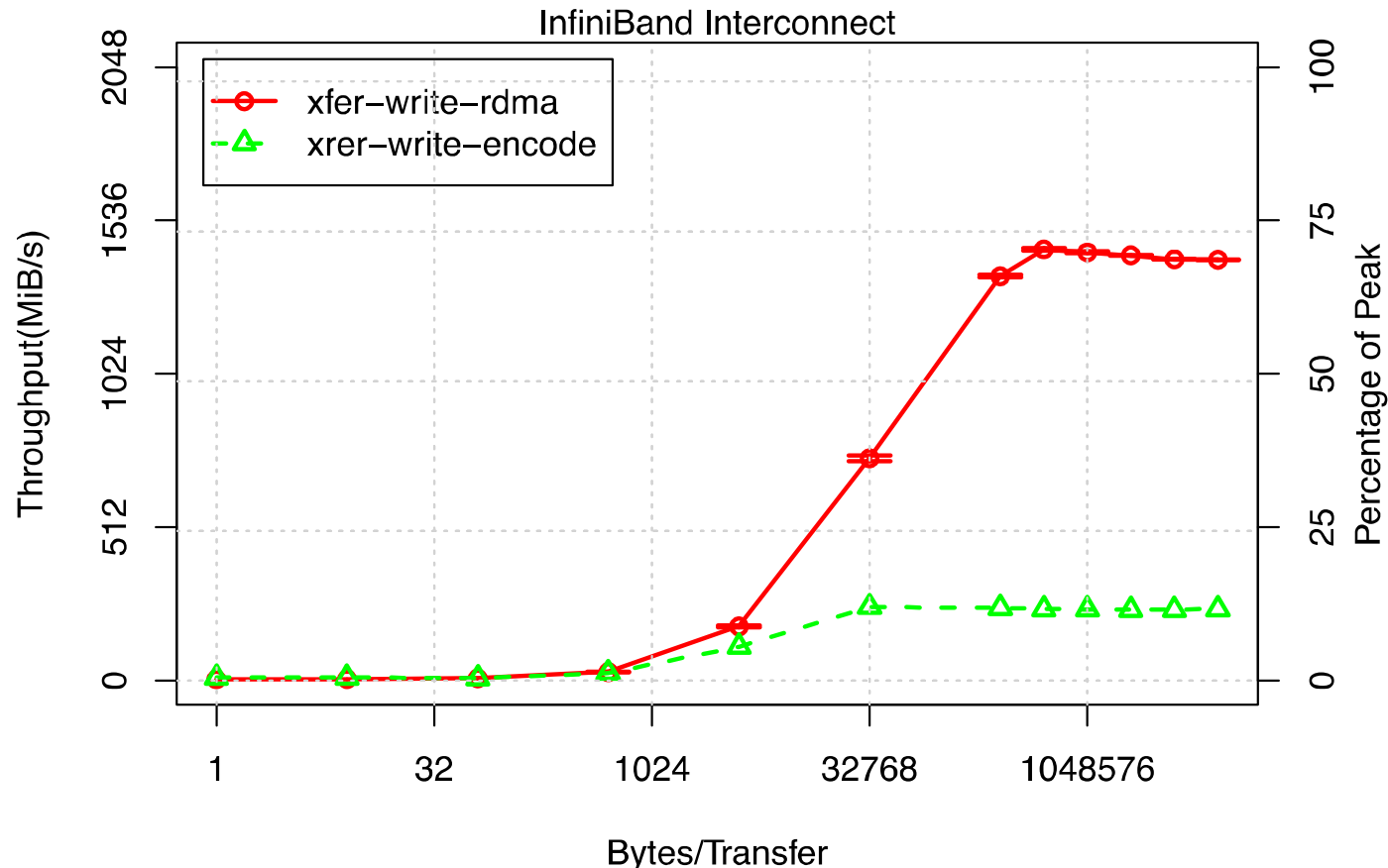
- Implement server stubs
 - Using standard stub args
 - For `xfer_write_rdma_srvr`, the server pulls data from client
- Implement server executable
 - Initialize Nessie
 - Register server stubs/callbacks
 - Start the server thread(s)

```
int xfer_write_rdma_srvr(  
    const unsigned long request_id ,  
    const NNTI_peer_t *caller ,  
    const xfer_pull_args *args ,  
    const NNTI_buffer_t *data_addr ,  
    const NNTI_buffer_t *res_addr)  
{  
    const int len = args->len;  
    int nbytes = len*sizeof(data_t);  
  
    /* allocate space for the buffer */  
    data_t *buf = (data_t *)malloc(nbytes);  
  
    /* fetch the data from the client */  
    nssi_get_data(caller , buf , nbytes , data_addr);  
  
    /* send the result to the client */  
    rc = nssi_send_result(caller , request_id ,  
        NSSI_OK, NULL, res_addr);  
  
    /* free buffer */  
    free(buf);  
}
```

Evaluating the Transfer Service

InfiniBand Interconnect

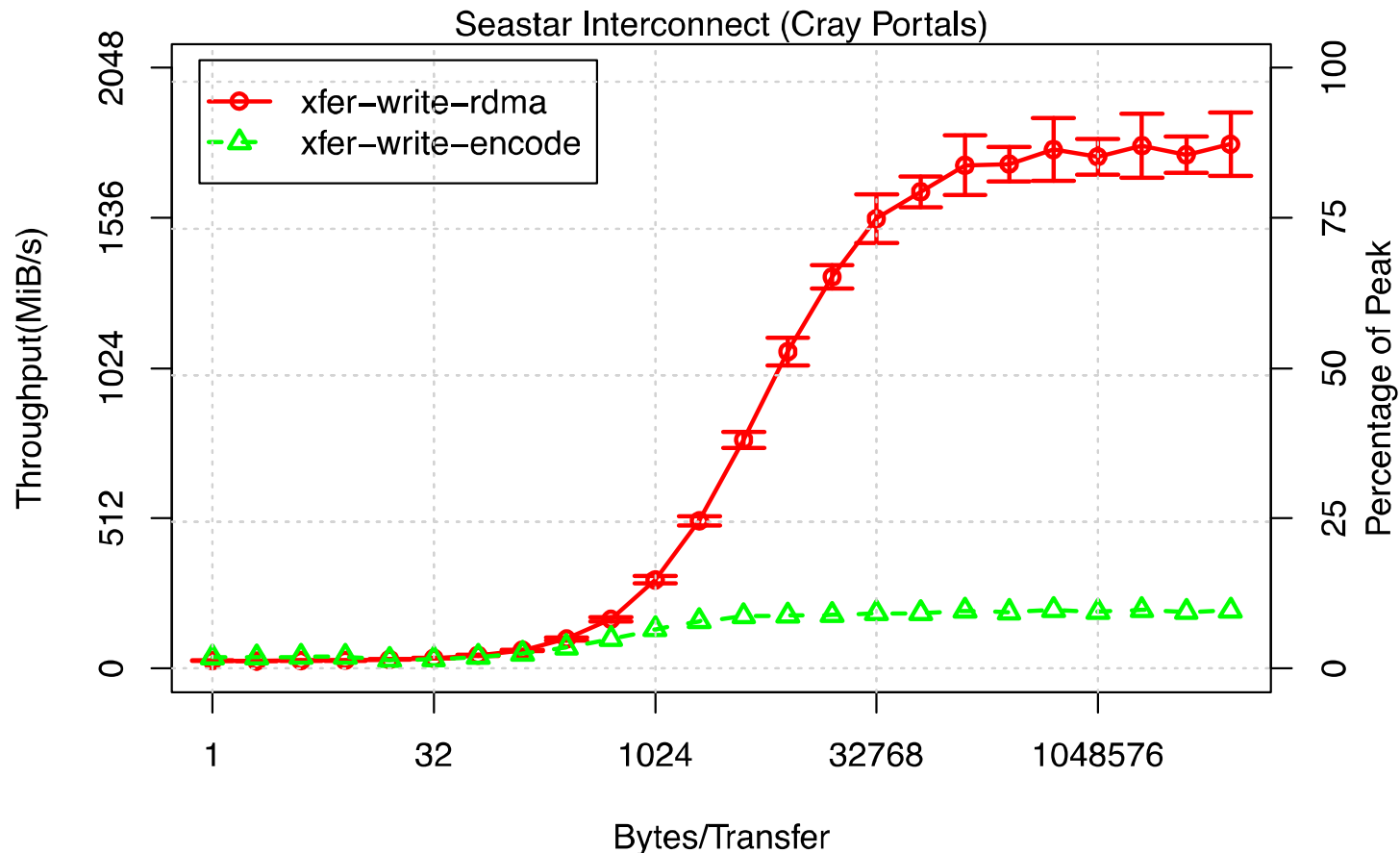
Comparison of Write Methods on RedSky



Evaluating the Transfer Service

SeaStar Interconnect (Portals)

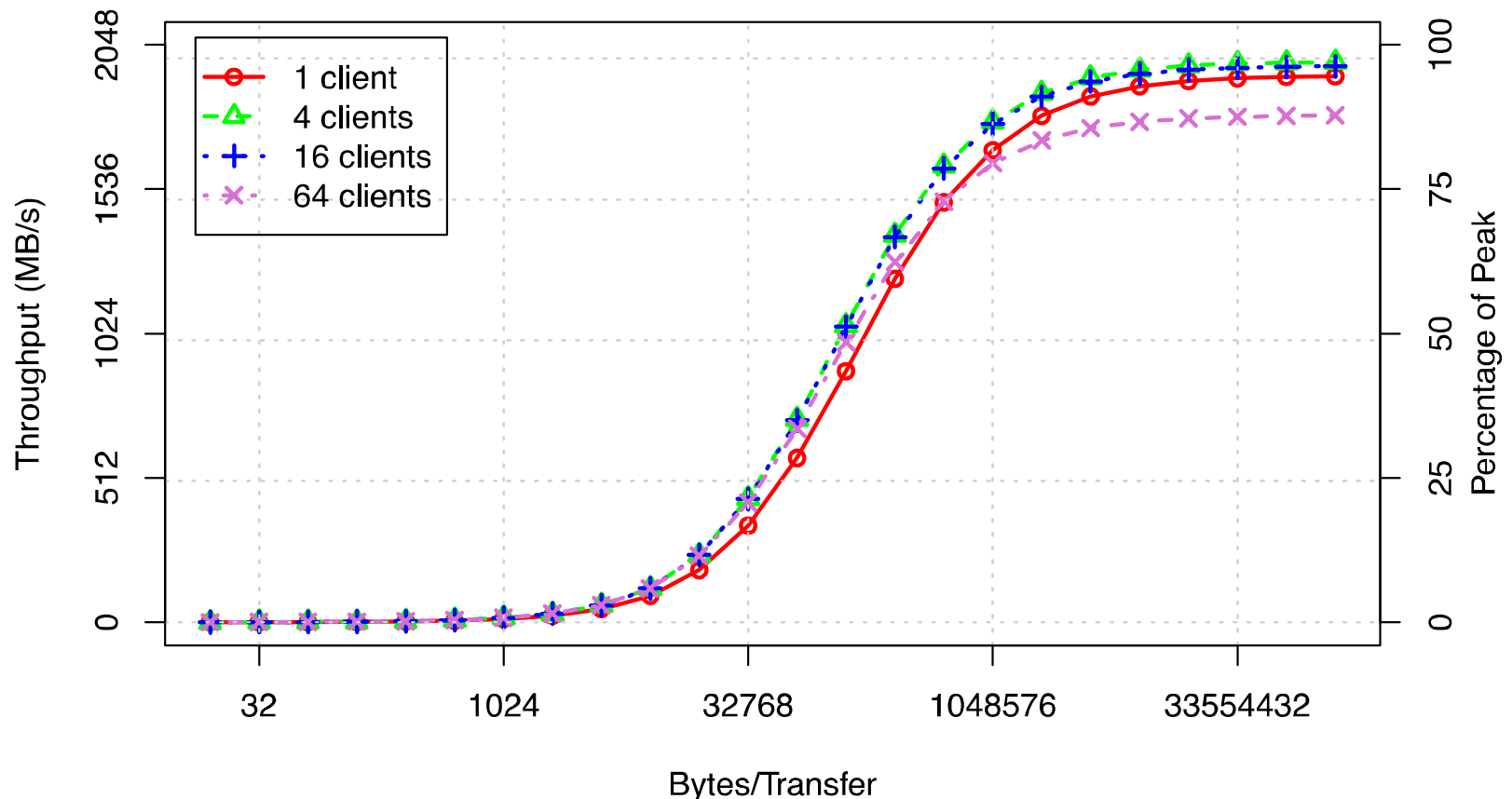
Comparison of Write Methods on RedStorm



Evaluating the Transfer Service

SeaStar Interconnect (Portals)

NSSI Scaling Performance on Red Storm
SeaStar Network



Data-Service Applications

Salvo Seismic Imaging: The Inspiration

Salvo's I/O Partition

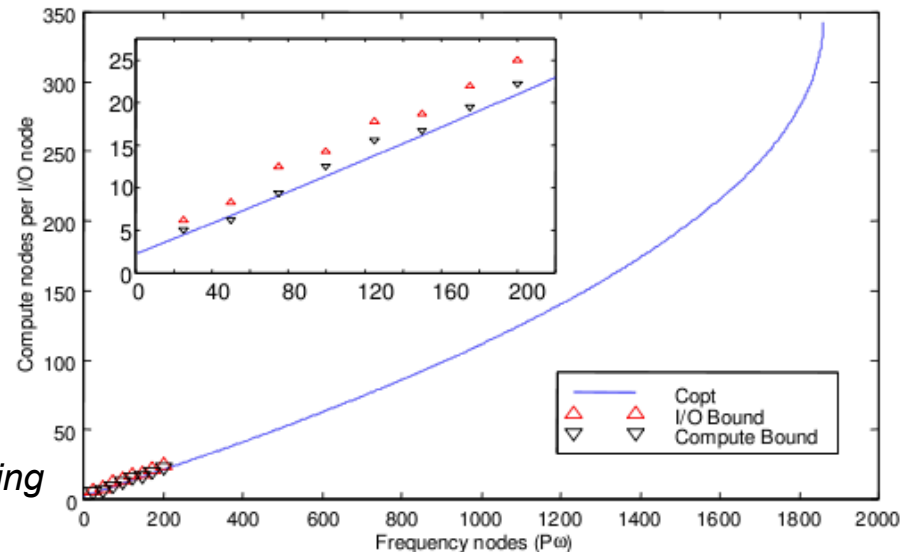
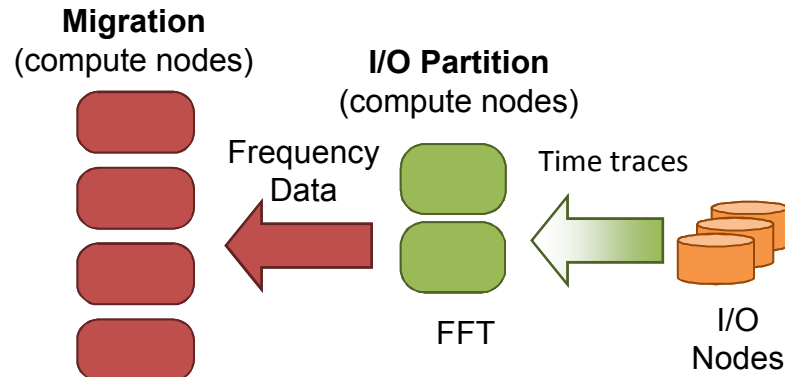
- Partition of application processors (used separate MPI Communicator for I/O)
- Used for FFT, I/O cache, and interpolation
- Async I/O allowed overlap of I/O and computation (pre-process next step)

Results

- +10% nodes led to +30% in performance
- Modeling I/O and compute costs helped find the right balance of compute and I/O nodes

Contacts: Ron Oldfield, Curtis Ober
{raoldfi,ccober}@sandia.gov

Oldfield, et al. Efficient parallel I/O in seismic imaging.
The International Journal of High Performance Computing Applications, 12(3), Fall 1998

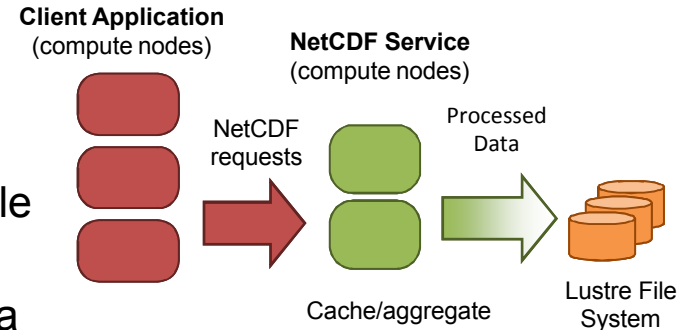


Scalable I/O Services

NetCDF I/O Cache

Motivation

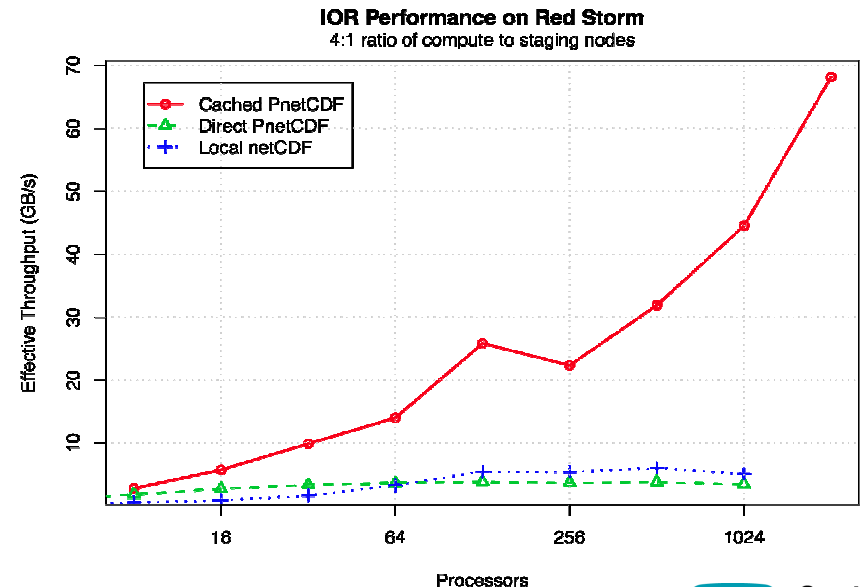
- Synchronous I/O libraries require app to wait until data is on storage device
- Not enough cache on compute nodes to handle “I/O bursts”
- NetCDF is basis of important I/O libs at Sandia (Exodus)



NetCDF Caching Service

- Service aggregates/caches data and pushes data to storage
- Async I/O allows overlap of I/O and computation

Presented at PDSW'11



Scalable I/O Services

CTH Fragment Detection

Motivation

- Fragment detection process takes 30% of time-step calculation
- Fragment tracking requires data from every time step (too data intensive for post processing)
- Integrating detection software with CTH is intrusive on developer

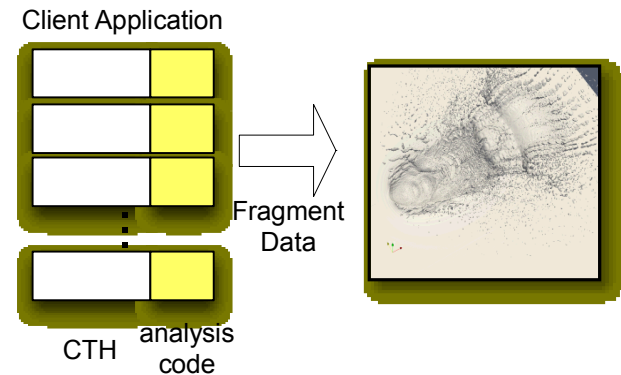
CTH fragment detection service

- Extra compute nodes provide in-line processing (overlap fragment detection with time step calculation)
- Only output fragments to storage (reduce I/O)
- Non-intrusive
 - Looks like normal I/O (pvspy interface)
 - Can be configured out-of-band

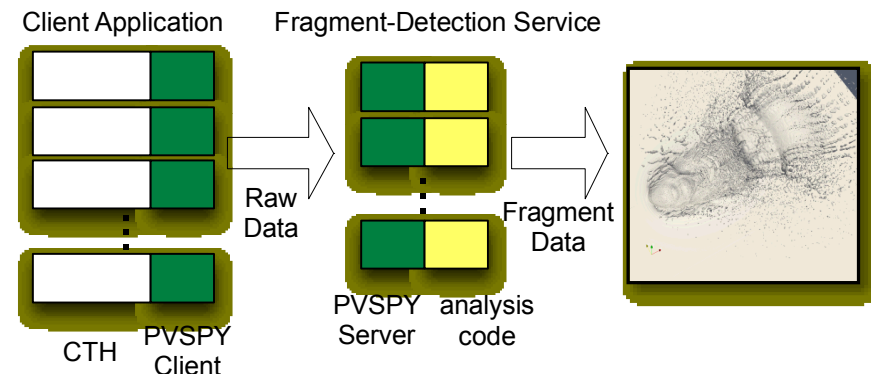
Status and Ongoing Work

- Porting to Cielo
- Comparison of in-situ and in-transit

In-Situ Fragment Detection



In-Transit Fragment Detection

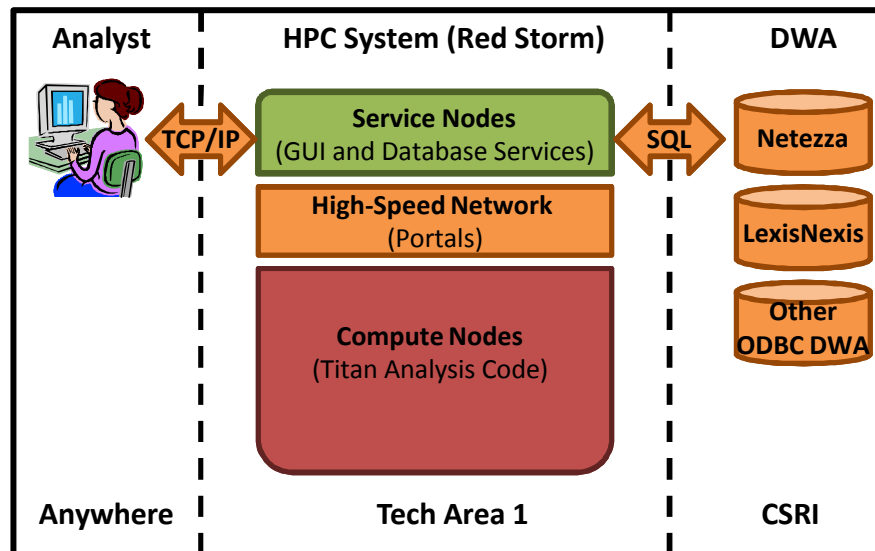


Data-Service Applications

SQL Service: Remote Access to Data Warehouse Appliances (DWA)

SQL Service*

- Provides “bridge” between parallel apps and external DWA
- Runs on Red Storm network nodes
- Titan applications communicate with service through Portals
- External resources (Netezza) communicate through standard interfaces (e.g. ODBC over TCP/IP)



The SQL service enables an HPC application to access a remote DWA

Additional Modifications for Multilingual

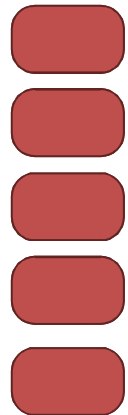
- Tokenization support on Netezza (goal is to count unique words)
- Developed a custom UTF-8 words splitter for SPU (snippet processing unit)
- Allows parallel tokenization and counting at storage device

** Results of SQL access from parallel statistics code presented at CUG'2009.*

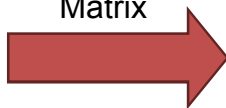
Data-Services Application

Interactive Visualization for Multilingual Document Clustering

Compute Nodes
(Trilinos Code)



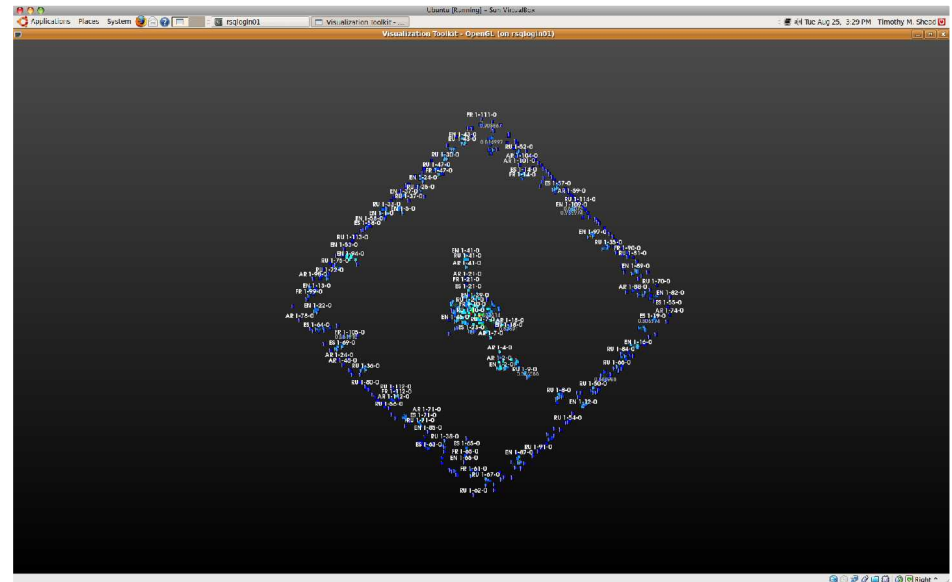
Similarity
Matrix



Service Nodes
(VTK Service)



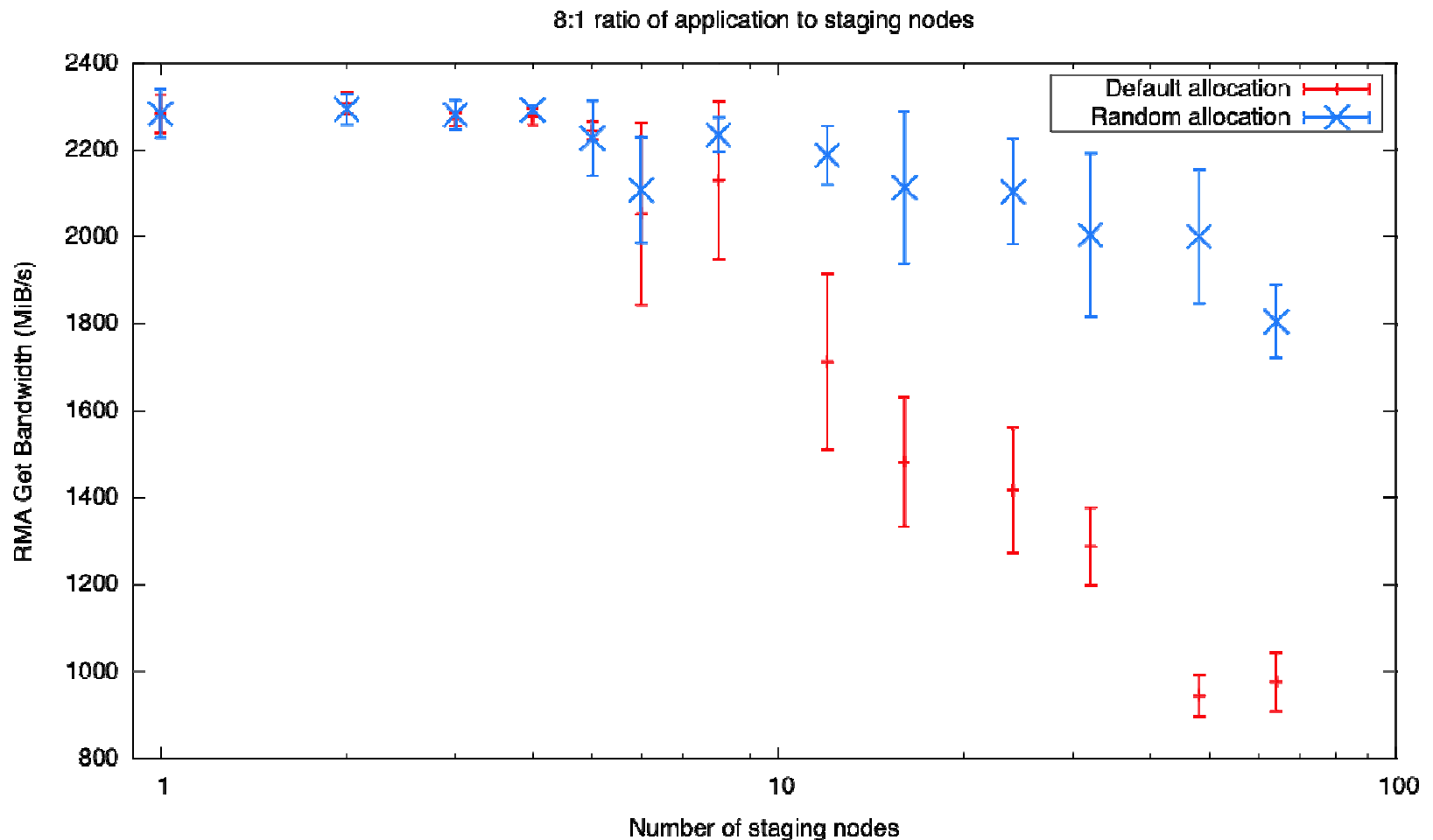
Titan
Visualization



Other Gaps to Fill

- System software support for data services
 - Support for dynamic allocation and reconfiguration
 - Data services: balanced workflow, reduce data movement, dynamic deployment
 - Smart placement (topologically aware scheduling)
 - Integrated support for NVRAM as a memory device
- Programming models
 - Standard approaches for integrating sim and analysis
 - Standard approaches for programming services (CPU, GPU, FPGA)
- Resilience
 - Storage-efficient app resilience is still a problem after 20+ years of research
 - Data service resilience: services use memory for transient data, how do we ensure resilience in such a model? We are working on this... let's talk again next year ;0)

Placement Issues for I/O Services



Summary and Staff

- **Trios Data Services reduce the impact of I/O on applications**
 - Reduce the “effective” I/O cost through data staging
 - Reduce amount of data written to storage (integrated analysis, data services)
- **Nessie provides an effective framework for developing services**
 - Client and server API, macros for XDR processing, utils for managing svcs
 - Supports most HPC interconnects (Seastar, Gemini, InfiniBand)
- **Trilinos provides a great research vehicle**
 - Common repository, testing support, broad distribution
- **Trios Data Services Development Team (and current assignment)**
 - Ron Oldfield: PI, CTH data service, Nessie development
 - Todd Kordenbrock: Nessie development, performance analysis
 - Gerald Lofstead: PnetCDF/Exodus service, transaction-based resilience
 - Craig Ulmer: Data-service APIs for accelerators (GPU, FPGA)
 - Ron Minnich: Protocol performance evaluations, Nessie BG/P support

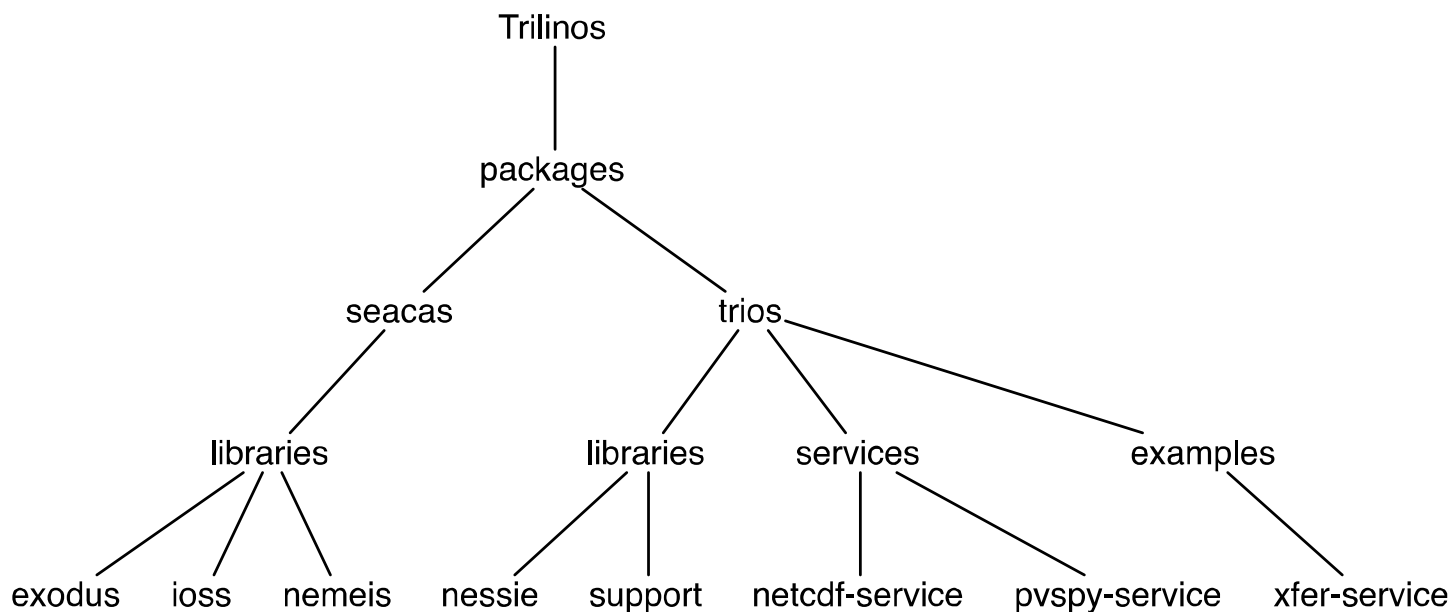
Extra Slides

Trilinos I/O Support (Trios)

- New Capability Area in Trilinos
 - Copyright assertion granted Oct. 2011
- Objectives
 - I/O Support for existing production codes
 - Exodus, Nemesis, IOSS
 - ***Vehicle for Open Source I/O R&D***
 - ***Trios Data Services***
- Benefits of Trilinos
 - Well-defined software-engineering framework
 - Broad distribution and access for I/O software developers
 - Increased opportunity for co-design with application developers and hardware vendors

Trios Status Update

Copyright assertion granted!



I/O Software spans two packages: SEACAS, Trios

Trios Status Update

Trios Package

- Libraries
 - Support (logger, timer, trace, ...)
 - Nessie (Portals, InfiniBand, LUC, Gemini)
 - CommSplitter (special for Cray XE6)
- Data Services
 - Transfer Service (example, tests, performance)
 - PnetCDF Staging Service
 - PVSpy Service (CTH in-transit analysis)
- Planned Work
 - Exodus staging service (like PnetCDF)
 - Transaction-based resilience for services
 - Accelerator-based services

SEACAS Package

- I/O Libraries (subpackages)
 - EXODUS II
 - NEMESIS
 - IOSS
- Planned Work
 - Eliminate “2-billion entities” problem
 - Native support for higher-order vector, tensor, and quaternion data
 - Store model hierarchy/part in the Exodus data model
 - Permit storing of transient data on the parts and assemblies
 - Exodus Support for changing topologies
 - Parallel I/O support (netcdf4, PnetCDF)
 - C++ and Python support

Early Inspiration for Data Services

We did this for Salvo Seismic Imaging (circa 1996)

Salvo's I/O Partition

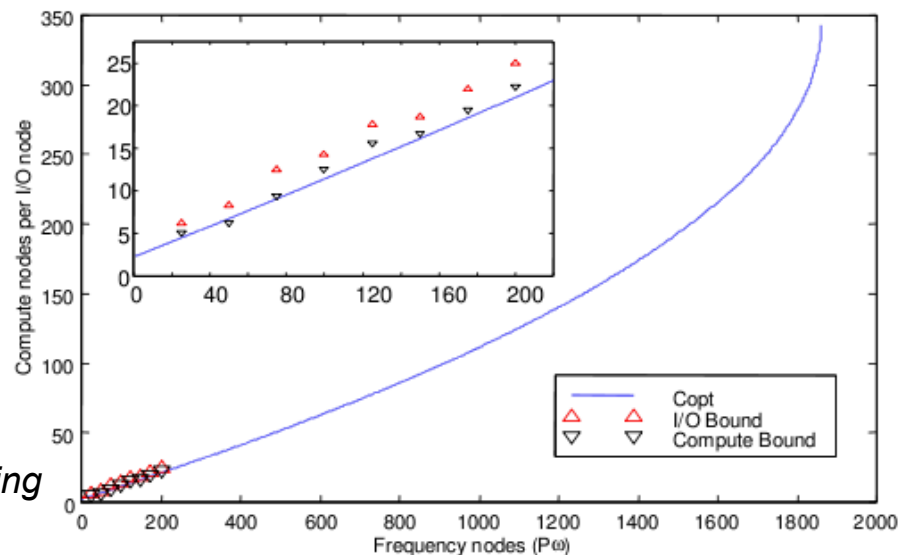
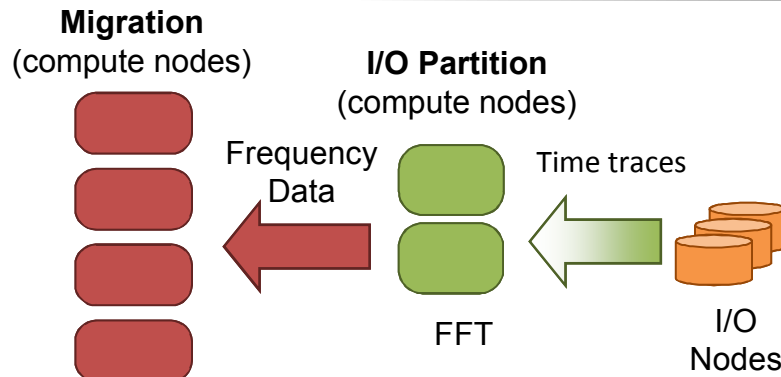
- Partition of application processors (used separate MPI Communicator for I/O)
- Used for FFT, I/O cache, and interpolation
- Async I/O allowed overlap of I/O and computation (pre-process next step)

Results

- +10% nodes led to +30% in performance
- Modeling I/O and compute costs helped find the right balance of compute and I/O nodes

Contacts: Ron Oldfield, Curtis Ober
{raoldfi,ccober}@sandia.gov

Oldfield, et al. Efficient parallel I/O in seismic imaging.
The International Journal of High Performance Computing Applications, 12(3), Fall 1998



Evaluating the Transfer Service

Gemini Interconnect

Comparison of Write Methods on Cielo

