Experiences Applying Data Staging Technology in Unconventional Ways

Jay Lofstead*, Ron Oldfield*, Todd Kordenbrock†
*CSRI, Sandia National Labs.
†Hewlett-Packard Company

***Abstract*—Several efforts have shown the potential of using additional compute-area resources to enhance the IO path to storage. Efforts like data staging, IO forwarding, and similar techniques can accelerate IO performance and reduce the impact of IO time to a compute application. Hybrid staging enhanced this path by adding processing functionality to locations along the data path to storage. While these efforts have been effective, they have taken a somewhat limited view of the potential benefits using some additional compute resources can offer both to enhance a compute application as well as to offering a way to exploit HPC-style resources for non-traditional tasks.**

Over the last few years, we have been experimenting with the potential for other sorts of activities using a staging style approach to add or enable new functionality. The efforts in this area have yielded a collection of small projects that yield some insights into both the potential and limitations of this approach for both achieving exascale computing and for enabling alternative uses for HPC resources.

I. Introduction

The growth of commodity clusters and Big Data have yielded a desire to use the compute capacity of these machines to try to address massive data analysis challenges more efficiently. Connecting these resources with the users to perform analysis tasks requires a bit of a different infrastructure. End user machines will have to have connections into the cluster compute area with a two-way communication channel for data movement and processing instructions. These sorts of connections are typically difficult. Direct connections are prevented to avoid the jitter involved with the additional network traffic.

For both traditional data staging and these other needs outlined above, using a portion of a cluster can either enable different processing or even offer a connection between the HPC resources and other systems creating a hybrid offering new functionality. With these scenarios in mind, we chose to explore different sorts of uses for these resources and the supercomputer.

To make the connection, we used the Network Scalable Services Interface (NSSI) [1] developed by our group as part of the Lightweight File Systems project (LWFS) [2]. NSSI offers an rpc-style mechanism to simplify connecting resources on Cray SeaStar and Portals [3], [4], InfiniBand [5], and the new Cray Gemini [6] interconnects. Our strategy was to re-implement the communication API used in the application, relink, and then the new functionality would be available. For scenarios that work in the other direction, the network transport-level support offers an opportunity to connect outside accessible nodes with the compute area.

The examples we demonstrate start with a more traditional staging example first described at PDSW @ SC 11 [7], but with many of the questions left open at that time answered. In this case, rather than just outsourcing the data movement to a staging area, as many existing data staging examples do, we also moved the communication phase of the IO operation as well. Next, we look at enabling an informatics application by running data analysis operations in the compute area of an HPC resource. The challenge with this scenario is connecting to the backend data store to both pull data into the compute area and also push results back into the data store. We developed a service affording database connectivity from within the compute area to areas outside the HPC machine. We then show two examples that use the HPC resource to support an external application. We evaluate both multi-lingual document clustering and streaming analysis of network traffic data. In the document clustering case, we bring in the corpus and process in parallel to support an end user's interactive use. Finally, the streaming analysis of network traffic data uses the HPC resource to do new real-time analysis of data about network streams.

In all of these cases, we leverage the core idea of data staging, i.e., the use of additional resources to enhance some core computation through the use of some connecting code, in ways very different from existing work. Some custom solutions somewhat similar to these may have been built historically for custom hardware and application platforms, but none have aimed to use these general purpose resources for these different scenarios.

The remainder of this paper is organized as follows. Section II presents a short overview of the related work. We introduce the NSSI protocol central to all of these examples in Section III. We next present the three examples motivating unconventional use of data staging areas in Section IV. Section V offers conclusions and future work.

II. Related Work

Data staging, or more generally using a small amount of additional resources to accelerate IO in some fashion, has been in use for many years. We first posited this approach in 1996 to help with imaging for a Seismic modeling application [8]. In that work, by adding 10% more nodes, a 30% performance improvement was achieved. It hosted FFT operations to process the data and used asynchronous IO to overlap the IO with computation.

More recently, the use of an IO Delegate and Caching System [9] showed the performance gains for collecting the IO requests at the IO layer. This system cached data for future requests and aggregated small requests into larger ones to improve performance.

Similar to this effort is the IO Forwarding Scalability Layer [10]. It added some scheduling of IO requests to try to manage the impact of IO operations on the file system. These optimizations focused on the path between the staging nodes and the file system.

The DataTap [11] system and the follow-on work has focused on how to effectively leverage asynchronous IO to data staging areas with the observation of interference effects that IO-related data movement may have with application-related communication tasks. This work was motivated by the observation that naive use of asynchronous IO can *increase* the total run time of an application by as much as 30% rather than reducing the total time by eliminating nearly all of the time previously spent performing IO.

More richly, the PreDatA [12] project has demonstrated that hosting functionality along this IO path has different performance characteristics depending on where the operation is placed and the kind of operation. It also shows that using these sorts of operations in the IO path can still improve the total wall clock time while massaging data into a more desirable form when it reaches disk, even when accounting for the additional resources used for the data staging services.

The DataSpaces [13] project has focused on using data staging as a way to perform code coupling operations. It has focused on using asynchronous IO to move data into a staging area and then having a different application retrieve some portion of that data at a future time according to its need.

A recent effort called Glean [14] from Argonne is a start towards both accelerating IO performance and integrating functionality, such as analysis routines, at the right place transparently. It is very similar to PreDatA, but extends the location of operations to potentially beyond the current machine.

All of these efforts have been very strictly focused on using a data staging approach as a way to accelerate the IO path while offering some level of additional functionality. None of these efforts have attempted to use the idea of adding resources for alternative purposes, along with a transparent interface, to do different sorts of operations and integrations.

III. The Network Scalable Service Interface

NSSI offers an RPC-like interface that can operate over a variety of network layers affording the opportunity to communicate between applications within the compute area of an HPC resource and between an HPC application in the compute area and a service node that has access to resources external to the HPC machine. This connectivity opportunity affords the different examples demonstrated in this paper. Rather than require replacing an existing API with a new API or operating at a lower layer introducing a loss of opportunities and information, NSSI uses rely on maintaining the API and replacing the implementation with link-time compatibility. Re-implementing the API does have limitations, but it has proven to offer a considerable amount of functionality. An illustration of how this replacement would work is in Figure 1.

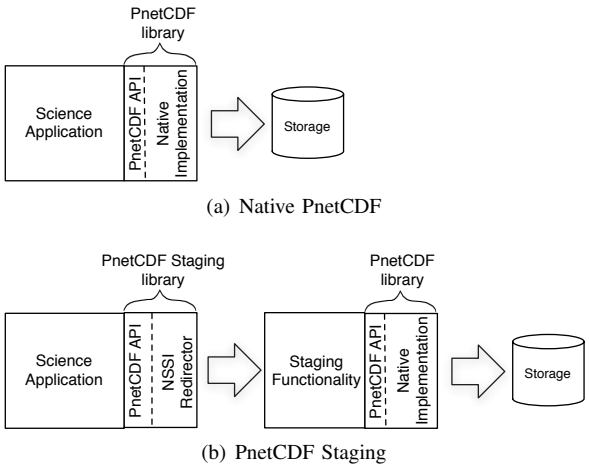

Fig. 1. Example of How NSSI is Incorporated With PnetCDF

Other uses of NSSI, including some of those in this paper use a slightly different approach, but still follow the same spirit of finding a way to minimally insert a new implementation that affords connecting with different resources, potentially external to the current machine.

IV. Unconventional Data Staging Examples

We outline three different categories of unconventional uses of data staging technology. These approaches expand the functionality of HPC resources and incorporate HPC resources into workflows that typically cannot leverage these resources. This connection is typically due to both the limited external interface between the compute area of the machine and the external environment and the lack of deployable algorithms into the environment due to this limited accessibility.

Experimental Setup The experiments are performed on a variety of different machines. We first describe the machines used for each experiment followed by the specifications for these machines. Two of these machines, JaguarPF and Red Storm have since been decommissioned.

For the first example of moving communication requirements along with data, the initial tests are performed on JaguarPF at Oak Ridge National Laboratory. The subsequent tests are performed on RedSky capacity machine at Sandia National Laboratories.

For the second example of informatics on HPC resources, the Red Storm machine and a Netezza appliance are employed.

For the third example of streaming data analysis, Red Storm is employed.

JaguarPF was a Cray XT5 containing 18,688 compute nodes in addition to dedicated login/service nodes. Each compute node contains dual hex-core AMD Opteron 2435 (Istanbul) processors running at 2.6GHz, 16GiB of DDR2-800 memory, and a SeaStar 2+ router. The SeaStar 2+ routers are connected in a 3D torus topology for scalability. The resulting partition contains 224,256 processing cores, more than 300TiB of memory, over 6 PB of disk space, and a peak performance of 2.3 petaflop/s. For all tests, Spider, the ORNL shared

scratch space Lustre file system, is employed. The peak IO performance for Spider's widow2 or widow3 partition used in this evaluation from JaguarPF is around 60 GiB/sec for writing in parallel to the 336 storage targets.

Red Storm was a Cray XT3 located at Sandia. At the time of testing, Red Storm had 12960 dual-core compute nodes. The compute nodes are arranged in a regular three-dimensional grid, connected with a hypertorus topology. Each node has an interconnect with a custom Cray SeaStar networking chip and a dedicated PowerPC chip. The interconnect is coupled to the processor using a HyperTransport link that has a theoretical (excluding wire protocol overhead) bandwidth of 2.8GB/s [15]. Each of the six links from each node can support 2.5GB/s, after protocol overheads. Low-level software access to the interconnects is provided through the Portals library [4], which provides a connectionless RDMA-based interface.

RedSky is an unclassified capacity machine. It is a Sun Blade center with Sun X6275 blades containing 2823 nodes running Intel Xeon 5570 processors (8 cores each) with 12 GB of RAM per node and QDR InfiniBand arranged in a 3-D toroidal mesh as the communication fabric.

A. Moving Communication Along With Data

Our first example is most simialr to typical data staging, but moved more than just the IO operations. In our previous workshop paper [7], we showed the initial results suggesting the advantage of shifting more than just the IO operation to the staging area. The base results from the JaguarPF machine at Oak Ridge are reproduced in Figure 2 for reference with the discussion below. As is our pattern with NSSI, the PnetCDF API is re-implemented to move the call and associated data to the staging area before any coordination among participating processes occur. This shifts the communication phase of two-phase IO to the staging area. Traditionally, systems like the IO Forwarding Scalability Layer just shift the actual IO system calls leaving the communication phase on the compute nodes. In the results in Figure 2, a single staging node with 12 processes (1 per core) is used. The two techniques of either caching individual IO requests or aggregating IO requests into larger requests are consistently better performance by a percentage that exceeds the compute job cost of 1 additional node. The direct calls simply pushes the call to the staging area and executes it synchronously. The lack of a real difference between the aggregating and the caching performance was left as an open question. Further experimentation, discussed below, has revealed the reason that aggregating to form fewer, larger IO calls did not yield better performance than simply caching requests and executing them in a batch. For all of the tests performed in this section, at least four repetitions of each test is performed and the best time for each test is selected.

The additional experimentation is done on the RedSky machine at Sandia yielding the results in Figures 3, 4, 5 and 6. Figures 3 and 5 are comparable with JaguarPF in that they also use a single staging server while Figures 4 and 6 show the same tests performed with N staging servers, 1 staging node per 1024 compute processes. In both cases, one staging process is used for each of the 8 cores on each staging node. The results seen on JaguarPF were considerably slower overall than those on the RedSky machine. After consulting with the team at Oak Ridge, we determined that the root cause of the problem was one of two possibilities. The first possibility was due to the Spider file system using hardware RAID while the initial tests on RedSky used a local only software-based RAID file system called scratch. A second set of tests were performed on a hardware RAID file system gscratch when the scratch file system was decommissioned. Unfortunately, scratch was decommissioned before all testing could be completed resulting in the incomplete results presented in Figures 3 and 4. The initial guess based on the system configuration suggested it is the difference between using shared locks for all processes on a node, such as scratch was configured, and not sharing locks, as the Spider file system at Oak Ridge is configured. All three are Lustre file systems. Spider is shared across the entire leadership computing facility at Oak Ridge making it heavily used from a variety of different machines. The scratch file system on Red Sky is a local only file system while gscratch is shared among several machines at Sandia.

The second set of results refines the understanding by evaluating against the hardware RAID file system gscratch, but with shared locks for all processes on each node. The performance of the staging results for both 1 and N staging servers on both file systems on RedSky are essentially identical. The performance of using 1 staging node on JaguarPF showed a considerably different picture. The performance on JaguarPF with 1 staging server is $10\times$ worse than either file system on RedSky. This suggested strongly there is a very large penalty for using individual locks for all processes. By looking at the PnetCDF performance across the three sets of tests, broader results appear. The performance on the shared file system gscratch on RedSky is about $3\times$ worse than the local only scratch file system. The performance of PnetCDF on JaguarPF is essentially the same as the gscratch performance on RedSky, also a shared file system.

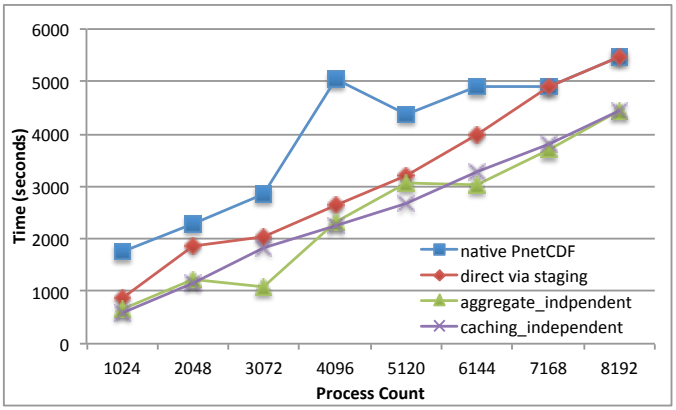

Fig. 2. Writing Performance on JaguarPF with One Staging Node (12 Processes)

Since JaguarPF is a capability-class machine, data security is seen as paramount. The possibility of corruption by sharing locks between processes on a single node is considered too high of a risk. The performance penalties are seen as acceptable compared with the cost of the computation itself. On RedSky, the old file system was a local only, software-

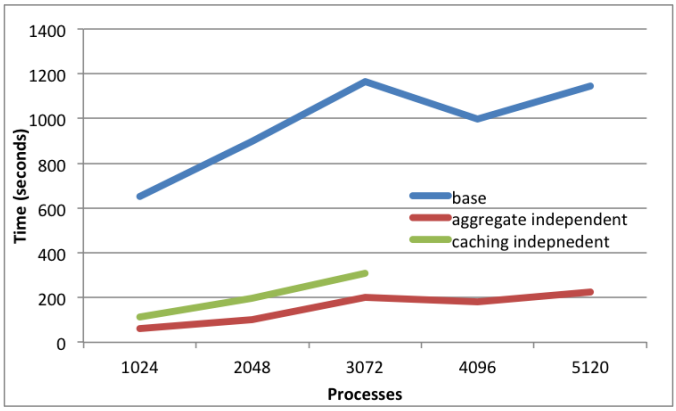

Fig. 3. RedSky scratch Results (1 staging server)

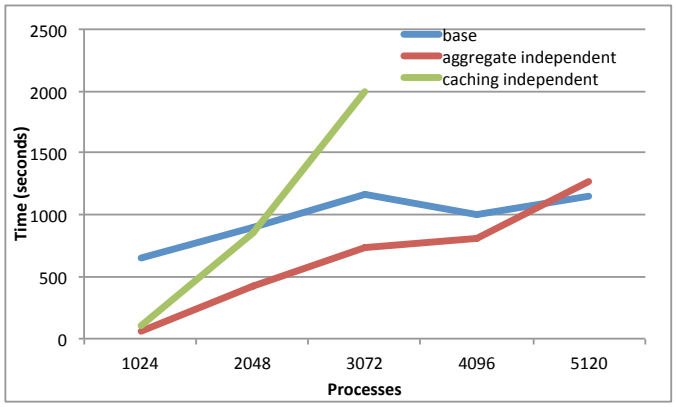

Fig. 4. RedSky scratch Results (N staging servers)

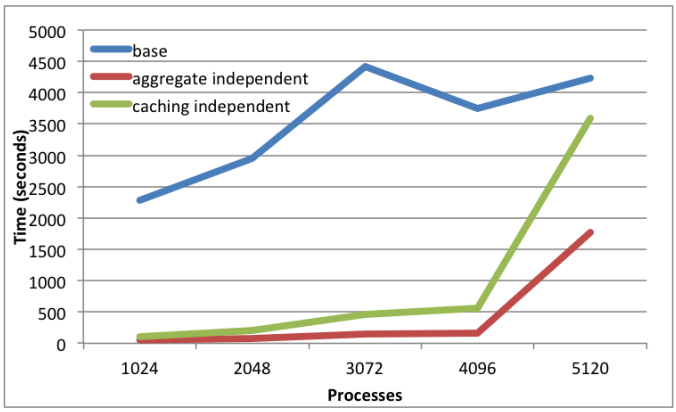

Fig. 5. RedSky gscratch Results (1 staging server)

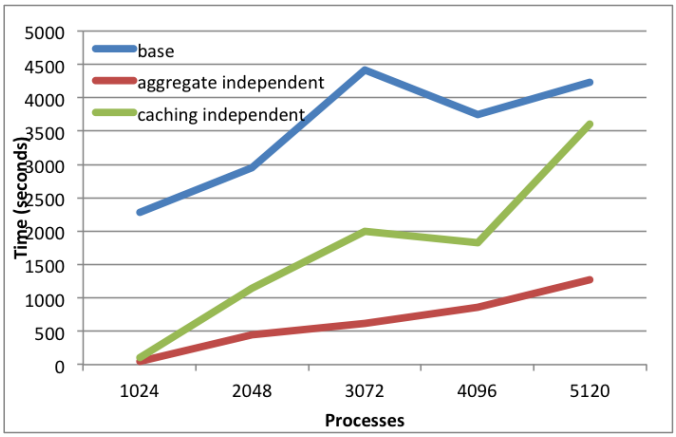

Fig. 6. RedSky gscratch Results (N staging servers)

based RAID system where file system locks are shared by all processes on a single node. This reduces the contention between processes by trusting them to safely share access to the file system. RedSky is a capacity machine not intended for production scientific runs affording the less strict data security semantics. The last configuration is the current RedSky setup with the shared, hardware-based RAID system gscratch with the locking configured like the software RAID system.

Overall, these tests have yielded the following discoveries:

- Base PnetCDF performance is roughly the same on both shared file systems. The base-line performance on the local-only software RAID was a factor of $3\times - 4\times$ better.
- Hardware vs. software RAID staged performance is roughly the same on RedSky. The 5120 process results for the hardware RAID with a single staging server is an aberration compared with the other results and is likely due to the intermittent, but regular problems the file system seems to suffer.
- Staged performance between RedSky and JaguarPF is off by an order of magnitude roughly. RedSky, no matter the file system used, had roughly a $10\times$ performance advantage. With how busy the Spider file system is at Oak Ridge and the intermittent performance variability [16], this shows the advantages of having a less busy file system for performance. With the file system using rotational media, this result is not too surprising. The surprise is the degree of difference.

While these results are strong enough to override some complicating factors, discussion of known potential impacts are discussed below.

While JaguarPF has reasonably consistent performance, the variability in the file system performance of Spider is well documented [16]. This variability affects the consistency of these results. While this variability is known, RedSky can be even more variable. Because RedSky is not intended as a capability platform, it is frequently used to test new algorithms and parallelization techniques that can detrimentally affect the performance of other applications running on the machine concurrently. With seemingly regularity, runaway processes can cause the file system to suffer very poor performance until the culprit processes are killed. By using the best time rather than an average of some sort, we do our best to control for the variability introduced by both platforms.

Another wrinkle in these results is that they do not fully control for placement of processes on compute nodes. The RedSky job scheduler slurm, by default, does not assign processes in a fixed ordering attempting to pack nodes. Instead, there is a somewhat random distribution of the processes across those assigned to this compute job. These results do partially control for placement by running the tests multiple times and using the best results from each set. Ongoing work has observed that sub-optimal placement can have a significant impact on the node-to-node performance.

B. *Informatics on HPC Resources*

Traditionally, HPC resources have focused on supporting scientific applications with little thought to using these expensive resources for a broader variety of applications.

In this section, we discuss two different applications related to informatics that leverage HPC resources. For the first, the HPC resources parallelize data processing tasks for a data warehouse appliance. For the second, this model is extended to incorporate the HPC resource into an interactive visualization system for exploring the relationship between various documents. Both of these examples share an offloading of once serial computation onto a parallel resource to accelerate the computation.

1) Connecting to Informatics Resources: Informatics applications offer a new opportunity to leverage parallel computation to accelerate data processing tasks. The challenge is getting the data into the compute area from outside the HPC resource. Mathematical operations like mean, variance, skewness, kurtosis, the covariance matrix and its Cholesky decomposition can be done piecemeal and the end result assembled by aggregating the local values and completing the computation. For this scenario, it is necessary to provide a way to retrieve data into the compute area and then write the results back to the data store.

An appliance like a Netezza data warehouse engine is frequently used to store data used by informatics applications. The derived values that help offer insights into the entire data set must be generated and stored for frequent use. To leverage the HPC resource, a two-way connection between the Netezza appliance and the HPC code must be created. This connection will have to provide the following functionality:

1) Bridge the gap between the compute area and the network outside the HPC resource.
2) Provide an interface capable of extracting the data from the appliance into the compute area.
3) Assert the results back into the appliance in a way that appropriately links it to the raw data.

Accomplishing these ends requires working with the available interface to Netezza. There are three interface options available, but only the ODBC interface offers remote access to the device from another machine. For operations on the device, a SQLite facility that processes operations through the SQL interface based on a list of commands in a file is simple and offers relatively good performance using the industry standard SQL language to encode the commands. The custom NZLoad facility offers higher performance than the SQLite interface and is intended for bulk loading of data into the device avoiding the overhead of SQL processing.

Working with the ODBC interface is fairly straightforward, but requires a direct network connection between the two sides of the API call. To make a connection between the compute area and the service nodes capable of connecting via ODBC, the Titan [17] component of VTK, which has an ODBC mode, is used. A staging area hosted on the service nodes receives the calls from the re-implemented Titan interface using NSSI underneath. From the staging area, the native implementation is invoked directly for ODBC calls on the Netezza interface. Any return values generated by the Netezza are routed back through NSSI to the compute area caller affording an indirection that bridges the barrier between the compute area and the outside network. A detailed description of this architecture was presented previously [18], but without any evaluation of the available performance. Since that time, an evaluation has been performed and is shown in Figure 7.

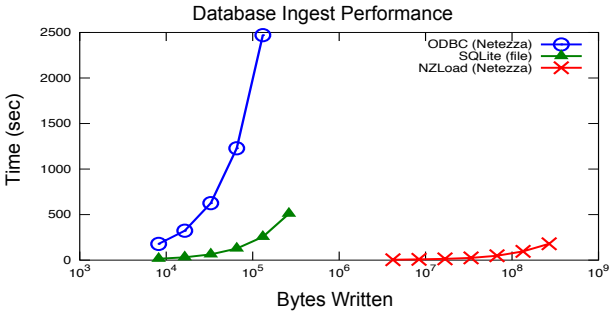

Fig. 7. Informatics Applications Interface Performance

The network link between RedStorm and the Netezza appliance is 1Gb/s. This evaluation shows three different performance profiles for the same operations. The first demonstration of these results is the capability of accessing the Netezza appliance from the compute area of Red Storm. While the performance of this ODBC interface seems poor, it is attributable to how it was used. In this case, a naive use of the API that forces a complete transaction for each insert or select operation was used. ODBC does offer higher performance options, such as combining several commands into a single transaction as well as bulk loading options. The full implications of these alternative options were not understood until after access to the Netezza device had been discontinued preventing tests using the ODBC interface more efficiently. Overall, the performance of the ODBC interface using it more efficiently should be more similar to the SQLite performance.

While this ODBC interface affords a generic interface usable for a variety of applications, the performance is not as good as custom methods, not surprisingly. However, the availability of this functionality outweighs the apparent performance penalty. Native, proprietary methods will generally offer higher performance at the cost of development time. In this case, any scenario that requires an ODBC interface can now be easily incorporated into the compute area of an HPC resource without significant additional programming efforts.

One alternative that can also be explored using this model includes exploiting the SQLite interface. The idea with this approach is to maintain the ODBC interface in the compute area, but change how the staging area works. For this case, for many scenarios, caching a collection of SQL commands into a buffer that is then sent to a local process on the Netezza appliance. That local process can then use this buffer as input for the SQLite interface. This is not applicable to all scenarios because of the potential sequence of calls, such as a select, insert, select, update sequence where the insert and update rely on values retrieved from the select commands. For cases where a bulk insert or a series of selects are being performed, these can be cached and handled in bulk through the SQLite interface.

Ideally, if a local process on the Netezza were possible, the Titan/NSSI/ODBC interface could be coded to extract the data from the ODBC calls into the proper format for the NZLoad interface and take advantage of the performance possibilities. Like the SQLite interface described above, this is

limited to a subset of possible operations limiting its use. For scenarios where this is applicable, as the evaluation shows, the performance benefits can be substantial.

2) Leveraging HPC for Document Clustering and Interactive Visualization: A more complex interaction between an HPC resource and external resources can be seen in the document clustering example. Document clustering is a technique that analyzes a collection of documents to determine a relative "closeness" between document pairs. Latent Morpho-Semantic Analysis (LMSA) [19] is a newer technique to generate these clusters, but with the ability to better handle morphologically complex languages such as Arabic. Only through exploiting large computational resources can these sorts of algorithms scale to handle millions of elements.

Architecturally, Red Storm is used for the mass computation, the clusters for the visualization and interactive controls, and data warehouse appliances for database functionality (such as a Netezza appliance) is illustrated in Figure 8. At a detail level, LMSA is used for dataset generation, the Trilinos [20] library is used for the general computation, Titan [17] performs the visualization and NSSI glues the systems together.

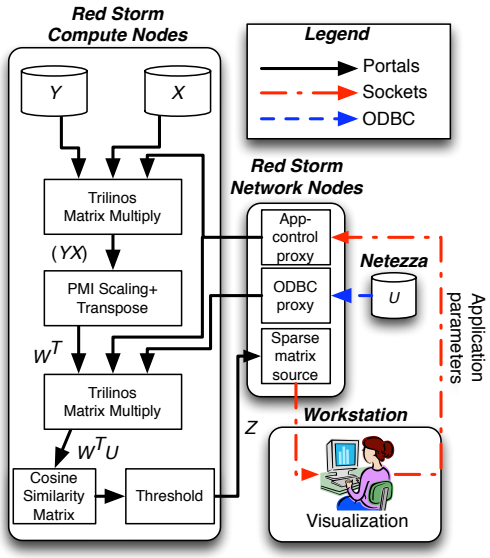

Fig. 8. Architecture for Interactive Document Clustering

For this scenario, the data staging services provides the connectivity between the HPC compute resource, the data warehouse's database functionality, the visualization platform, and the end user's terminal to view the rendering and manipulate how and what portion of the data set displayed. Leveraging the HPC resources yield a tremendous time to solution advantage over using a stand-alone workstation for the entire system. The use of NSSI to replace the existing data access/movement APIs affords a transparent way to connect all of these resources by affording moving components onto different platforms more suited to scaling the workload.

To test the effectiveness of this architecture, public domain copies of the Bible in five different languages is processed to demonstrate the clustering of all of the different language versions of the same source document. A performance evaluation is shown in Figure 9 and the workstation view of the clustering

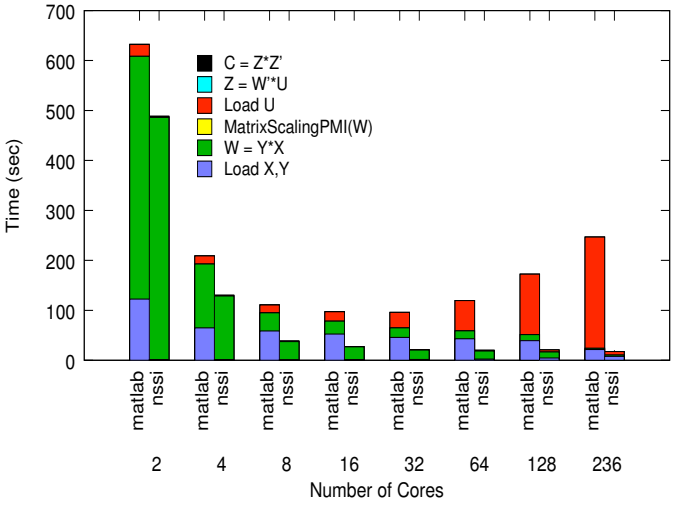

Fig. 9. Document Clustering Performance on For Bible Dataset

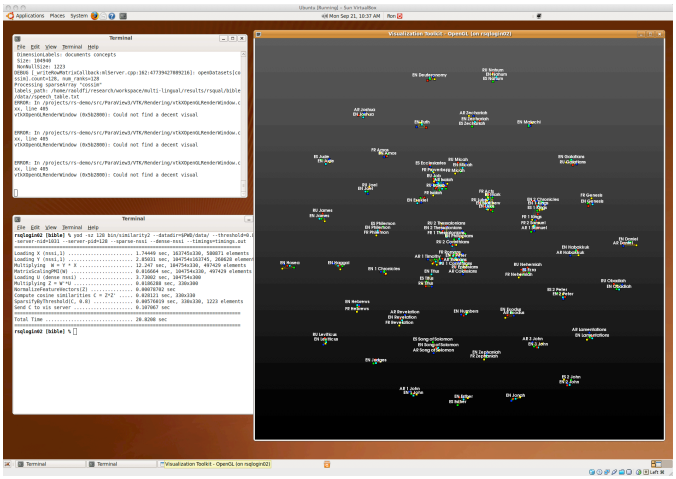

Fig. 10. Workstation View of Document Clustering

is shown in Figure 10. The workstation view shows the user interface an end user would experience. The large area on the right side shows that the elements analyzed have successfully clustered, even though they are in different languages, using the LMSA algorithm. For the performance comparison, Matlab is compared against using Red Storm as the computational engine. In Figure 9, the red portion of the bars labeled, 'Load U', represents the time spent bringing data into the processing engine. With Matlab, the computation is replaced by IO as this computation scales. For the NSSI-based version using Red Storm, the load times are minimal and the various computation times reduce continuously as the number of processing cores increases. This approach offers faster time to solution offering the user a more flexible 'what if' opportunity than could be performed before.

The integration of the HPC and visualization resources into the end-user workstation workflow offers far greater computation capabilities than available on the workstation alone. The interactivity limitations of HPC resources require some other interface to the end user. This integration demonstrates the potential of leveraging the computation capabilities of HPC resources to feed interactive displays. Ultimately, this model

may inform how exascale science workflows are performed. For example, an interactive visualization is manipulated by a scientist exploring a running simulation. At the appearance or notice of a particular feature, the end-user can then inform the HPC resources to adjust the computation, such as moving backwards in time to just before the feature appears, and then move the simulation forward. This integration of various heterogeneous resources as demonstrated in this scenario will be critical should this model become mainstream.

C. Streaming Analysis

Like the document clustering example in Section IV-B1, the HPC resource can be leveraged for other non-scientific simulation applications with the right infrastructure. In this case, analysis of Ethernet frames to detect patterns of or particular network traffic can leverage the parallel processing capabilities of the HPC platform to accelerate analysis. For this case, compressed collections of Ethernet frames are sent into the HPC resource where it is distributed to the parallel processing engines to perform the analysis. This is illustrated in Figure 11.

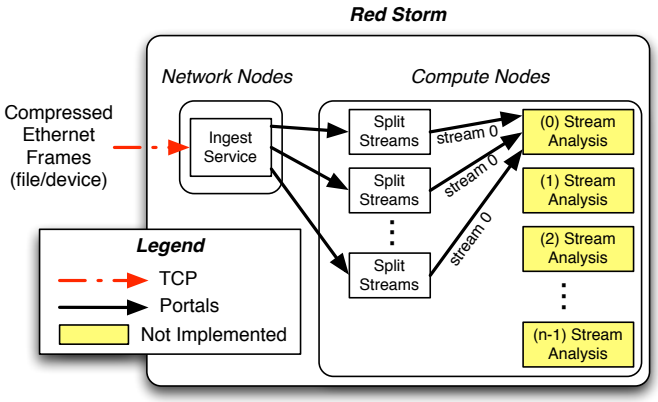

Fig. 11. Streaming Analysis Architecture

To test the performance of this architecture, an ingest service reads from a file of 8543 compressed IP packets for processing. These compressed packets are sent to RedStorm for processing. Client processes request 100 frames at a time for processing. Each client then decompresses, analyzes and sends the results back to the server. The results are presented in Figure 12. Through using this architecture, we are able to achieve 900 Mbps of sustained analysis performance and can scale this up adding additional processing simply based on the available compute capacity. The mostly embarrassingly parallel nature of the analysis makes this an ideal candidate for parallel processing using resources such an Red Storm. In this case, the long term bottleneck will be the bandwidth between the HPC resource and the outside network where the packets are sourced. Time for additional analysis of the packets can be hidden through the parallel processing maintaining the throughput of the system.

These sorts of streaming-style examples with potentially embarrassingly parallel analysis do not really need the high speed interconnect typical of an HPC platform, but the parallel

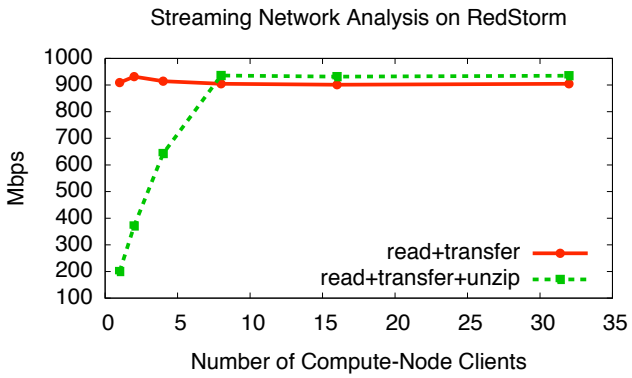

Fig. 12. Streaming Analysis Performance

computation capacity provides excellent scalability. As mentioned above, the ingress and egress bandwidth to the cluster are really the scaling limiting factors.

V. Conclusions and Future Work

These various experiments have demonstrated the potential of using a few additional resources, such as is done with traditional data staging, to enable different kinds of processing using HPC resources. By exposing access to external resources, affording rapid data insertion and extraction, or integrating the HPC resource into an interactive workflow all yield interesting potentials.

While these scenarios have demonstrated both the generality and performance of this approach, more work still needs to be done. First, programming models need to be developed that simplify this sort of processing. The NSSI library has afforded a relatively simple integration by offering indirection services for nearly any API. While the Titan/ODBC service offers a more general tool, this approach generally requires custom development for each class of solution. Efforts must be made to develop a general way to describe these sorts of services that use a proxy element for connecting between disconnected resources.

Along with the programming models, security must be addressed. HPC resources have compute nodes isolated from the outside network not just to avoid potential interference and jitter introduced from external sources, but also to offer a security envelope that offers some guarantees about who and from where data can be accessed. In this case, a global security envelope that extends beyond a single resource to protect data must be developed. ORNL has developed a primitive version of this for use with the Kepler workflow engine by using an SSH proxy to tunnel into the HPC resource from outside [21]. This tunnel is then used to communicate between the compute processes and the external workflow control system. This system is not generic enough nor does it address this as a general problem.

One challenge that these techniques currently experience is the difficulty of scheduling service jobs along with the compute processes. Current schedulers do not offer support for scheduling the deployment of applications onto service nodes as part of compute job. Instead, the service processes must be deployed ahead of time and wait for the compute job to start. Ideally, different classes of resources will be made available

on HPC resources with joint or separate scheduling. This will afford requesting 1 service node along with 100,000 compute cores to enable these sorts of integrations of HPC resources into external applications.

Ideally, the hardware for the service nodes should be able to be custom or specialized pieces of hardware affording tighter integration or better processing options than a standard compute node may offer. For example, the Netezza appliance has an interface hardware module that could be installed as part of the HPC resource to afford direct NZLoad access from the compute area. Alternatively, a service node with a large amount memory compared to a compute node can be used to perform visualization processing or other data intensive tasks that are not as communication intensive. This specialized hardware would complicate the HPC resources, but the advantages make the costs worthwhile for many applications.

Lastly, as with any increase in resources for a compute job on the HPC resource, resilience must be considered. Lack of support for scheduling jobs on service nodes makes a failure of those nodes fatal to the larger compute job. This sort of fragility is not acceptable for long-term use of this architecture. Instead, resilience support for integrating these various components so that connections can be reformed after failures must be developed.

VI. Acknowledgements

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

An award of computer time was provided by the Innovative and Novel Computational Impact on Theory and Experiment (INCITE) program. This research used resources of the Oak Ridge Leadership Computing Facility, located in the National Center for Computational Sciences at Oak Ridge National Laboratory, which is supported by the Office of Science of the Department of Energy under Contract DE-AC05-00OR22725.

References

[1] R. A. Oldfield, P. Widener, A. B. Maccabe, L. Ward, and T. Kordenbrock, "Efficient data-movement for lightweight I/O," in *Proceedings of the 2006 International Workshop on High Performance I/O Techniques and Deployment of Very Large Scale I/O Systems*, Barcelona, Spain, Sep. 2006. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/CLUSTR.2006.311897

[2] R. A. Oldfield, A. B. Maccabe, S. Arunagiri, T. Kordenbrock, R. Riesen, L. Ward, and P. Widener, "Lightweight I/O for scientific applications," in *Proceedings of the IEEE International Conference on Cluster Computing*, Barcelona, Spain, Sep. 2006. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/CLUSTR.2006.311853

[3] R. Brightwell, K. Pedretti, K. Underwood, and T. Hudson, "SeaStar interconnect: Balanced bandwidth for scalable performance," *IEEE Micro*, vol. 26, no. 3, pp. 41–57, 2006.

[4] R. Brightwell, R. Riesen, B. Lawry, and A. B. Maccabe, "Portals 3.0: protocol building blocks for low overhead communication," in *Proceedings of the International Parallel and Distributed Processing Symposium*. IEEE Computer Society Press, Apr. 2002.

[5] I. T. Association, "InfiniBand Architecture Specification, Release 1.2," October 2004.

[6] R. Alverson, D. Roweth, and L. Kaplan, "The gemini system interconnect," *High-Performance Interconnects, Symposium on*, vol. 0, pp. 83–87, 2010.

[7] J. Lofstead, R. Oldfiend, T. Kordenbrock, and C. Reiss, "Extending scalability of collective io through nessie and staging," in *The Petascale Data Storage Workshop at Supercomputing*, Seattle, WA, November 2011.

[8] R. A. Oldfield, D. E. Womble, and C. C. Ober, "Efficient parallel I/O in seismic imaging," *The International Journal of High Performance Computing Applications*, vol. 12, no. 3, pp. 333–344, Fall 1998. [Online]. Available: ftp://ftp.cs.dartmouth.edu/pub/raoldfi/salvo/salvoIO.ps.gz

[9] A. Nisar, W.-k. Liao, and A. Choudhary, "Scaling parallel I/O performance through I/O delegate and caching system," in *SC '08: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*. Piscataway, NJ, USA: IEEE Press, 2008, pp. 1–12.

[10] N. Ali, P. Carns, K. Iskra, D. Kimpe, S. Lang, R. Latham, R. Ross, L. Ward, and P. Sadayappan, "Scalable I/O forwarding framework for high-performance computing systems," in *Proceedings of the IEEE International Conference on Cluster Computing*, September 2009. [Online]. Available: http://www.mcs.anl.gov/research/projects/iofsl/pubs/cluster09-paper.pdf

[11] H. Abbasi, J. Lofstead, F. Zheng, S. Klasky, K. Schwan, and M. Wolf, "Extending i/o through high performance data services," in *Cluster Computing*. Luoisiana, LA: IEEE International, September 2009.

[12] F. Zheng, H. Abbasi, C. Docan, J. Lofstead, S. Klasky, Q. Liu, M. Parashar, N. Podhorszki, K. Schwan, and M. Wolf, "PreDatA - preparatory data analytics on Peta-Scale machines," in *In Proceedings of 24th IEEE International Parallel and Distributed Processing Symposium, April, Atlanta, Georgia*, 2010.

[13] C. Docan, M. Parashar, and S. Klasky, "DataSpaces: An interaction and coordination framework for coupled simulation workflows," *HPDC '10: Proceedings of the 18th international symposium on High performance distributed computing*, 2010.

[14] V. Vishwanath, M. Hereld, and M. Papka, "Toward simulation-time data analysis and i/o acceleration on leadership-class systems," in *Large Data Analysis and Visualization (LDAV), 2011 IEEE Symposium on*, oct. 2011, pp. 9 –14.

[15] R. Brightwell, K. D. Underwood, and C. Vaughan, "An evaluation of the impacts of network bandwidth and dual-core processors on scalability," in *International Supercomputing Conference*, Dresden, Germany, June 2007.

[16] J. Lofstead, F. Zheng, Q. Liu, S. Klasky, R. Oldfield, T. Kordenbrock, K. Schwan, and M. Wolf, "Managing variability in the IO performance of petascale storage systems," in *Proceedings ofSC2010: High Performance Networking and Computing*, Nov. 2010.

[17] B. Wylie and J. Baumes, "A unified toolkit for information and scientific visualization," in *Proceedings of the SPIE Conference on Visualization and Data Analysis*, K. Börner and J. Park, Eds., vol. 7243, no. 1. San Jose, CA, USA: SPIE, Jan. 2009.

[18] R. A. Oldfield, A. Wilson, G. Davidson, and C. Ulmer, "Access to external resources using service-node proxies," in *Proceedings of the Cray User Group Meeting*, Atlanta, GA, May 2009.

[19] P. A. Chew, B. W. Bader, and A. Abdelali, "Latent Morpho-Semantic Analysis: Multilingual information retrieval with character n-grams and mutual information," in *Proceedings of the 22nd International Conference on Computational Linguistics (Coling 2008)*, Manchester, UK, Aug. 2008, pp. 129–136. [Online]. Available: http://www.aclweb.org/anthology/C08-1017

[20] M. Heroux, R. Bartlett, V. H. R. Hoekstra, J. Hu, T. Kolda, R. Lehoucq, K. Long, R. Pawlowski, E. Phipps, A. Salinger, H. Thornquist, R. Tuminaro, J. Willenbring, and A. Williams, "An overview of trilinos," Sandia National Laboratories, Tech. Rep. SAND2003-2927, 2003.

[21] N. Podhorszki, S. Klasky, Q. Liu, H. Abbasi, J. Lofstead, K. Schwan, M. Wolf, F. Zheng, C. Docan, M. Parashar, and J. Cummings, "Plasma fusion code coupling using scalable i/o services and scientific workflows," in *In Proceedings of The 4th Workshop on Workflows in Support of Large-Scale Science at Supercomputing 2009*, 2009.