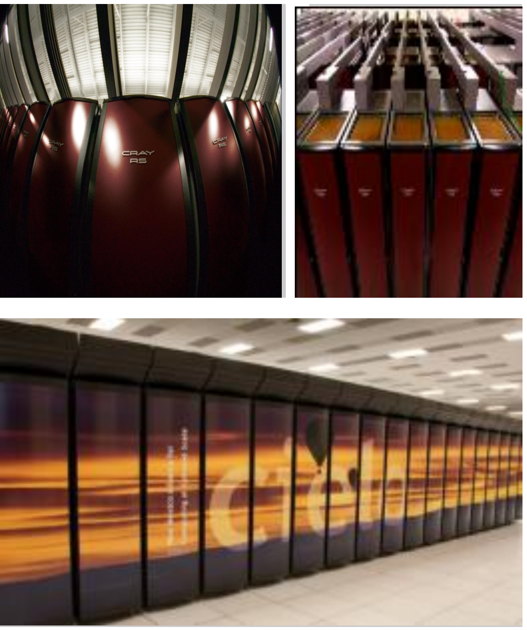


# Developing Integrated Data Services for Cray Systems with a Gemini Interconnect

**Ron A. Oldfield**
**Scalable System Software**
**Sandia National Laboratories**
**Albuquerque, NM, USA**
**raoldfi@sandia.gov**

**Cray User Group Meeting**

**May 3, 2012**

*Exceptional service in the national interest*

# 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. Manage "bursts" of data with network caching (also called burst buffer)
  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"***

## Everyone Jump on the Bandwagon…

We're not the only ones doing data services

- Past Efforts
  - Active Storage/Networking (CMU, Duke, PNNL, Netezza,…)
  - Armada (Dartmouth)
  - DataCutter (Maryland, OSU)
  - I/O Partition for seismic imaging (Sandia)
- 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

# I/O Processing for Seismic Imaging

## Our First Data Service (1996)

Salvo's I/O Partition

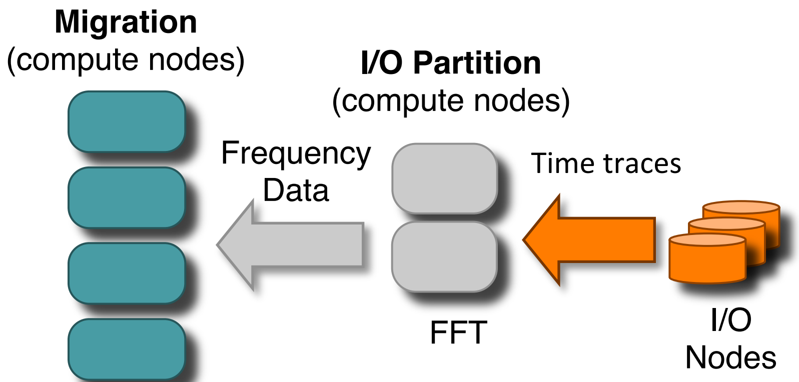
- Partition of application processors (used separate MPI Communicator for I/O)
- Used for FFT, I/O cache, and interpolation
- Async I/O allowed overlap of I/O and computation (pre-process next step)

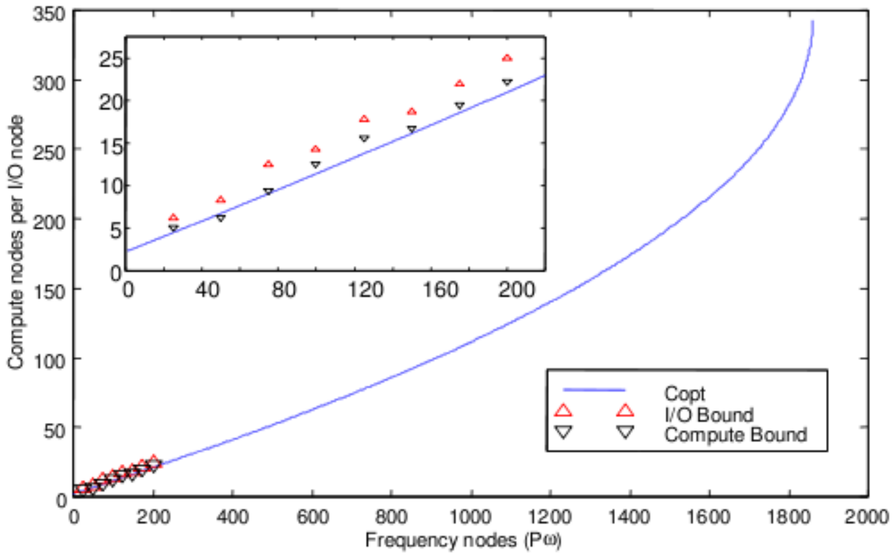
Results

- +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





# NEtwork Scalable Service Interface (Nessie)

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

### Approach

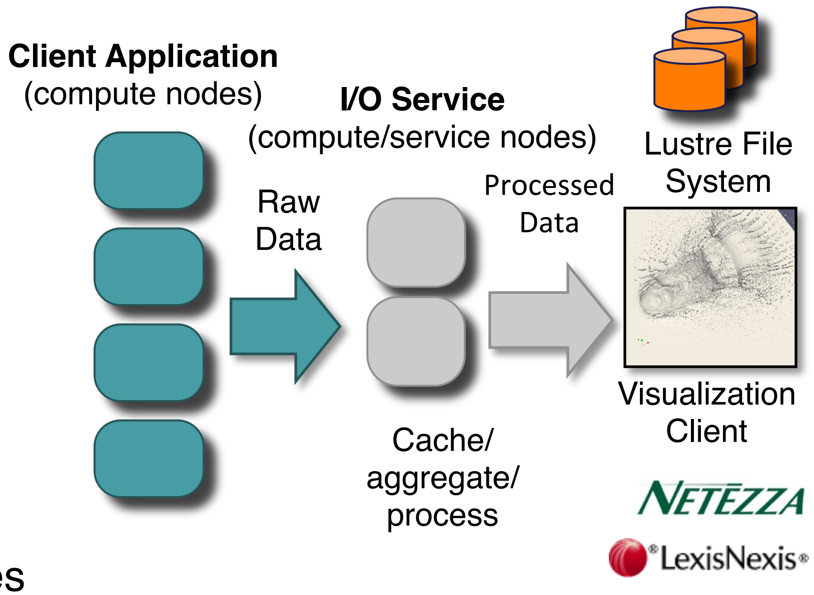
- Leverage available compute/service node resources for I/O caching and data processing

### Application-Level I/O Services

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

### Nessie

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

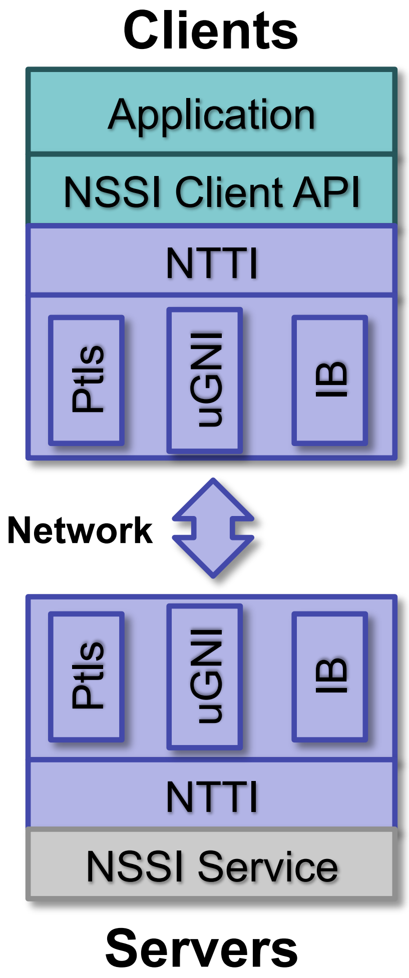


NEtwork-Scalable Service InterfacE

# Nessie Network Transport Layer (NNTP)

An abstract API for RDMA-based interconnects

- Provides portable methods for inter-application communication
- Basic functionality
  - Initialize/close the interface
  - Connect/disconnect to a peer
  - Register/deregister memory
  - Transport data (put, get, wait)
- Supported Interconnects
  - Seastar (via Portals3)
  - InfiniBand
  - Gemini (details in the paper)



## Issues Unique to Cray XE

- Communication Domains prevent inter-application communication*
  - Each process in a job shares a *protection tag* (ptag)
  - Each job is assigned a unique ptag by ALPS
  - Peers with different ptags are not allowed to communcate
- We overcame this limitation by running in MPMD mode
  - Jobs chained together share a global MPI_COMM_WORLD (problematic for legacy applications).
  - We developed a CommSplitter library that splits comms using the MPI profiling interface
- Example

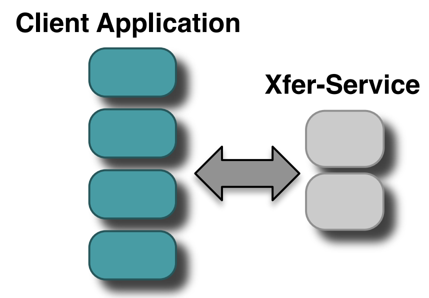
```
% aprun -n 10 xfer-service [opts] : -n 1000 xfer-client [opts]
```

** Cray recently addressed this issue in a new release of GNI and DMAPP APIs. See http://docs.cray.com/books/S-2446-4003/S-2446-4003.pdf for details*

# Example: A Simple Transfer Service

## Trilinos/packages/trios/examples/xfer-service

- Used to test Nessie API
  - ***xfer_write_rdma***: server pulls raw data using RDMA get
  - ***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;
};

...
```

# 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

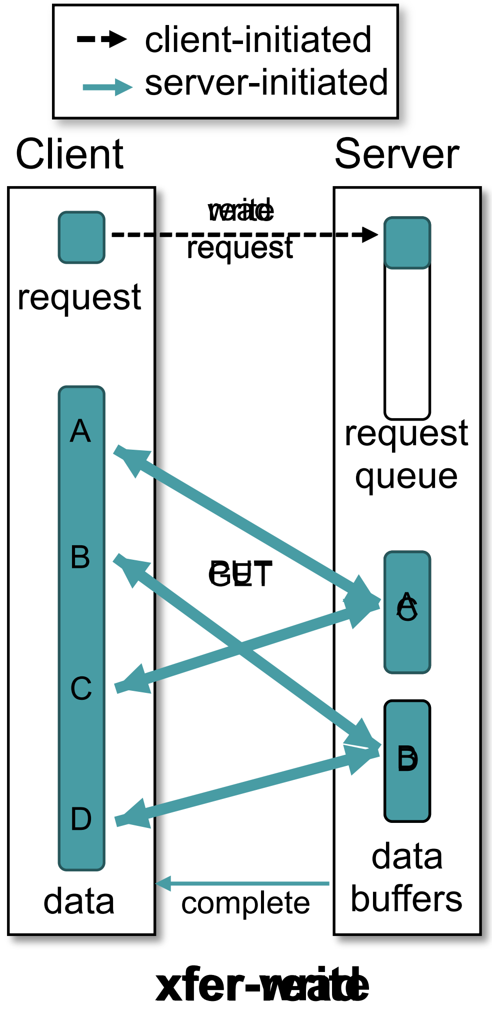
## Implementing the Server

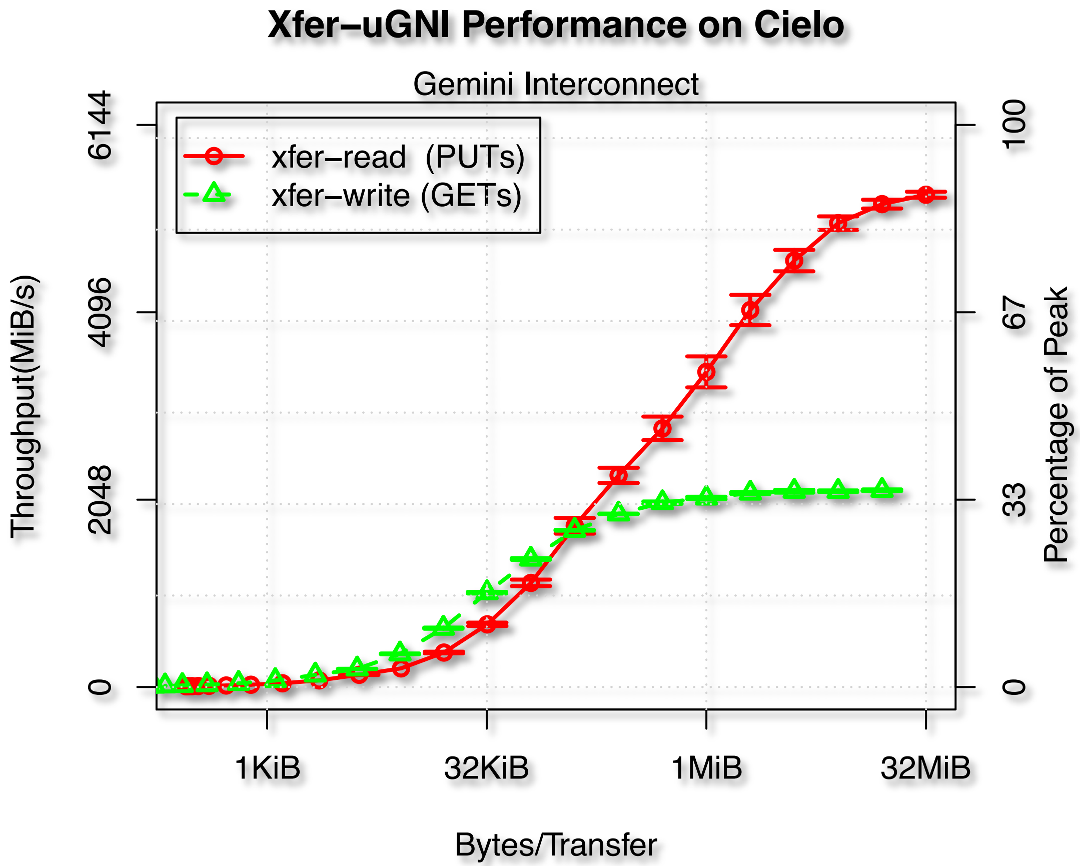
- Implement server stubs
  - Using standard stub args
  - For **xfer_write_rdma_srvr**, the server pulls data from client
- Implement server executable
  - Initialize Nessie
  - Register server stubs/callbacks
  - Start the server thread(s)

```
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);
}
```

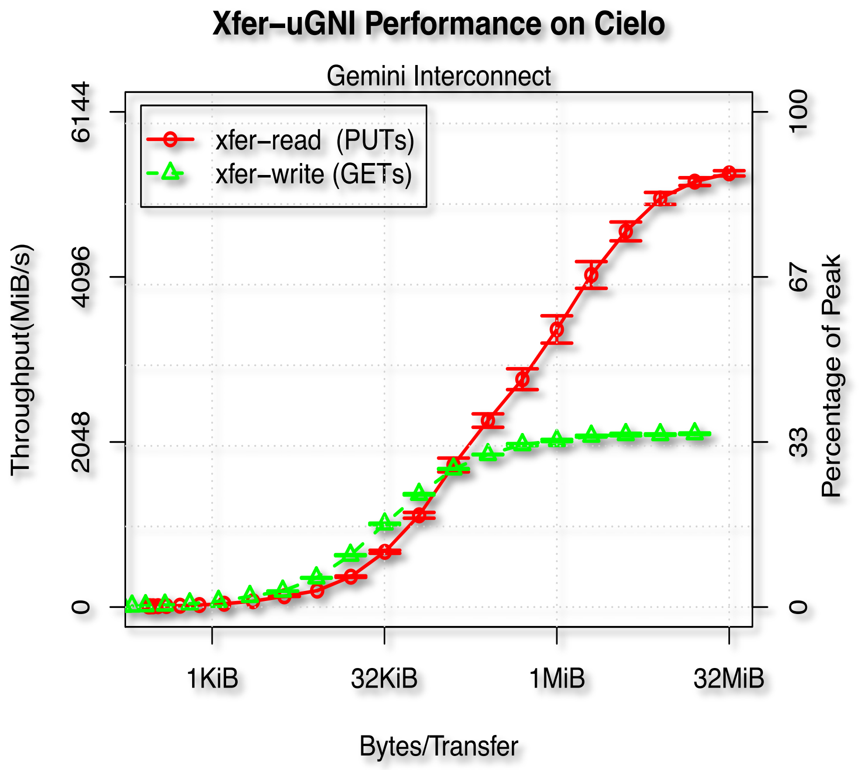
# Transfer Service Evaluation:

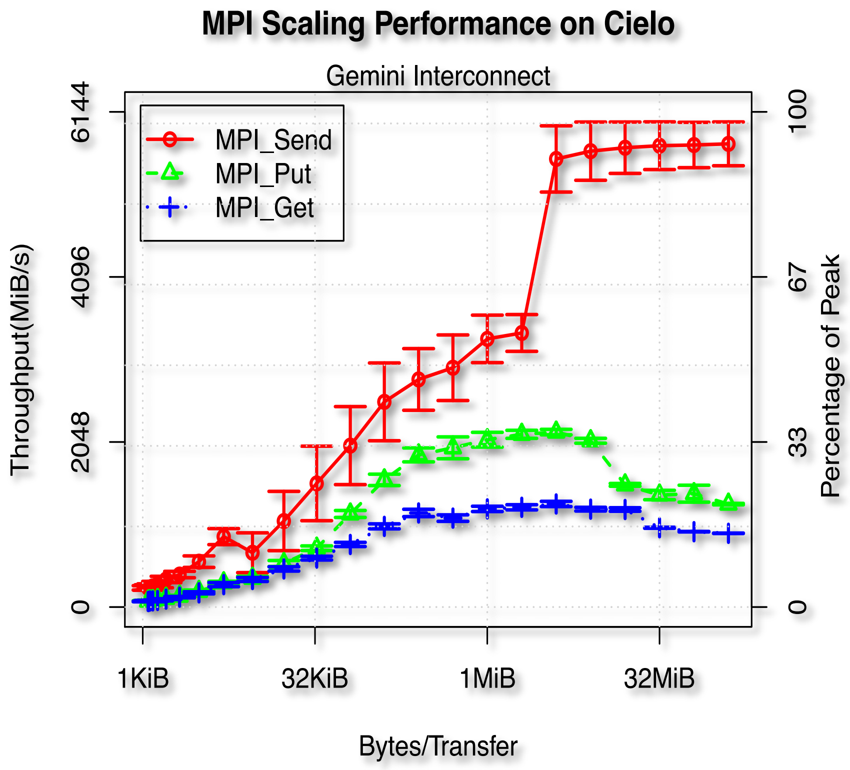
## Read/Write performance





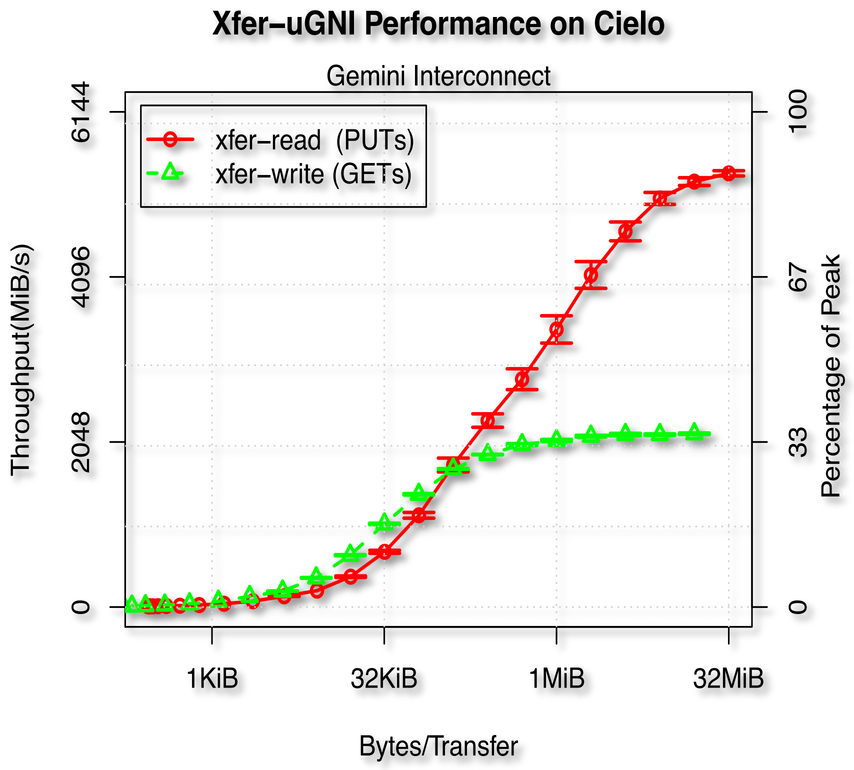
# Transfer Service Evaluation:
## Comparison to MPI

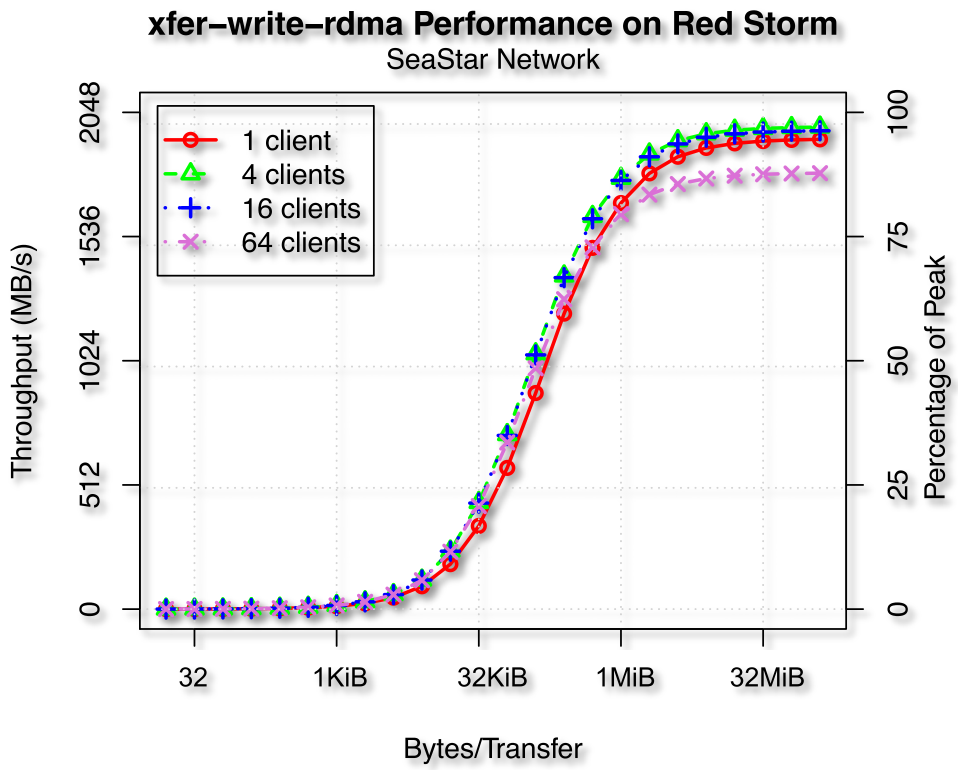




# Transfer Service Evaluation:

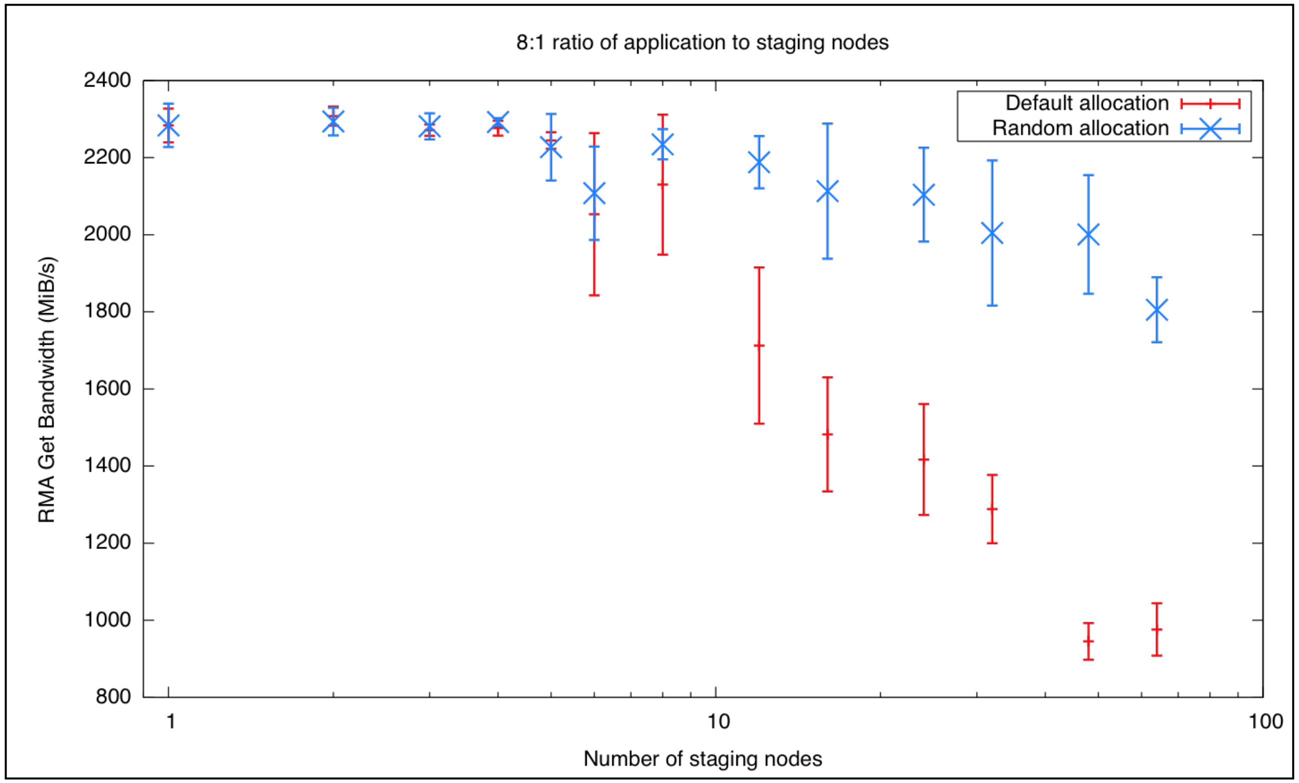
## Comparison to RedStorm





# So What's Missing?

- Scheduling
  - dynamic allocation, reconfiguration, and placement
  - Balance workflow, reduce data movement, on-demand services
  - The ability to control placement is important

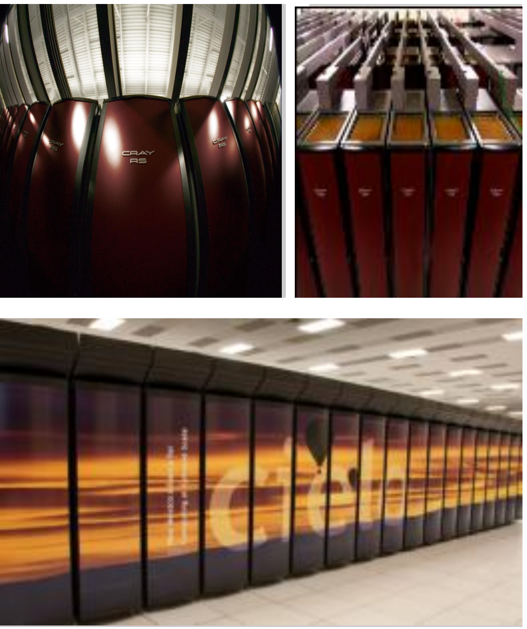


# So What's Missing?

- 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)
- 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?  Transactions?

## Summary

- Data Services are here!
  - Nearly every Lab has their own "ad-hoc" approach (Nessie, Glean, DataStager, …)
  - Scheduling, programming models, 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, IBM)
- Cray XE6 services were challenging
  - Used MPMD mode to overcome ptag/cookie issue
  - CommSplitter lib logically splits MPI communicators among apps/services
  - Latest version of uGNI allows inter-application comm (apps can share ptag/cookie)
  - Still need to address GET performance (only 1/3 of peak)



Sandia National Laboratories

*Exceptional service in the national interest*

# Developing Integrated Data Services for Cray Systems with a Gemini Interconnect

**Ron A. Oldfield**
**Scalable System Software**
**Sandia National Laboratories**
**Albuquerque, NM, USA**
**raoldfi@sandia.gov**

**Cray User Group Meeting**

**May 3, 2012**

U.S. DEPARTMENT OF ENERGY

NNSA National Nuclear Security Administration