Shared Libraries on a Capability Class Computer

Suzanne M. Kelly[1], Ruth Klundt[2] and James H. Laros III[1]
[1]Sandia National Laboratories*, [2]Hewlett-Packard Company

smkelly@sandia.gov, rklundt@sandia.gov, jhlaros@sandia.gov

ABSTRACT

Popularity of dynamically linked executables continues to grow within the scientific computing community. The system software implementation of shared libraries is non-trivial and has significant implications on application scalability. This paper will first provide some background on the Linux implementation of shared libraries, which was not designed for distributed HPC platforms. This introductory information will be used to identify the scalability issues for massively parallel systems such as the Cray XT/XE product lines. Lastly, the presentation will describe the considerations and lesson learned in file system placement of the shared libraries on Cielo, a Cray XE6 system with over 100,000 cores. Scaling results and comparisons are included.

Keywords

Shared Libraries, Shared Objects, XE6, HPC, MPP.

1.0 Problem Description

The Department of Energy's National Nuclear Security Administration (NNSA) has a long history of simulating complex problems that require massive amounts of memory and CPU cycles. To solve these problems, the Advanced Simulation and Computing (ASC) Program within NNSA has provided capability-class computers which are designed to support a single job executing on all nodes, and all cores, of a massively parallel processor (MPP) supercomputer. Cielo, a Cray XE architecture platform, is NNSA's latest capability class system. It contains 143,104 compute cores (8944 *nodes* x 16 *cores per node*) and almost 290 Terabytes of DDR3 memory.

Figure 1: Cielo is located at Los Alamos National Laboratory and has 96 XE6 cabinets.

Historically, applications running in capability mode are built statically. Increasingly, application code groups are asking for the flexibility provided by dynamic shared objects. Dynamically linked executables have been supported on desktop system for decades. More recently, they have been successfully used on cluster computers of considerable scale (thousands of cores). When specifying the Cielo system, the procurement team elected to specify support for dynamically linked executables. However, full scale testing was done with static binaries.

Following the delivery of the system, the bring-up team began assessing the options for providing support for dynamic libraries. We considered both hardware and software alternatives to achieve as much scalability as possible when using a dynamically linked executable. One important aspect was that in addition to system shared objects, the user community wished to provide their own shared libraries and other dynamic objects, such as python modules.

Section 2 of this paper provides background information on shared objects. We follow that with a description of Cray CLE software support and associated hardware solutions in Section 3. Section 4 provides results on the final solution as

* Sandia National Laboratories is a multiprogram laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the United States Department of Energy's Nuclear Security Administration under Contract DE-AC04-94AL85000.

well as some comparison results from alternatives that had been considered.

2.0 Background on Shared Objects

2.1 Historical Drivers for Shared Objects

Support for shared objects began appearing in Unix-based systems in the late 1980's. There were three primary motivators for their introduction. 1) Shared libraries can save on physical memory. Frequently used libraries, such as the C standard library can be memory resident and be executed by multiple, disparate processes from the same memory space. 2) Libraries can be updated and fixes applied without significant disruption. It is no longer necessary for every program to be relinked to take advantage of the update. 3) With truly dynamic libraries, a decision can be made at run time as to which shared object is needed by the program.

For systems, such as Cielo, the first advantage is moot. There is only one executable portion of the application binary image on each compute node. It is shared amongst the cores whether it is statically or dynamically linked. The ability to update shared libraries and put them in place without a relink is a two-edged sword. Application results may not be repeatable since libraries may have changed since the last time the application was run. This makes configuration management for applications considerably harder. On the plus side, applications can immediately take advantage of enhancements or bug fixes in a new version of the shared library. The third motivator is very compelling to the scientific community. Applications are becoming very large and at times, exceeding 1GB in size. Developers have been forced to build multiple applications that combine some subset of their total capability. With the use of dlopen(), for example, the application can dynamically determine which functionality to load, based on the input provided at run time.

2.2 Issues for High Performance Computing

One of the differentiators between capability computing and the more ubiquitous capacity cluster computing is the speed and mechanism by which an application is launched on the nodes assigned to the job. Run time software, such as Cray's Application Level Placement Scheduler (ALPS), provides a hierarchical launch distribution and execution of the application binary. When the application is dynamically linked, there is no way under the Linux implementation to collectively, or hierarchically distribute the shared objects to each node. Instead, requests for shared objects must be serviced individually. This can generate a significant I/O metadata server load while the linker searches for libraries. There can be tens of thousands of simultaneous requests for the same set of relatively small files.

As the application continues to run, there is potential for subsequent demand paging of the shared objects as they are utilized during the job. This unpredictable load can have significant impact on the run time of the job as it introduces considerable noise into the completion rate of the application on each processing element. This consideration is particularly important given the bulk synchronous nature of the typical jobs run on MPP systems, such as Cielo.

If other jobs are running on the system at the same time, their I/O performance may be impacted by the random nature of the shared library access. Parallel file systems are not tuned for the sporadic bursts from mmap() calls.

3.0 Approaches for Shared Object Support in CLE

3.1 System DSOs

Originally the Cray XT systems only supported statically linked application binaries on compute nodes. More recent versions of CLE have leveraged the I/O forwarding layer, Data Virtualization Service (DVS) [1], to provide access to system shared objects. Instead of the limited RAMFS-based root file system, each compute node uses DVS to access the same shared root file system as the service (e.g. login) nodes. A pool of DVS nodes NFS-mount the shared root in read-only mode and then project it to the compute nodes in 'loadbalance' mode. This mode allows each DVS node to service a subset of compute nodes. There is no need for a locking protocol since the files are assumed to not change. Each I/O request goes to a DVS server based on the compute node ID, which distributes the load. Caching for data and metadata is done on the client nodes after the first request is satisfied locally. The second tier

of caching is on the DVS servers. Caching is not done by the DVS software, but relies on the capabilities of the Linux kernel.

The number of DVS servers is site-configurable. Starting with CLE 3.1, it is also possible for the DVS servers to be repurposed compute nodes, rather than service nodes [2]. The shared root is NFS-mountable over the high speed network. Figure 2 shows a generic configuration for system shared library support to compute nodes.

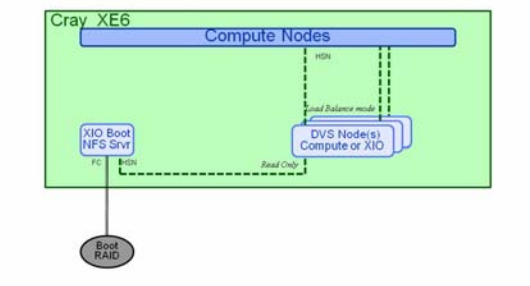

Figure 2: Conceptual diagram of components needed for system shared library access by compute nodes on a CLE system.

3.2 User built DSOs

Cray's solution using DVS for supporting system shared libraries seemed like a very viable starting point for user-built dynamic shared objects. For security and stability reasons, storing application Dynamic Shared Objects (DSO)s with system files is not desirable. Other possible options are: 1) load shared objects from standard locations such as /home or /projects, 2) load shared objects from the parallel (e.g. scratch) file systems, or 3) provide a separate file partition specifically for shared libraries that could be mounted in loadbalance mode.

The home, projects, and scratch file systems are mounted read/write on Cielo. Read-write DVS mounts use a client-server mapping based in part on the file inode number and the offset within the file. For a (typically small) DSO, these will all translate to the same DVS server. On Cielo, the home and projects file systems are only served by eight DVS servers, which is unlikely to be able to service over 8000 nodes for concurrent access to shared libraries. The scratch parallel file system has considerably more DVS servers projecting it to compute nodes. However, the impact of sharing the DVS node cache with potentially heavy I/O traffic patterns from other running jobs was unknown.

The current solution is to provide a dedicated location on Cielo for user shared libraries. Due to concerns of overloading the boot or sdb node, a separate service node was connected to a RAID to store a "/udsl" file system (udsl is an abbreviation for user dynamic shared libraries). This file system is mounted read-write to the login nodes for compilation of the objects. Fifty repurposed compute nodes project in loadbalance (read only) mode the shared root and the /udsl file system to the compute nodes. The number fifty was selected somewhat arbitrary, but it was based on an educated guess that one DVS node could reasonably service 150-200 compute nodes.

Since Cielo has four external login nodes, the service nodes providing /udsl had to be provisioned with two PCI slots—one for the fiberchannel connection to the RAID and one for the Ethernet to connect to the LAN between the external logins and Cielo itself. The /udsl file system is auto-mounted to the external login nodes, so that "df" commands would not hang when Cielo is down for maintenance. This final configuration is shown in Figure 3.

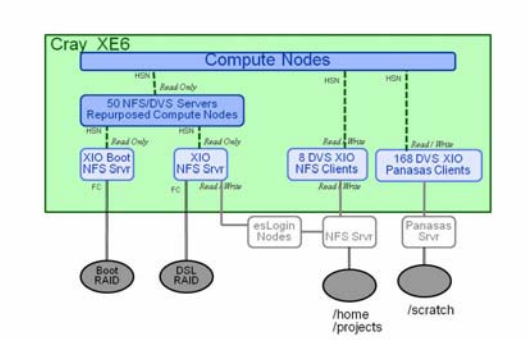

Figure 3: Cielo's file system configuration with support for user dynamic shared libraries.

4.0 Results

In Section 3.2, we presented the three options we considered for supporting user-built shared objects. We selected the third option, but unfortunately they were presented in user convenience order. Most codes are built in users' /home or /project space. The second option of using /scratch space would be somewhat familiar

to the user community. They tend to manage some project files, data bases, and input decks from this area. The /udsl file system is a separate and quite small area. It is less than 1TB in size and must be shared by all users. Entire applications cannot be built in this area. Instead, applications must "install" their shared objects here and ensure careful use of the –rpath option during the link stage and proper setting of the LD_LIBRARY_PATH environment variable.

Since no solution was ideal, we ran tests from each of the alternatives to confirm the need for the /udsl file system. We used the pynamic benchmark [3] as our test case since one of the applications targeted for Cielo has similar characteristics. We did not use the production /home or /projects file system since it is a shared resource across numerous computers and our test could have a very negative impact. Instead, we used a similarly configured (netapp) scratch file system. The results are shown in Figure 4.

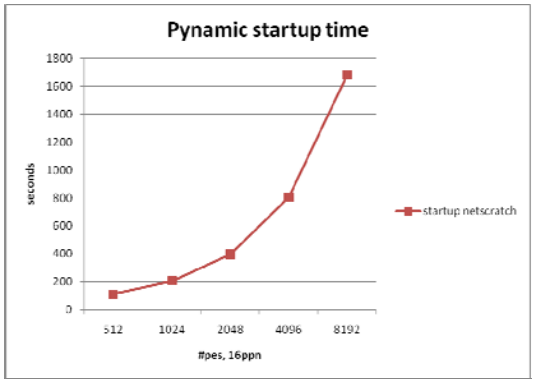

Figure 4: Startup time of Pynamic benchmark from network-attached NFS server.

We cancelled the test rather quickly when the trend showed that the network attached NFS file system would not scale adequately. The graph in Figure 4 only reports the startup time. The actual run time was approximately 2-3 times larger.

For the second test, we used an existing mount of one of the scratch parallel file systems. This would be familiar to the user community as they store the bulk of the data from their current runs in this file system. The /scratch file system is a high end Panasas file system with 72 DVS nodes. The results are shown in Figure 5. Like the results from the network-attached NFS server configuration, the slope of the run times over processing elements was unacceptable.

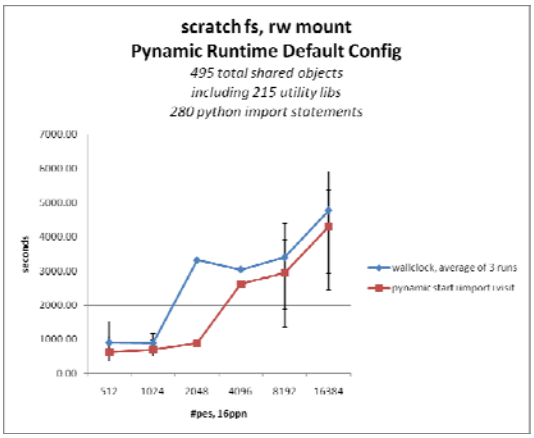

Figure 5: Run time of pynamic benchmark from a Panasas parallel file system.

It is theoretically possible to mount either the network-attached NFS partition or the Panasas file system partition in read-only mode using a second, different, mount point. This would allow the DVS servers to use loadbalance mode and likely provide better and more scalable response times. This option seemed operationally confusing for the users and it was not clear that the DVS has ever been exercised in this way.

Our /udsl results are promising. At 32K processing elements, we can run the default configuration of the pynamic benchmark in about 30 minutes. We have identified some performance/QOS issues in how the dlopen's are serviced on the individual cores of a compute node and are hopeful that a subsequent release of CLE will fix the issue. This should result in even better performance of the pynamic benchmark.

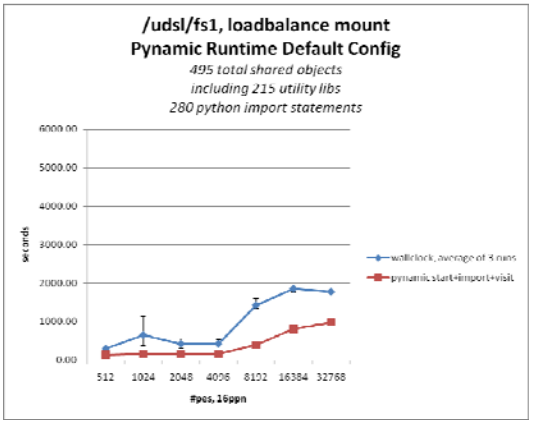

Figure 6: Run time of pynamic benchmark from a dedicated shared library file system.

5.0 Conclusions and Future Work

Statically-linked binaries remain the most viable choice for achieving highly scalable results. When shared objects are mandated by other technical needs, Cray's shared library implementation is a reasonable solution that can be applied to user shared libraries. The use of a separate file system added some operational overhead but isolated the impact of shared libraries to only the applications using them.

At least two outstanding questions remain unanswered with the selected configuration. It is not clear that 50 repurposed compute nodes were required for the DVS shared library servers. The CPU and memory utilization figures on the DVS nodes are very low. Experimentation could identify a more optimal number of DVS nodes. Alternatively, the DVS nodes serving the parallel scratch file system might be able to handle the incremental load from the shared libraries. All of the 50 compute nodes could be reclaimed, if the impact was negligible. However, since parallel file systems tend to be a fragile component of most HPC systems, this option should be carefully studied before being placed into a production environment.

Another possible approach to the problem is to develop a "from scratch" solution to dynamic shared objects for high performance computing. The Unix/Linux implementation was not designed for MPP architectures.

Acknowledgement

The design and implementation of Cielo's user shared library support was accomplished after discussion with colleagues at Cray (David Hensler and Dean Roe), LANL (Parks Fields, Brett Kettering, Jim Lujan, and Josip Loncaric), NERSC (Jeff Broughton, Tina Butler, and Nick Cardo), SNL (Bob Ballance and Doug Doerfler), and ORNL (Don Maxwell). We appreciate their thoughtful and helpful advice.

References

[1] *Introduction to Cray Data Virtualization Service*, Cray Manual S-0005-3102, January 2011.

[2] *Repurposing Compute Nodes as Service Nodes on Cray XE and Cray XT Systems*, Cray Manual S-0029-3101, September 2010

[3] Lee, G.L.; Ahn, D.H.; de Supinski, B.R.; Gyllenhaal, J.; Miller, P., "Pynamic: the Python Dynamic Benchmark," *IEEE 10th International Symposium on Workload Characterization, 2007. IISWC 2007.*, pp.101-106, 27-29 Sept. 2007.