

SANDIA REPORT

SAND2008-3684
Unlimited Release
Printed July 2008

I/O Tracing on Catamount

Ruth Klundt, Marlow Weston, Lee Ward

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

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

Approved for public release; further dissemination unlimited.



Sandia National Laboratories

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

NOTICE: This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government, nor any agency thereof, nor any of their employees, nor any of their contractors, subcontractors, or their employees, make any warranty, express or implied, or assume any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represent that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government, any agency thereof, or any of their contractors or subcontractors. The views and opinions expressed herein do not necessarily state or reflect those of the United States Government, any agency thereof, or any of their contractors.

Printed in the United States of America. This report has been reproduced directly from the best available copy.

Available to DOE and DOE contractors from

U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: <http://www.osti.gov/bridge>

Available to the public from

U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd.
Springfield, VA 22161

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.fedworld.gov
Online order:
<http://www.ntis.gov/help/ordermethods.asp?loc=>

[7-4-0#online](#)



SAND2008-3684
Unlimited Release
Printed July 2008

I/O Tracing on Catamount OS

Ruth Klundt, Marlow Weston, Lee Ward
01422 Scalable Computer Architecture
Sandia National Laboratories
P.O. Box 5800
Albuquerque, New Mexico 87185-MS1319

Abstract

This paper describes a lightweight method for collection of real application I/O traces at scale. An example set of runs is described, and some details of the implementation and trace output format are presented.

ACKNOWLEDGMENTS

We gratefully acknowledge the assistance of Thomas Gardiner and Duane Labreche in identifying and constructing appropriate input decks for running problems under Alegra. We also thank Sue Kelly and Robert Ballance for access to Red Storm.

CONTENTS

Acknowledgments.....	4
Contents	5
Introduction.....	7
Description of the Trace Utility	7
Test Platform.....	8
Application Setup.....	8
Test Results.....	8
Details of the Trace Output Format	9
References.....	12
Distribution	13

INTRODUCTION

Investigation of application I/O access on a machine with tens to hundreds of thousands of clients operating as one coordinated parallel job is problematic. The typical methods for collection of I/O trace information have a number of drawbacks in terms of overhead, scalability, ease of use, and portability. Our colleagues at Los Alamos National Labs have categorized an array of the typical methods in use up to this point. [3] There is no instance of this type of tracing information having been collected successfully from a real user application at large scale.

DESCRIPTION OF THE TRACE UTILITY

In order to provide a tool for file system developers and administrators to acquire good understanding of usage modes on large scale machines, we have implemented a tracing utility to be used with the Catamount LWK [1] which runs on Red Storm, a Cray XT3 capability class machine¹. Catamount itself provides no directly user accessible I/O capabilities as it is a custom microkernel. Application I/O is implemented by means of the user level library sysio². This library provides a virtual file system which allows application access to parallel file systems and to standard I/O. Each library call provides a hook at entry and exit. If tracing is not initialized, the hook does nothing.

When the tracing functionality is activated, traversal of an entry or exit hook triggers an event, which consists of a call into the tracing code, passing a record of qualifying information for the call. Each event results in the encoding of the call, type of hook, and the qualifying information into a buffer in the tracing code. The design includes double buffering and use of asynchronous write calls to dump a buffer when full³.

File system interactions of the tracing utility itself are not traced. This is accomplished by using lower level calls into sysio for file open, write and close. On detection of internal error conditions the tracing halts, and allows the application to continue without interruption if possible. If the application generates events at a rate that causes both buffers to fill before one can be dumped, the tracing code begins to drop events keeping a count of the drops. When a buffer becomes available again, the number of dropped events is recorded as part of the trace output.

Each process in the application job generates one file containing the I/O traces for that client encoded in an efficient, platform independent binary format. A dictionary describing the binary file format must be generated using a provided external utility on the run platform. The resulting traces can be decoded into readable format by using the dictionary and the provided binary decoder on any machine.

¹ <http://cray.com/products/xt4/index.html>

² <http://sourceforge.net/projects/libsysio>

³ Asynchronous behavior is dependent on the capability of the underlying file system where the trace data is written.

TEST PLATFORM

Some initial trace sets at client sizes of 2744 and 5832 have been generated using Alegra [2], a premier Sandia National Laboratories simulation application, on Red Storm. The test platform is a Cray XT3 machine running the Catamount LWK on the compute partition. Software release version at the time of the runs was Cray Unicos/lc 2.0.41. Tests were run on a partition of the machine having a total of 3360 compute nodes available, each one providing a Dual-Core AMD 64 bit Opteron™ processor and 2 or 4 GB of RAM.

APPLICATION SETUP

Tracing functionality is activated via an environment variable which provides a file system path for the trace dumps, and a buffer size for the caching of trace data. Currently we require a re-link of the application which includes the new library and a global constructor on the link line. The constructor reads the environment variable and passes the value to the tracing initialization function. If no call is made to initialize tracing, the trace functions are not entered at all during execution. Our selection of run sizes was dictated by the problem sets available for Alegra.

A physics simulation problem typical of Alegra usage was employed for the tracing runs. The runs were performed alongside regular user runs, not during dedicated system time. Runs were done in two modes, with either one or two cores per node, and were repeated with and without tracing enabled. Trace data was written to a subdirectory of the jobs' default output directory. Trace data buffer size was set to 2 MiB⁴.

TEST RESULTS

Each run generated 4 restart dumps, and ran for 20 simulation cycles. Trace dump file sizes were in the range of 30-50 KiB per process, except for the root process which records progress in the job via stdout. The extra output resulted in trace sizes of 300-600 KiB from the root process only. The trace output from these runs has been released.

⁴ <http://physics.nist.gov/cuu/Units/binary.html>

DETAILS OF THE TRACE OUTPUT FORMAT

Each process in the parallel job generates a set of trace events. The final translated output is the concatenation of all the trace sets, and is in human readable format. Each set of per-client trace events begins with a header line with the following format (bold indicates keywords):

header(*headerbom*,**headernode**(*node_nid.pid*)**headerlength**(<bytes>))

The fields are currently meaningless other than the identification of the node where the trace events in this set were generated by *node_nid.pid*.

All other contents of the final output have the following format:

tracetype(**ENTER/EXIT**)**time**:(*secs*)**time**:(*msecs*)**str**(<*event_name*>)<*infolist*>

Each system call which occurs in the traces will have an Enter/Exit pair present. The time fields above denote seconds and microseconds since the Epoch (00:00:00 UTC, January 1, 1970) as returned by the `gettimeofday` call.

The <*infolist*> contains a hierarchically arranged collection of qualifying information for the particular I/O system call and varies according to the call.

A simple example is the event for entry to the `open` syscall. The trace event contains, after the timestamp, the name of the call, a string with a sanitized version of the pathname, and the incoming values for open flags and mode.

tracetype(**ENTER**)**time**:(1205528630)**time**:(529780)**str**(open)**str**(/uneYK8P4/rXGyxud/HeiDle/iH6EljFSS/xehVgST74jJGO/XMbsHdI/rd3S/InUrvrGZ3i5NrU/tSn319JlbbZkCBaJ/TbVO3MXV isKt/DlG2/My87pEl28ixAx5rVhMe3qXQCyKnv)**flags**(578)**mode**(436)

A more complex example is the `stat` call, which resolves to a call to `fxstat` within `libsyo`. Here are the Enter and Exit events for that call:

tracetype(**ENTER**)**time**:(125528630)**time**:(531185)**str**(fxstat)**ver**(1)**fd**(3)
tracetype(**EXIT**)**time**:(125528630)**time**:(531368)**str**(fxstat)**return**(0)**stat**(st_dev(0)st_ino(10702772)st_mode(33204)st_nlink(1)gid(41776)uid(41776)st_rdev(0)st_size(0)st_atime(1205528558)st_mtime(1205528630)st_ctime(1205528630)st_blksize(2097152)st_blocks(0))

The Enter event for `fxstat` provides the incoming values of the version and file descriptor. The Exit event provides a return code, and the contents of the `stat` struct items which have been acquired by the call. The item names are taken from the `stat` struct definition of the machine where the traces have been generated.

The call names in the trace events reflect the actual API call within the sysio library where the trace event is being recorded. Generally these follow the POSIX⁵ definitions and should be self-explanatory.

Some calls are specific to libsysio. For example, all I/O transfers in the released traces are implemented at the lowest level with the asynchronous set of calls `ireadv`, `iwritev`, and `iowait/iodone`⁶. The `ireadv` and `iwritev` trace events provide information identifying the file descriptor, the pointer to an I/O vector, and the length requested for the transfer. The request is queued, to be completed asynchronously if possible, and a call to `iowait` completes the transfer. Alternatively an application can check the status of the transfer using `iodone`. The return code in the `iowait` Exit event reports the number of bytes transferred. Here is an example of the sequence of trace events associated with a write call:

```
tracetype(ENTER)time:(125528630)time:(586386)str(iwritev)fd(3)ziovec(base(0x200EFDC0)len(524288))  
tracetype(EXIT)time:(125528630)time:(590489)str(iwritev)ptr(0x200B1110)  
tracetype(ENTER)time:(125528630)time:(590490)str(iowait)ptr(0x200B1110)  
tracetype(EXIT)time:(125528630)time:(590530)str(iowait)return(524288)
```

Also provided are a set of asynchronous vector I/O calls which perform extent based transfers. Both `ireadx` and `iwritex` calls take as input a file descriptor, a list and count of memory specifications (struct `iovec *`) and a list and count of extent specifications (struct `xtvec *`). The return value is an `ioid_t`. Transfer completion is tested by a call to `iowait` or `iodone`, passing the `ioid_t` value returned by `ireadx` or `iwritex`. The extent specifications are (offset, length) pairs and define the locations in the file involved in the data transfer, as follows:

```
struct xtvec {  
    off_t  xtv_off; /* Stride/Extent offset. */  
    size_t xtv_len; /* Stride/Extent length. */  
};
```

The `ireadx/writex` calls traverse the memory specification list and file extent list independently, performing transfers as directed, until one or the other of the lists is exhausted. Here is an example of the trace events produced by `iwritex` and the corresponding `iowait`:

```
tracetype(ENTER)time:(1216152765)time:(250413)str(iwritex)fd(1)count(4)ziovec(base(0x537D50)len(10),base(0x538590)len(10),base(0x537D70)len(10),base(0x538BB0)len(17))count(4)yxtvec(off(0)len(15),off(20)len(5),off(30)len(10),off(50)len(10))  
tracetype(EXIT)time:(1216152765)time:(250487)str(iwritex)ptr(0x538DF0)  
tracetype(ENTER)time:(1216152765)time:(250505)str(iowait)ptr(0x538DF0)  
tracetype(EXIT)time:(1216152765)time:(250521)str(iowait)return(40)
```

The above trace indicates that the `iwritex` call sourced data from four buffers of length 10, 10, 10, and 17, and sent the data to four locations in the file, specified by (offset, length) pairs (0,15),

⁵ IEEE Std 1003.1 available at <http://standards.ieee.org/>

⁶ man pages for the calls `ireadv`, `ireadx`, `iwritev`, `iwritex`, `iodone`, and `iowait` are available on Cray XT3 machines

(20,5),(30,10),(50,10). The Exit event from the iowait call reported 40 bytes transferred. This implies that 7 bytes in the last buffer went unused, which is correct since the extent specification was exhausted before using all data in the memory specification.

REFERENCES

1. Kelly, S.M., Brightwell, R.B.. Software Architecture of the Lightweight Kernel, Catamount. In *Proceedings of the 2005 Cray Users' Group Annual Technical Conference*, Albuquerque, New Mexico, May 2005.
2. Alegra: Shock and Multiphysics Family of Codes
http://www.cs.sandia.gov/ALEGRA/Alegra_Home.html
3. Konwinski, A., Bent, J., Nunez, J., and Quist, M. 2007. Towards an I/O tracing framework taxonomy. In *Proceedings of the 2nd international Workshop on Petascale Data Storage: Held in Conjunction with Supercomputing '07* (Reno, Nevada, November 11 - 11, 2007). PDSW '07. ACM, New York, NY, 56-62.
4. Roth, P. C. 2007. Characterizing the I/O behavior of scientific applications on the Cray XT. In *Proceedings of the 2nd international Workshop on Petascale Data Storage: Held in Conjunction with Supercomputing '07* (Reno, Nevada, November 11 - 11, 2007). PDSW '07. ACM, New York, NY, 50-55.
5. Hedges, R., Loewe, B., McLarty, T., and Morrone, C. 2005. Parallel File System Testing for the Lunatic Fringe: The Care and Feeding of Restless I/O Power Users. In *Proceedings of the 22nd IEEE / 13th NASA Goddard Conference on Mass Storage Systems and Technologies* (April 11 - 14, 2005). MSST. IEEE Computer Society, Washington, DC, 3-17
6. Miller, E. L. and Katz, R. H. 1991. Input/output behavior of supercomputing applications. In *Proceedings of the 1991 ACM/IEEE Conference on Supercomputing* (Albuquerque, New Mexico, United States, November 18 - 22, 1991). Supercomputing '91. ACM, New York, NY, 567-576

DISTRIBUTION

1	MS1319	Lee Ward	1422
1	MS1319	Marlow Weston	1422
1	MS1319	Ruth Klundt	1422
2	MS0899	Technical Library	9536



Sandia National Laboratories