Analysis of the Blue Waters File System Architecture for Application I/O Performance

Kalyana Chadalavada and Robert Sisneros
National Center for Supercomputing Applications
University of Illinois at Urbana-Champaign
Urbana, IL USA
Email: {kalyan, sisneros}@illinois.edu

***Abstract*—The NCSA Blue Waters features one of the fastest file systems for scientific applications. Using the Lustre file system technology, Blue Waters provides over 1 TB/s of usable storage bandwidth. The underlying storage units are connected to the compute nodes in a unique fashion. The Blue Waters file system connects a subset of storage units to the high speed torus network at distinct points. Utilizing standard benchmarks and scientific applications, we examine the impact of this architecture on application I/O performance. Given the size of the system and its intended applications, scaling I/O performance will be a challenge. Identifying the optimal I/O methodology can help alleviate a large number of application performance issues. All exercises are done in a production environment to ensure that beneficial results are directly applicable to Blue Waters users.**

***Keywords*-File systems**

I. Introduction

The NCSA Blue Waters features one of the fastest file systems for scientific applications. Using the Lustre file system technology, Blue Waters provides over 1 TB/s of usable storage bandwidth. Due to the size of the system and the intended applications, scaling I/O performance will be a challenge. Identifying the optimal I/O methodology can help alleviate a large number of application performance issues.

The underlying storage units of Blue Waters are connected to the compute nodes in a unique fashion; the file system connects a subset of storage units to the high speed torus network at distinct points. This means that any given node on the system is not at a uniform distance from the disks, routers, and servers that make up the file system. It is foreseeable that in such an architecture the node allocation for a job, the location of the processes performing I/O, and the target disk on the torus network could affect I/O throughput. In case of striped files, a process may experience variation in I/O throughput due to the distribution of the file stripes among available disks.

Utilizing standard benchmarks and scientific applications, we explore the impact of a jobs node allocation as well as the location of its files on an application's performance and runtime consistency. We present results from experiments with varying Lustre stripe settings for applications targeted for Blue Waters. We then apply the results and observations our experiments to a real world scientific application appropriate for the Blue Waters system. All experiments take place in a normal production environment to ensure the results are collected under conditions representative of those a user would experience on the system. That is, our resulting experimental outcomes are not altered through the use of dedicated resources or optimized configurations.

II. Overview of the Blue Waters I/O Subsystem

Blue Waters provides users with three distinct file systems: home file system for user home areas, project file system for group level file sharing and collaboration, and a scratch file system for applications. All are implemented using the Cray Sonexion Lustre storage technology and each has its own metadata and object storage servers. Separating the metadata services eliminates interference among file systems thereby providing interactive users as well as batch applications with the best possible metadata performance within their respective file systems. The scratch file system is the fastest of the three and provides approximately 980 GB/s peak throughput.

Cray Sonexion 1600 (CS-1600) units form the basis for storage on the Blue Waters system. Each CS-1600 enclosure hosts two Lustre Object Storage Servers (OSSs). Each OSS has an associated disk storage unit, the Scalable Storage Unit (SSU). Lustre Object Storage Targets (OSTs) carved out from the SSU are assigned to OSSs within the CS-1600 enclosure. Also, a CS-1600 hosts a Metadata unit hardware configuration to provide the Lustre Metadata Services (MDS).

Cray provides a Lustre Network Driver (LND) for its Gemini high-speed network (HSN). Using this LND, compute nodes mount Lustre file systems over the Gemini network. File system operations from the clients are routed to the appropriate Lustre device using the Cray XIO nodes. The XIO nodes implement the LNET routing services and handle packet routing between different networks. Currently, the compute nodes use Lustre version 1.8.6. The Lustre connectivity and software stack of Blue Waters is illustrated in Figure 1.

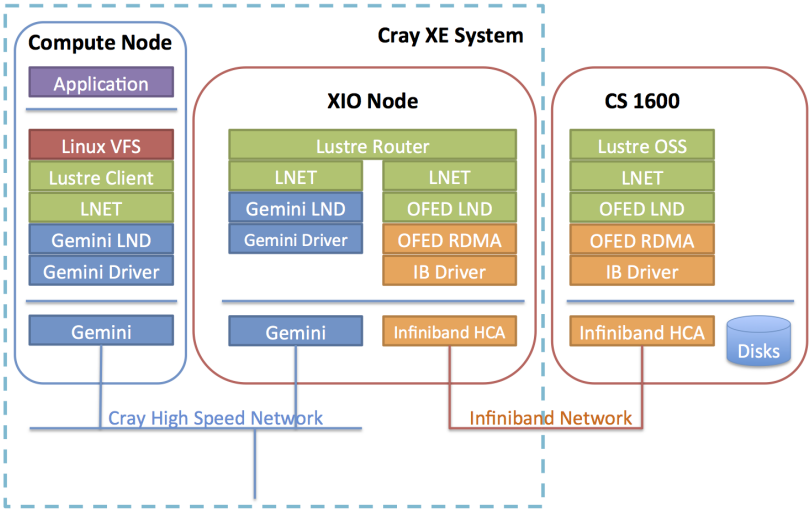


Figure 1: The Lustre connectivity and software stack.

For the remainder of this paper, we focus discussions and experiments on the Blue Waters scratch file system. The scratch file system consists of 1400 OSTs providing over 20PB of usable disk space. There are 482 LNET routers, 480 for OSS traffic and the remaining for MDS. These provide the connectivity from the Blue Waters Gemini HSN to the scratch file system.

Typically, LNET routers and Lustre devices are interconnected with a separate Infiniband or Ethernet network. On Blue Waters, LNET routers are organized into groups of four and each group is connected to three OSSs. A set of four LNET routers are configured as the primary routers for OSTs on these OSS units. A second group of LNETs act as a backup ensuring alternate routes to any OST in the event of failures. An overview schematic is given in Figure 2.

With this design, file system requests from compute nodes are routed on the Gemini network to the specific LNET routers that are configured as the primary routers for the target Lustre device. These unique characteristics of the Blue Waters file system present new opportunities for fine tuning low-level system software. Evaluation of how design consequences affect I/O performance may lead to the pinpointing of those components to leverage as well as those to minimize to increase performance. We hypothesize that the following are optimization targets with potential:

- Distance to various Lustre devices from a given point on the Blue Waters Gemini torus network varies, can this be leveraged to configure job/file layouts to improve performance?
- Application and Lustre traffic are concurrently routed on the same HSN. Will this cause undesirable interference among MPI and I/O operations? If so, is it avoidable?

Current layout of the LNET routers is shown in Figure 3. Each rectangle represents a Blue Waters rack populated with the multiple blades. A rack with one or more XIO blades is shaded with a light green color. A rack with no XIO blades is shaded in blue.

III. Evaluation

The Lustre file system allows a user to customize certain file parameters via a command line utility (`lfs`) as well as the provided API. These parameters include the number of OSTs to stripe a file across, the size of each stripe, and the OST on which to start striping (offset). Changing these values from the system defaults can have significant impacts on I/O for an application. The effects of stripe count and size are well known and are routinely targeted for performance increases and are documented by HPC centers to illustrate optimal use for their specific systems. Attention is rarely directed toward OST offset for performance increase in that for many systems is it unlikely (or impossible) for offset

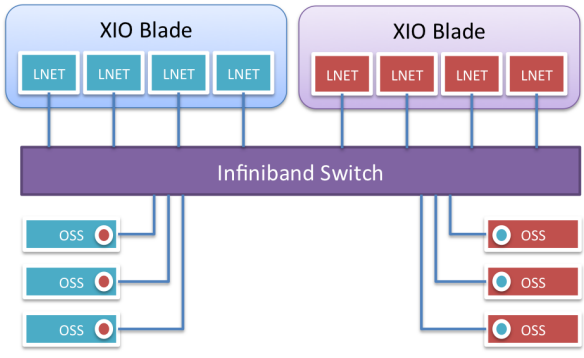


Figure 2: Blue Waters LNET connections. Each OSS is colored to match its primary group of LNETs and annotated with a circle colored to match its secondary.

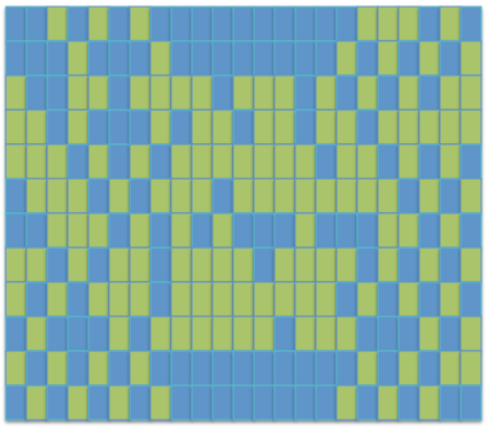


Figure 3: The machine room layout of Blue Waters' racks. Racks with one or more XIO blades are shaded light green, those with none are shaded blue

changes to affect performance. However, given the unique design of Blue Waters (Section II), it is unclear that this holds true. In this section we investigate the possibility that OST offset can improve I/O performance. To this end, we elicit illustrative data through system tests with an I/O benchmark.

A. IOR

The Interleaved or Random (IOR) I/O Benchmark [1] was developed at Lawrence Livermore National Laboratory for measuring read/write performance of parallel file systems. There are a number high-level customizable parameters that may be set; these include file size, I/O transaction size, sequential vs. random access, and single shared file vs. file-per-process. There are also several flags for fine tuning a particular run that allow a user to have IOR perform tasks such as consume system memory (in addition to measured I/O tasks) or keep temporary files created and written for tests. In addition, the diverse repertoire of configurations may be applied to any of IOR's supported APIs: POSIX, MPIIO, HDF5, and NCMPI. These features make IOR an ideal benchmark for an HPC facility and it is indeed widely used in that community [2], [3], [4].

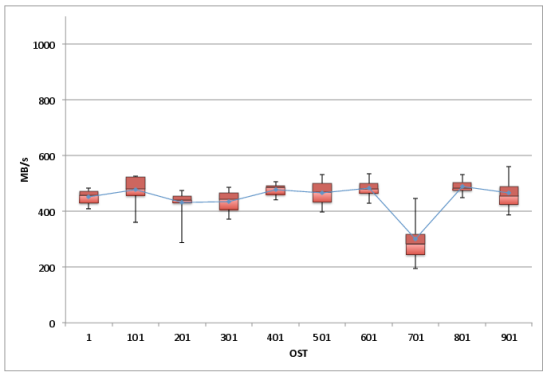
B. Performance

We performed two tests: the first to gain insight into the effects of default system OST selection and the second to formulate the performance expectations of OST selection. For both tests, IOR mimics a typical application using the POSIX API with an I/O transaction size of 16MB that consume 50% of the memory of each allocated node. The output files are always created in directories with 12MB stripe sizes which results in misaligned I/O operations coinciding with typical usage. Each test consists of multiple IOR runs via separate `aprun` calls as well as multiple reads and writes within a single IOR run for averaging. To maximize the equality of runs, we flush file system buffers between `aprun` calls, using `sync`, and have IOR wait 0.1 seconds between test iterations.

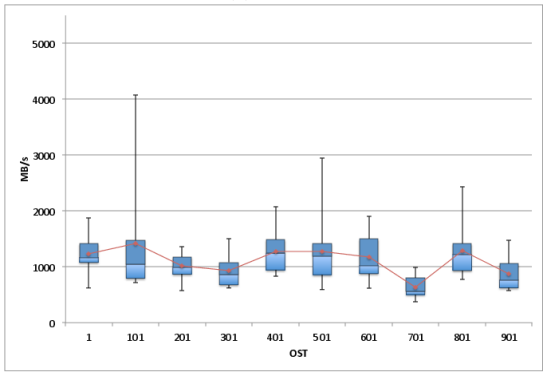
These tests are not designed to benchmark overall I/O throughput of the Blue Waters scratch file system. Rather, they are intended only to characterize the impact of OST distance/selection from a given fixed point on the network.

1) Parallel Variation: In this test, we aim to characterize the effects of OST selection on I/O performance. We do this by comparing runs that utilize parallel I/O while altering both OST offsets as well as the number of OSTs written to (file striping). Our test job consists of 64 cores on 4 nodes with each responsible for 16MB of a 1GB shared file. From a user's perspective an OST offset is assigned at random. To emulate this we arbitrarily selected 10 OSTs to serve as our different offsets $(1, 101, 201, \ldots, 901)$. We perform I/O with these offsets for files of a single stripe on up to files striped across 10 OSTs. Each of the 100 tests is written to a unique directory on Blue Waters' scratch file system where the Lustre parameters have been appropriately set (files created in a directory inherit the striping specifications of that directory). Also, each IOR run performs 10 reads and writes for averaging for a total of 1000 measurements of each.

Figures 4, 5, and 6 contain the results from these IOR

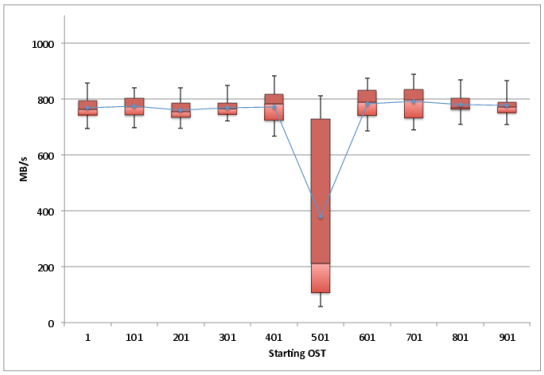


(a) writes

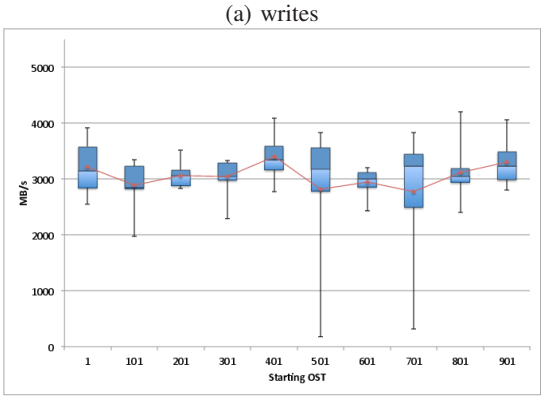


(b) reads

Figure 4: IOR variation test: single OST, differing offset.

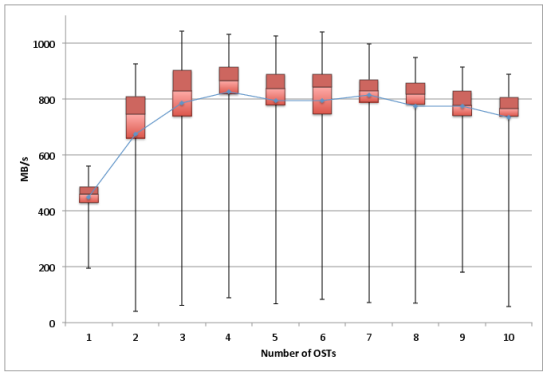


(a) writes

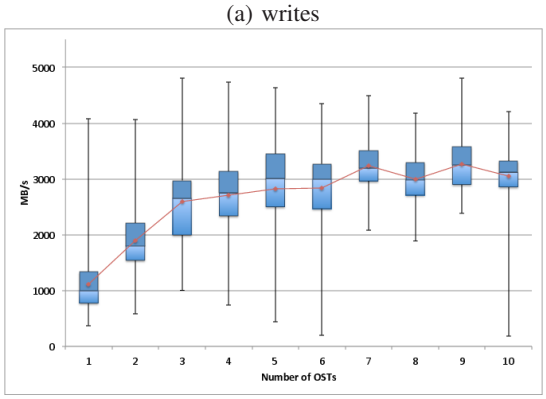


(b) reads

Figure 5: IOR variation test: 10 OSTs, differing offsets.



(a) writes



(b) reads

Figure 6: Averages of all runs sorted by number of OSTs.

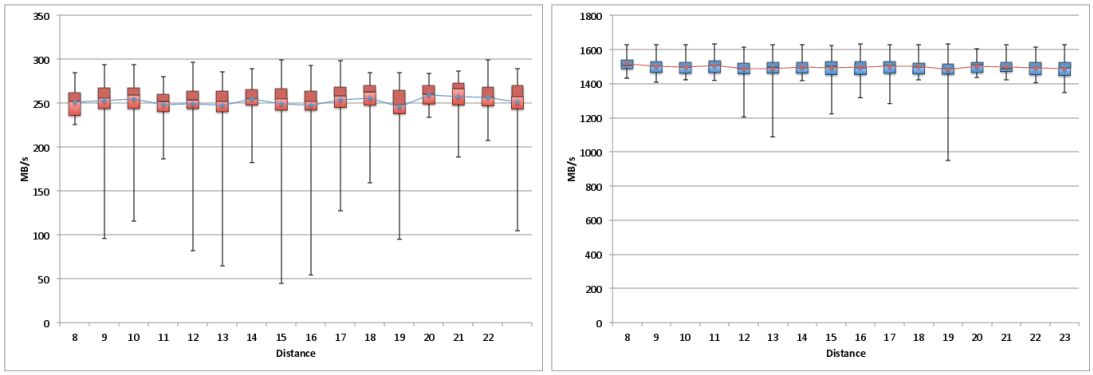
runs compiled into box-and-whisker plots (each box from bottom: minimum value, first quartile, median, third quartile, maximum value) with the points of the connecting line lying on each run's average value. We selected this plot style because both the throughput (average lines and boxes) and consistency (max/min error bars) are represented. In Figures 4 and 5 each "column" is calculated from 10 values, whereas for Figure 6 each is over all runs that share the same file striping (100 values).

In Figure 6 we see the expected increase in performance from moving from a single to multiple OSTs, but overall, there appears to be no appreciable difference in throughput connected to the selection of OSTs. There is however, a large variation of observed results that could be related to OST selection or interference from other jobs. We believe Figure 5 displays one such example of some combination of these factors severely degrading performance, the OST pool starting with OST 501. On the whole, these figures illustrate the benefit of increasing consistency in that there are many cases where in lieu of the obvious trend of multiple OSTs increasing throughput, increasing them may actually *decrease* throughput for the unlucky user.

In Section III-B2 we directly measure the effects of OST offset selections. We chose to have IOR only perform serial I/O for that test. From Figure 6(a) it is clear that the least variation occurs when writing to a single OST. Therefore, a serial I/O operation to a single OST represents the most difficult case in which to detect a basis for I/O improvement. Furthermore, a user without root privilege is currently unable to create a pool of OSTs, which we see as a necessity for definitive tests of multiple OST selection. However, serial measurable increase in I/O performance would provide rationale for enabling this functionality and lead to future tests.

2) OST Selections: For this test the Lustre utility was used to create 1440 directories each with a unique offset corresponding to one of the 1440 available OSTs. These directories are also located on Blue Waters' scratch filesystem. From the same compute node, IOR writes/reads a 768MB file (twice) to each of the 1440 directories. The 3000 I/O operations are then sorted by a simple routing approximation metric. This metric is simply the unweighted, torus-aware Manhattan distance (hop count) from the compute node to the primary LNET group associated with each OST. In addition to the omission of any intricacies of the system's actual routing algorithm that there is also no guarantee that I/O is actually routed through the primary LNET. These considerations motivate our use of the phrase "simple routing approximation."

Figure 7 shows the I/O performance of writes (a) and



(a) writes (b) reads

Figure 7: I/O from one compute node to each of the 1440 OSTs.

reads (b) of the 16 different distance values (unique routing approximates). First, we verified what was suggested in Section III-B1 regarding OST selection and I/O throughput; the calculation of Pearson's correlation coefficient reveals no correlation among them. We find this to be an excellent result in that it lends credence to the unique design of Blue Waters' I/O subsystem. We do, however, in Figure 7 notice that writing to the nearest for all save one distance provides the most consistent performance.

While there is no direct proportionality between offset distance and consistency, we still find inspiration to believe a wise selection of offset exists, and that making this choice can improve I/O consistency. First, refinement to our metric will provide updated and more accurate results. The wave patterns in Figure 7 hint that our metric may be mishandling at least part of the calculations around the 3D torus or incorrectly weighting torus dimensions. Fixing this may very well unravel the wave into a distribution with a more pronounced relationship between offset and consistency. On the other hand, if our metric is perfect the data in Figure 7(a) may still be reasonably interpreted as a write to the nearest LNET node will offer consistent performance 93.33% more often than not. That is, doing so is more consistent than 14 of the other 15 possibilities.

IV. Toward the Improvement of Application I/O

In this section we provide results toward verifying that distance from I/O performing compute nodes to LNET nodes is a heuristic worth minimizing to achieve consistent I/O performance.

Based on the results from the IOR experiments in section III, we applied the same approach to a scientific application, Enzo [5], [6]. Enzo is an adaptive mesh refinement (AMR), grid-based hybrid code (hydro + N-Body) which is designed to do simulations of cosmological structure formation. Enzo was chosen as it is in currently in use on Blue Waters. Enzo version 2.1.0 was used for the following results.

Enzo reads in initial conditions files and outputs simulation data using the HDF5 structured data format. Output data is then identified with a dump number, `NNNN`. The combination of dump number and a base name is used to create directories for each data dump. Each process writes its own file with the process rank `cpuNNNN` appended to the directory name. The size of these files remains constant across all processes per data dump but increases as the simulation progresses.

For the purposes of this paper, we used standard input parameter files, AMR dark matter cosmology simulation from the Enzo code base. This input set uses a grid size of 128^3 and was run using 128 processes on 8 Blue Waters compute nodes. The run generated a total of approximately 11000 files across 64 Data output directories and 17 RedShift data directories.

Using Lustre user space utilities, the output files were individually assigned specific OSTs, with a stripe count of 1 and a stripe size of 5MB. The OST offset selection was made so that the process writing data to that file was as close as possible to the OST, as calculated by the metric referenced in Section III-B2. Enzo uses the HDF API call `H5Fcreate` with the `H5F_ACC_TRUNC` access flag to create the output files. Creating the files before the simulation starts will not adversely impact the behavior of the `H5Fcreate` call.

The Enzo code was run three times using both specifically selected OSTs and the default file system allocation. Wall-clock times in each case were compared and are summarized in Table I. The results indicate a more consistent runtime for the earlier case while the latter shows a large runtime variance. The runtime variance is heavily influenced by the state of the machine. When the system has a high utilization, there is a higher probability of interference from other jobs. For a non I/O bound application or with low system utilization, the benefit from specifically picking closer OST offsets may not be significant.

This data affirms our expected increase in I/O consistency through the heuristic of simply selecting "close" offsets.

	Runtimes		Decrease
	Default Offset	Selected Offset	
Run 1	2571.37	2305.21	10.4%
Run 2	2968.69	2355.28	20.7%
Run 3	2594.47	2325.53	10.4%
Variability	223.02	25.18	88.7%

Table I: Enzo results. Tests were run with default as well as hand-picked offsets.

However, contrary to our metric suggesting no correlation between distance and throughput, we see faster runtimes when selecting close offsets as well. We are hopeful that tuning our calculations will lead to a more direct exposition of the relationship between offset selections and consistency as well as throughput.

V. Conclusion

In this paper we present the groundwork analysis toward understanding the Blue Waters file system architecture with regard to application runtime consistency and performance. Specifically, our tests were tailored to quantify the relationship between the variability of an application's I/O with the changes in distances to various file system components. Our results suggest, and our initial tests confirm, that exerting currently accessible fine-grained control over network locations can enhance I/O performance.

We are encouraged having achieved performance gains with such a limited estimation of routing protocols. We believe refining this metric is the next logical step in better defining and measuring the advantage of controlling the locations of I/O operations. Furthermore, it should be noted that in our IOR testing the metric calculations show the assigned compute node is no nearer than eight hops to any LNET router. Given that it is possible for a compute node to be only a single hop away, there are interesting extreme-case tests left to perform. Also, we were limited in our experiments by not being able to control striping on multiple OSTs; we are addressing the implementation of a workaround to expand our testing. Finally, we intend to ensure future research remains complementary to scientific applications running at larger scales.

Acknowledgment

The authors gratefully acknowledge Michelle Butler and Alex Parga from the Blue Waters Storage team as well as Manisha Gajbe from the Blue Waters Scientific & Engineering Applications team. Their expertise and guidance was invaluable over the course of this work. This research is part of the Blue Waters sustained-petascale computing project, which is supported by the National Science Foundation (award number OCI 07-25070) and the state of Illinois. Blue Waters is a joint effort of the University of Illinois at Urbana-Champaign and its National Center for Supercomputing Applications.

References

[1] "IOR: interleaved or random hpc benchmark." [Online]. Available: https://github.com/chaos/ior

[2] H. Shan and J. Shalf, "Using IOR to analyze the I/O performance for HPC platforms," in *Cray Users Group Meeting (CUG) 2007*, Seattle, Washington, May 2007.

[3] P. Wauteleta and P. Kestener, "Parallel io performance and scalability study on the prace curie supercomputer," Partnership For Advanced Computing in Europe (PRACE), Tech. Rep., September 2009.

[4] *Demonstrating lustre over a 100Gbps wide area network of 3,500km*, Salt Lake City, Utah, 11/2012 2012.

[5] "The enzo project." [Online]. Available: http://enzo-project.org/

[6] B. O'shea, G. Bryan, J. Bordner, M. Norman, T. Abel, R. Harkness, and A. Kritsuk, "Introducing Enzo, an AMR Cosmology Application," 2005, pp. 341–349.