SAND2012-0142P

# Supporting Integrated Data Services

## A New Challenge for High-Performance Computing

***Approved for Public Release: SAND2012-XXXXP***

Invited Talk
Argonne National Laboratories

Jan, 2012

***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.

# Some I/O Issues for Exascale

- Storage systems are the slowest, most fragile, part of an HPC system
  - Scaling to extreme client counts is challenging
  - POSIX semantics gets in the way, …
- 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
- One way to reduce pressure on the FS is to inject processing nodes
  1. Reduce the "effective" I/O cost through data staging
  2. Reduce amount of data written to storage (integrated analysis, data services)
  3. Present FS with fewer clients (IO forwarding)

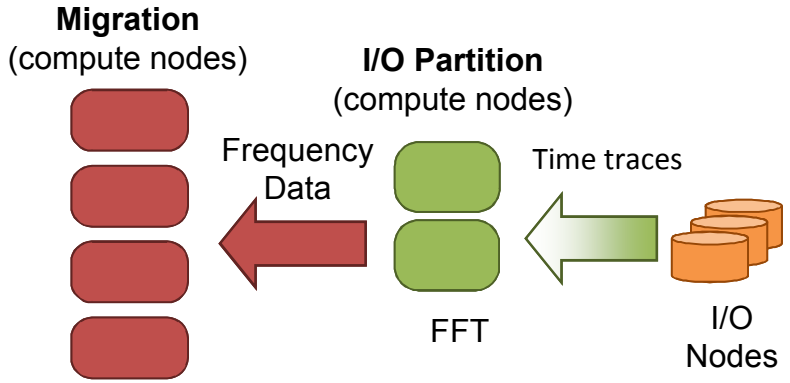
***We call this "Integrated Data Services"***

# I/O Processing for Seismic Imaging

## Our First Data Service (1996)

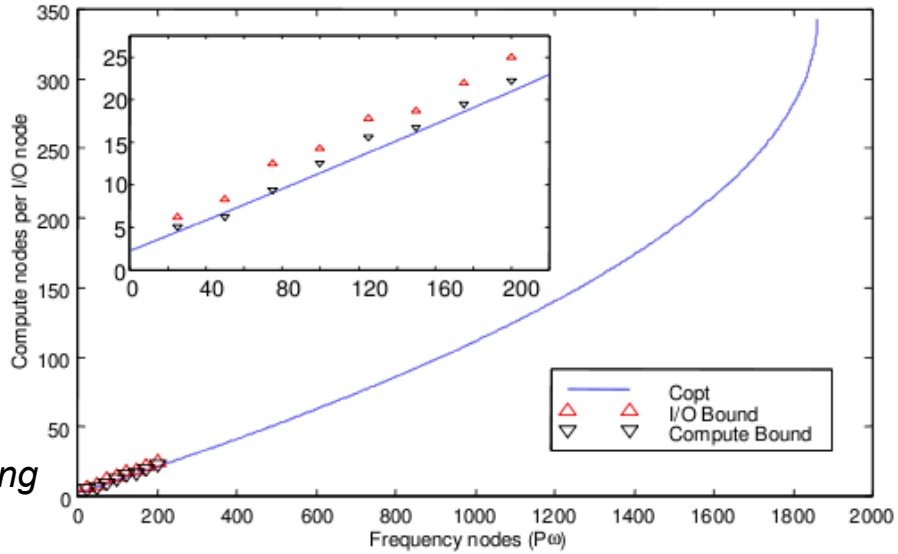
### Salvo's I/O Partition

- 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

# Everyone Jump on the Bandwagon…

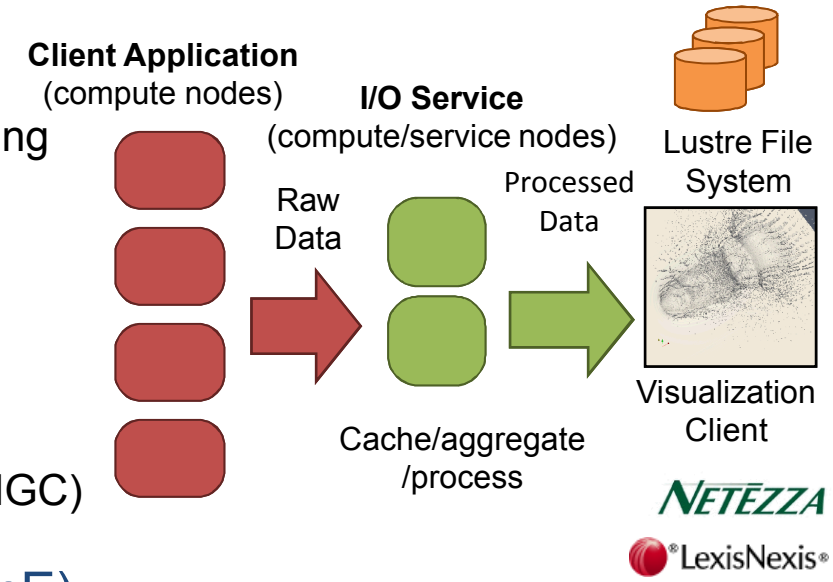
- Past Efforts
  - Active Storage/Networking (CMU, Duke, PNNL, Netezza,…)
  - Armada (Dartmouth)
  - DataCutter (Maryland, OSU)
- Current Efforts for HPC (no particular order)
  - ADIOS/DataStager/PreDatA (Ga Tech, ORNL)
  - DataSpaces (Rutgers)
  - Glean (ANL)
  - In-Situ Indexing (LBL)
  - I/O Delegation (NWU)
  - ParaView co-processing
  - VisIt remote vizualization

# NEtwork Scalable Service Interface (Nessie)

## Part of Trilinos I/O Support (Trios)

### Approach

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




### Application-Level I/O Services

- PnetCDF staging service
- CTH real-time analysis
- SQL Proxy (for NGC)
- Interactive sparse-matrix visualization (for NGC)

### Nessie (NEtwork Scalable Service InterfacE)

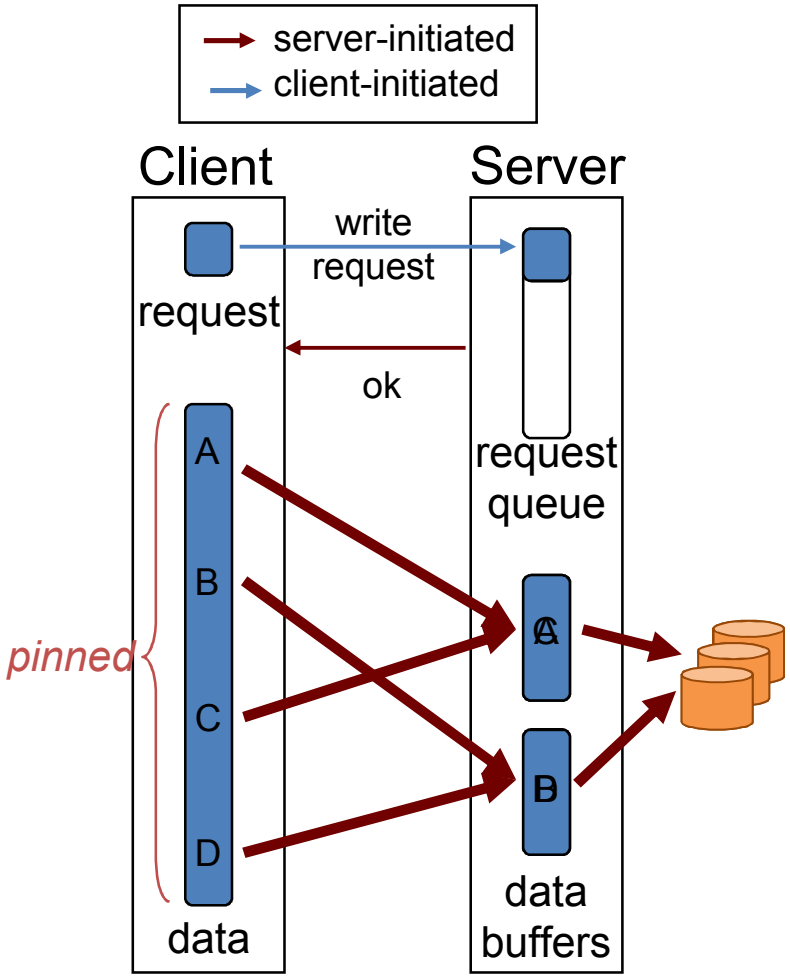
- Framework for developing data services
- Client and server libs, cmake macros, utilities
- Originally developed for lightweight file systems


***Currently in Trilinos Development Branch, release date in Jan/Feb 2012***

# 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





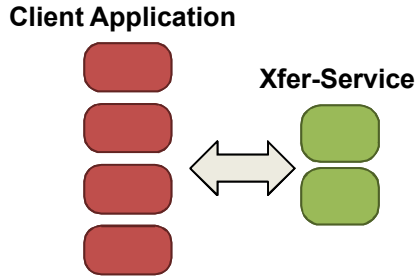

Nessie
NEtwork-Scalable Service InterfacE

Sandia National Laboratories

# 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




```
/* 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;
};

...
```



Nessie
NEtwork-Scalable Service InterfaceE

Sandia National Laboratories

# 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

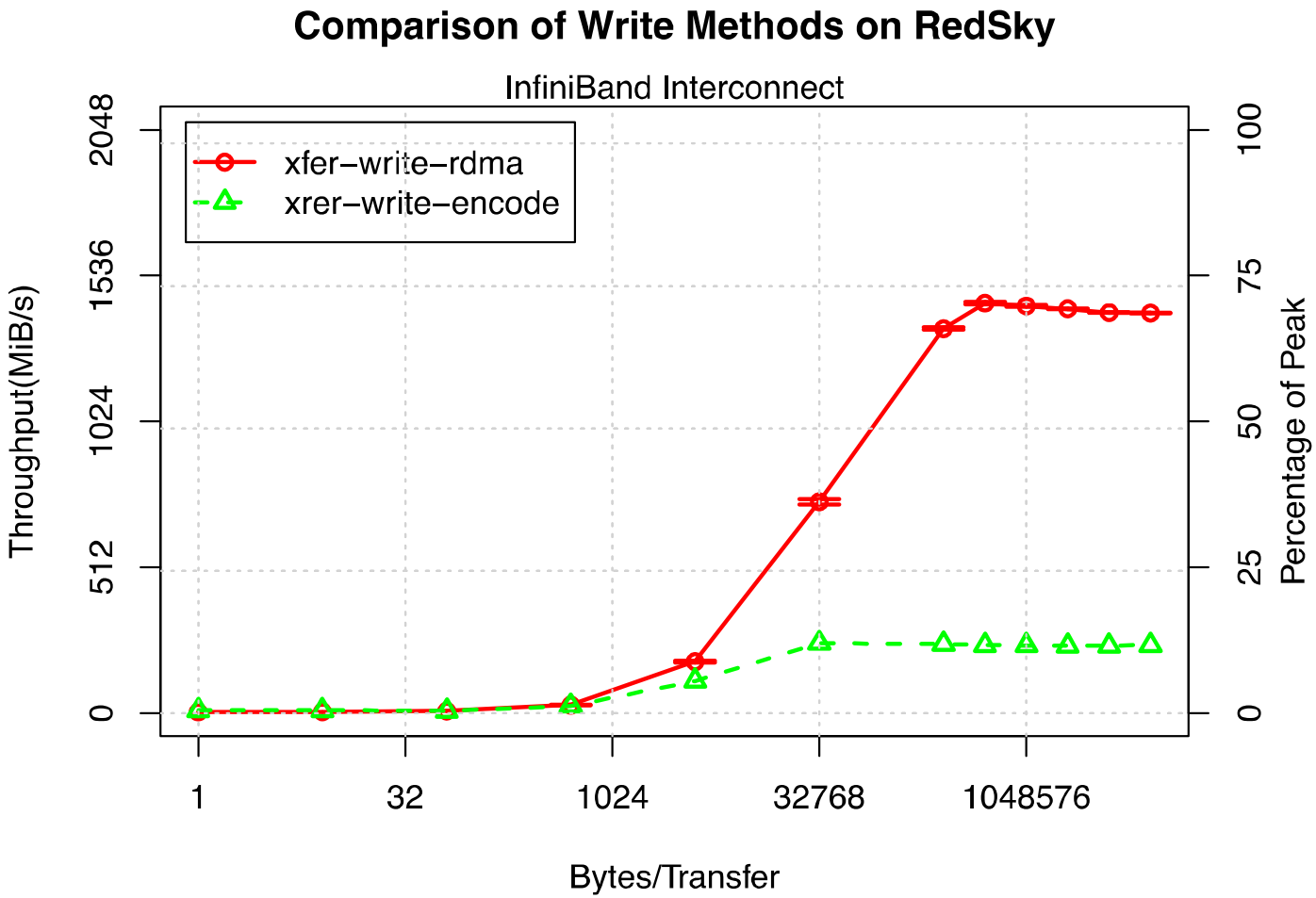
## Implementing the Server

- 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);
}
```

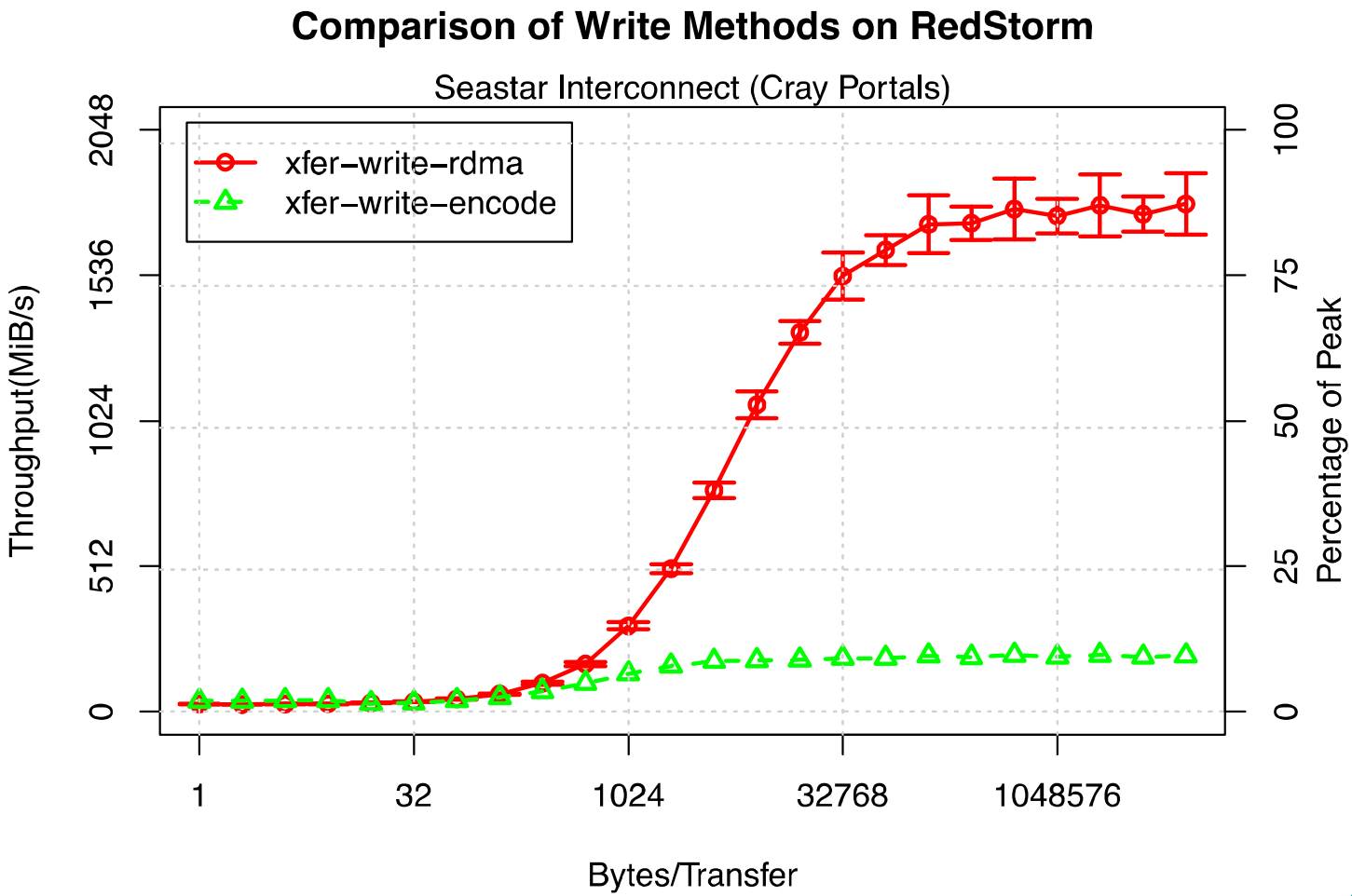
# Evaluating the Transfer Service

## InfiniBand Interconnect



# Evaluating the Transfer Service

## SeaStar Interconnect (Portals)



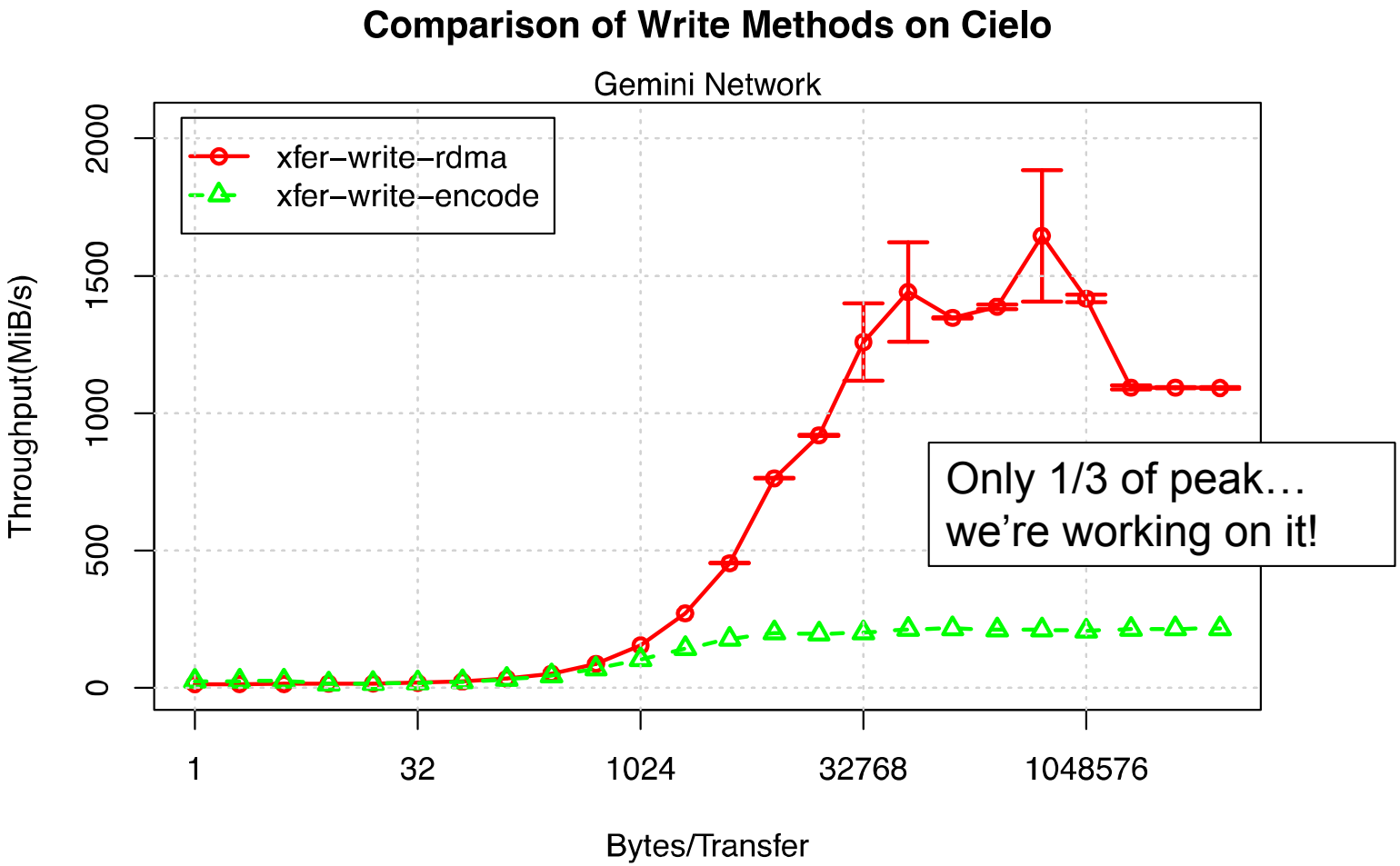



Nessie
NEtwork-Scalable Service InterfacE

Sandia National Laboratories

# Evaluating the Transfer Service

## Gemini Interconnect



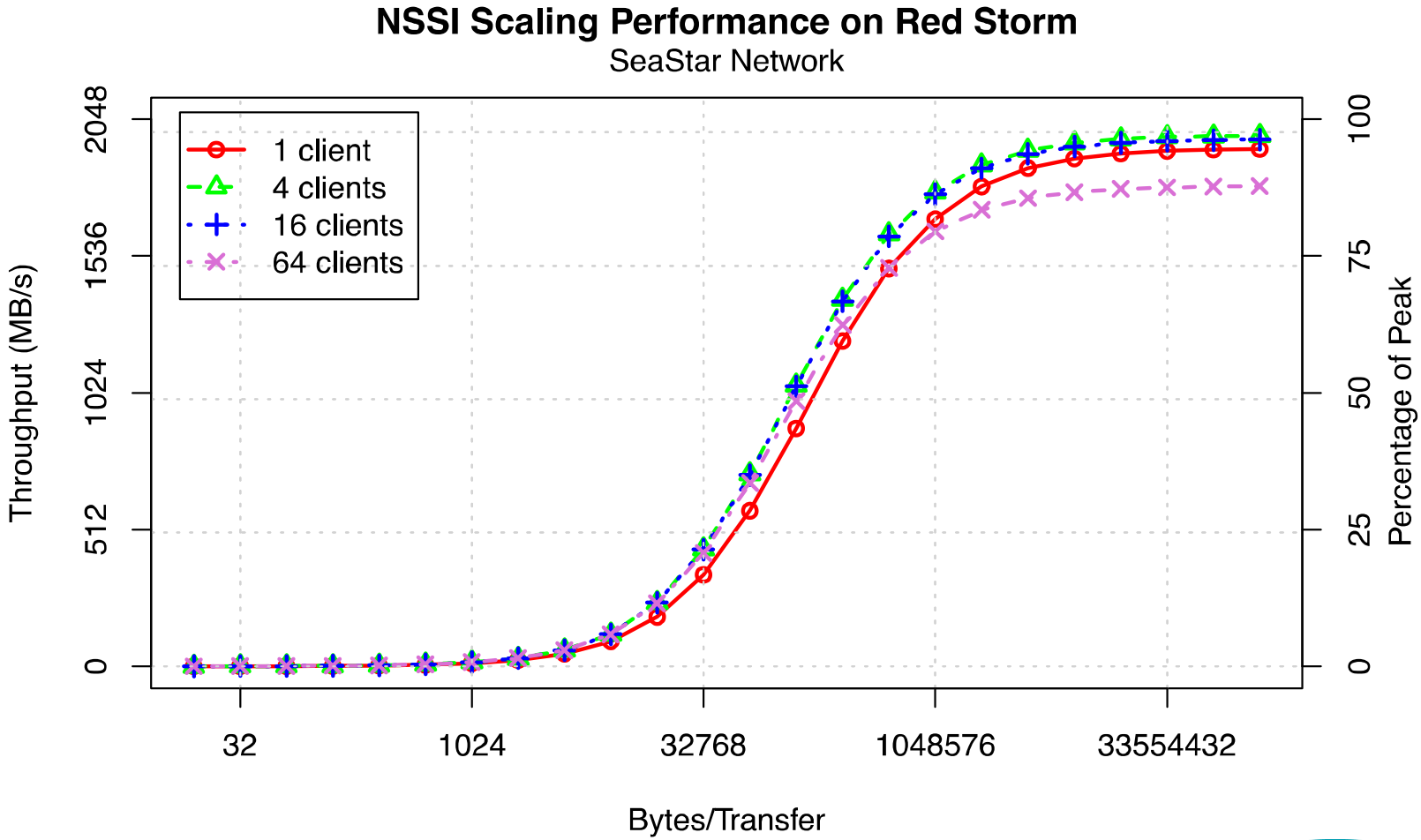



Nessie
NEtwork-Scalable Service InterfaceE

Sandia National Laboratories

# Evaluating the Transfer Service

## SeaStar Interconnect (Portals)






ASC

Nessie
NEtwork-Scalable Service InterfaceE

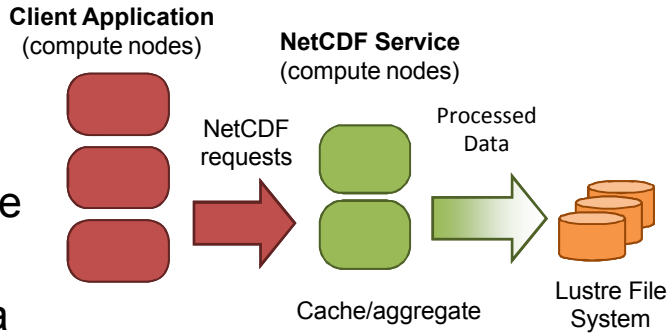
Sandia National Laboratories

# Scalable I/O Services

## NetCDF I/O Cache

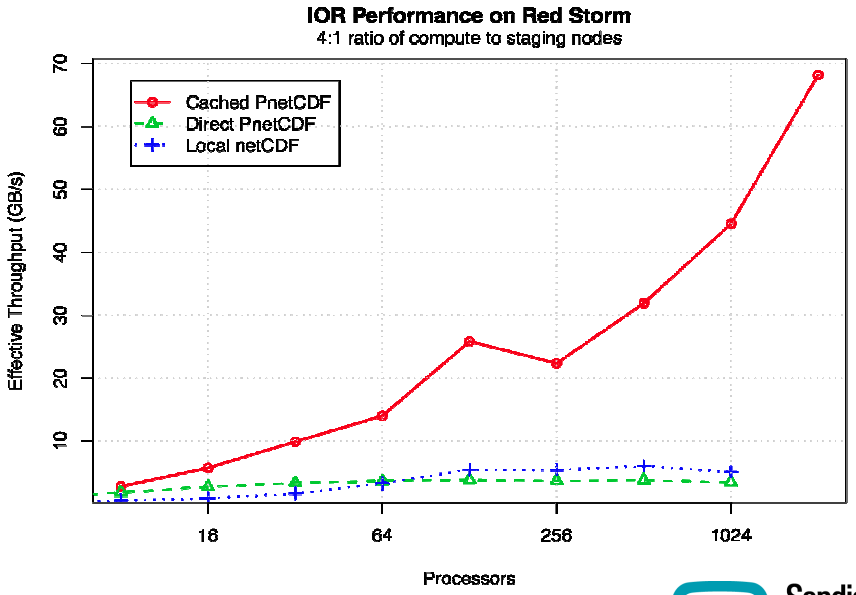
### Motivation

- 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



Nessie
NEtwork-Scalable Service InterfacE

Sandia National Laboratories

# Data Service Applications

## 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

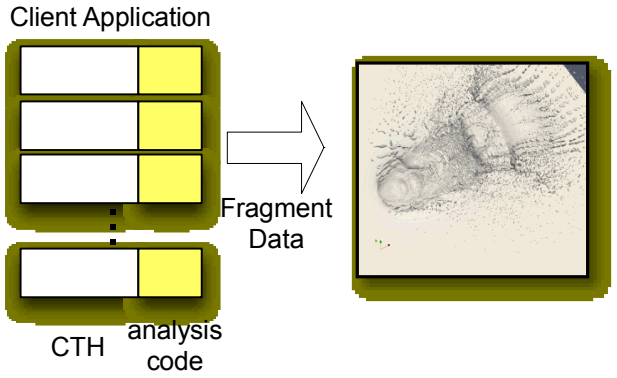
### CTH fragment detection service

- 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

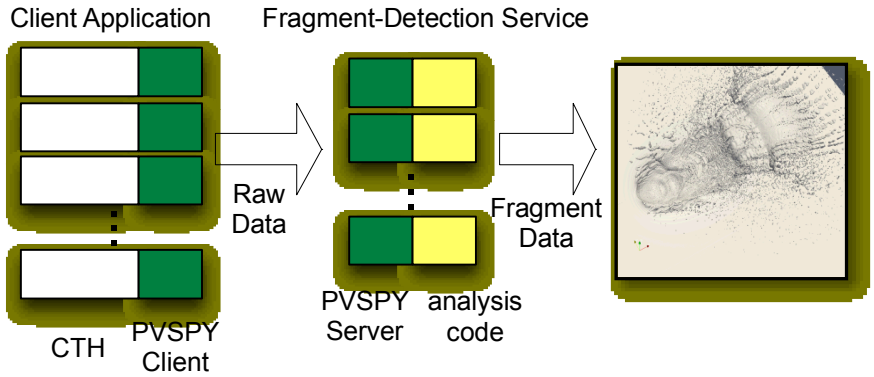
### Status and Ongoing Work

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

### In-Situ Fragment Detection




### In-Transit Fragment Detection






Nessie
NEtwork-Scalable Service InterfacE

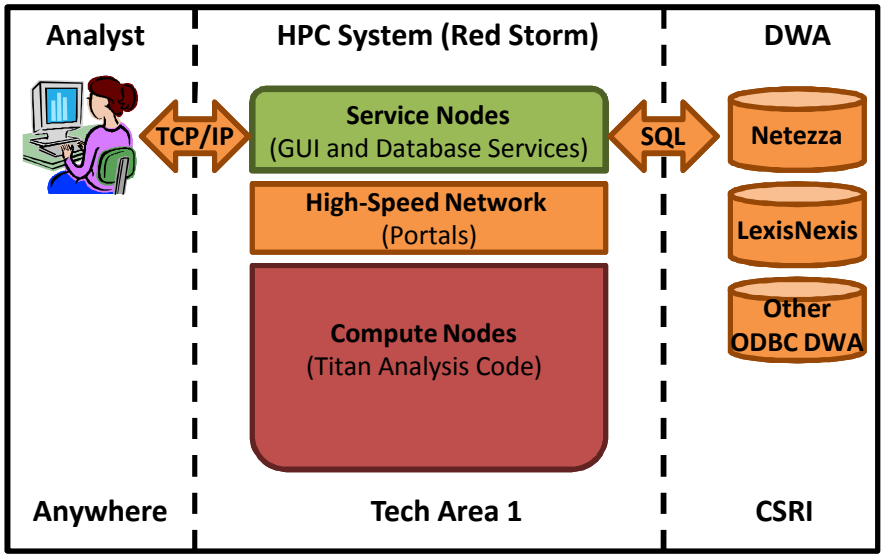
Sandia National Laboratories

# 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*

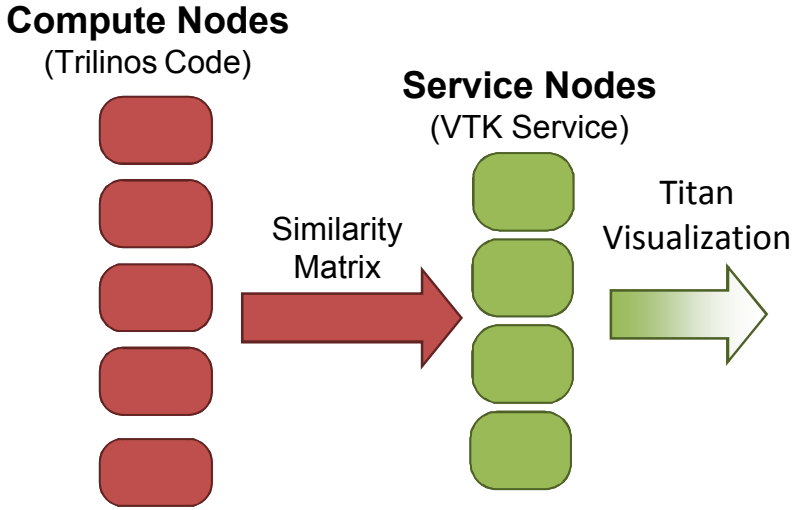
### Additional Modifications for Multilingual

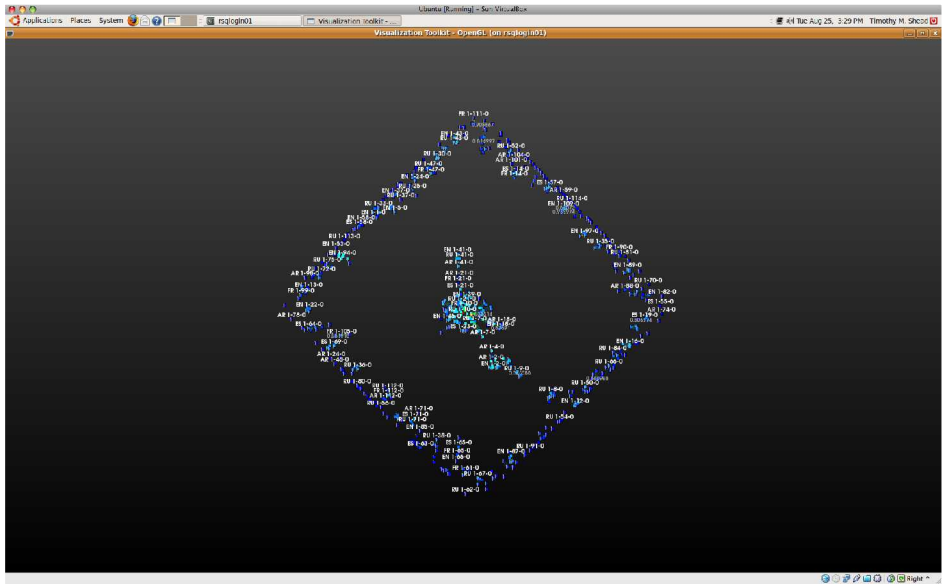
- Tokenization support on Netezza (goal is to count unique words)
- Developed a custom UTF-8 words splitter for SPU (snippet processing unit)
- Allows parallel tokenization and counting at storage device

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

# Data-Services Application

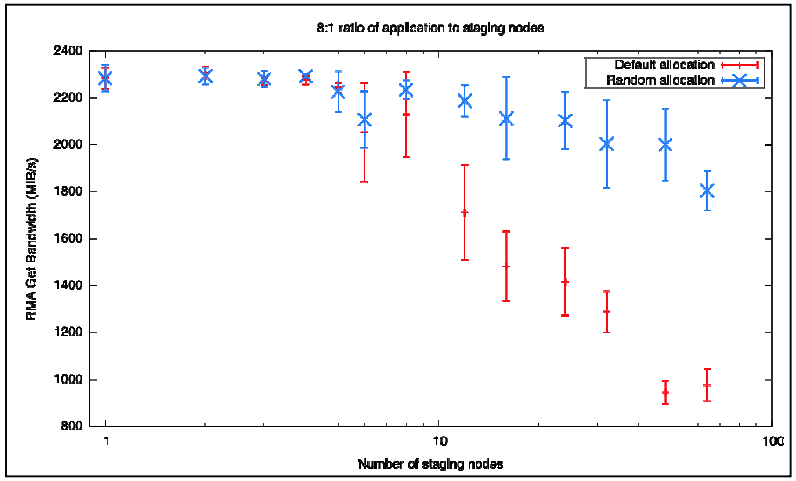
## Interactive Visualization for Multilingual Document Clustering



# So What's Missing?

- Two types of data services: application-level and system-level
  - Some system-level services already exist (e.g., file services)
  - The challenge is providing support for application-level services
- Scheduling: dynamic allocation, reconfiguration, and placement
  - Balance workflow, reduce data movement, on-demand services
  - The ability to control placement is important






ASC
Nessie
NEtwork-Scalable Service InterfacE
Sandia National Laboratories

# So What's Missing?

- Programming models
  - Standard API for inter-application communication (RDMA)
    - Portals, InfiniBand, Gemini, …
    - MPI-2 has this… kind of. Not many implementations.
  - Programming models for integrating analysis (co-processing)
  - High-level libraries for developing/integrating services (CPU, GPU, FPGA)

- Security models that support shared services
  - Cray XE6 doesn't even support private services




Sandia National Laboratories

# So What's Missing?

- Resilience
  - Storage-efficient app resilience is still a problem after 20+ years of research
  - Data services use memory for transient data, how do we ensure resilience in such a model?
  - We are exploring transaction-based methods
    - Goal is to provide assurances in multiple protection domains (e.g., the application, service 1, service 2,…)
    - Lofstead can talk all about this…

# Summary

- Data Services are coming!
  - Nearly every Lab has their own approach (Nessie, Gleam, ADIOS, …)
  - Scheduling, programming models, security, and resilience need to better support data services. Lots of work to do!
- 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



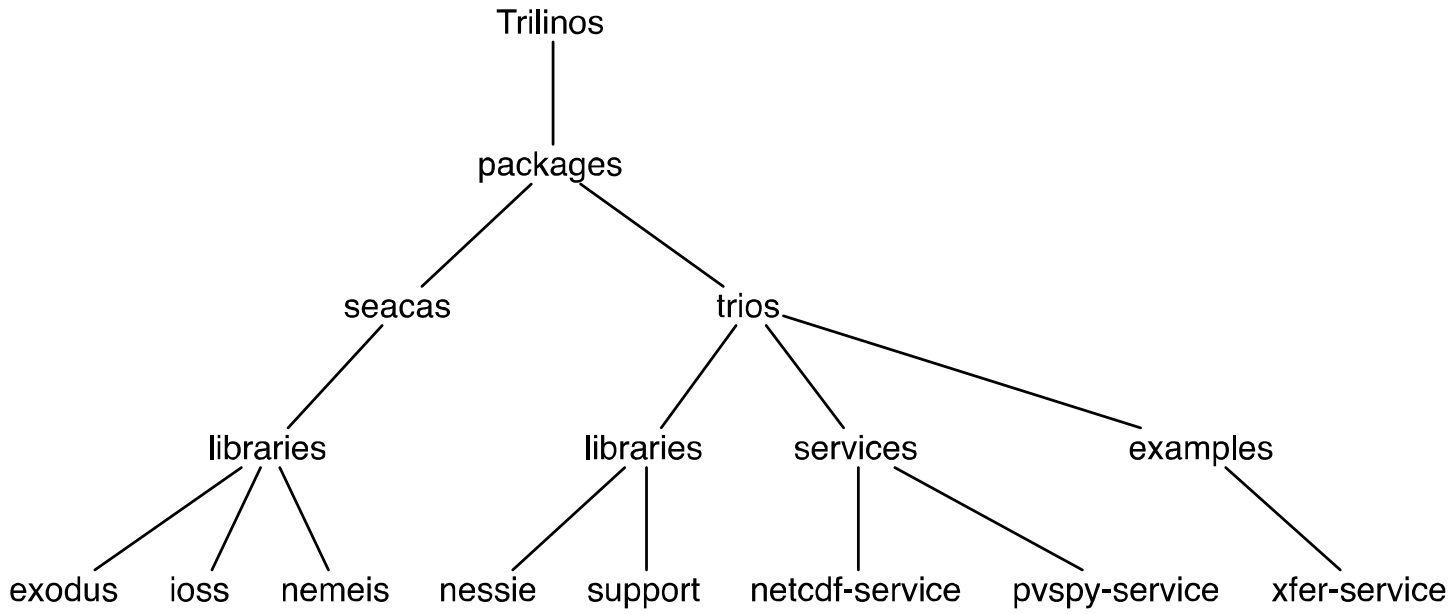

Sandia National Laboratories

# 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