Field Session 2009
Los Alamos National Laboratories


## Abstract

The Los Alamos National Laboratories' global parallel supercomputing network consists of tens of thousands of processors and upwards of 4 PB of data storage. Due to the large number of scientists running applications and the nature of large scale parallel computing, the increased probability of unrecoverable bit failure presents a significant problem. The solution to this is checkpointing, the process in which data is routinely dumped from memory to disk. There are three primary storage methodologies: N-N, N-1 strided, and N-1 non-strided. LANL needs a record of different applications checkpoint, as it is difficult to tell the format without in depth investigation of the code. Our project is to create a process to identify size, number of I/O calls, and the file storage methodology of I/O calls for Los Alamos National Laboratories' global parallel supercomputing network. Our team will create a kernel module to trace and log this information without requiring the alteration of applications making the I/O calls. This will be implemented using the FUSE libraries.

Field Session 2009
Los Alamos National Laboratories
Andrew Bowling, Jason Hopkins, Shannon Poole
Final Report Document

## Introduction

Los Alamos National Laboratory was founded during World War II as a secret facility to coordinate research of the Manhattan Project. It is one of the largest science and technology institutions in the world, specializing in research for national security, weapons systems, outer space, renewable energy, medicine, nanotechnology, and supercomputing.

Due to the increasing scale of scientific computation required for such research, there is a need for a rapidly increasing storage capacity for High Performance Computing (HPC) and Data Intensive Super Computing (DISC). Because of this, individual disk storage devices are rapidly getting denser while their bandwidth and processing speed are not growing at the same rate. With storage systems for HPC and DISC environments exceeding 50,000 disk drives involved in one parallel job, there exists a desire to efficiently utilize the massive resources of these tasks using more serial I/O workloads. Los Alamos utilizes a Redundant Array of Inexpensive/Independent Disks (RAID) data storage scheme to store this massive amount of data and protect against data loss in the event of hardware failure. When a file is written to the RAID, the controller writes portions of the file to different disks. In the event of disk failure, only a portion of the data is lost and can usually be reconstructed.

Checkpointing is the periodic backup of an application's processes to one or more files in the eventuality of server failure, which occurs every six hours on average. Los

Alamos primarily uses three different types of checkpointing formats: N-N, N-1 non-strided, and N-1 strided. The N-N method of data storage writes N processes to N unique files, each containing all of that process' data members. N-1 non-strided writes all the process information sequentially to a single file, while in N-1 strided all the information is written such that all similar data members from each process are grouped together within the file.

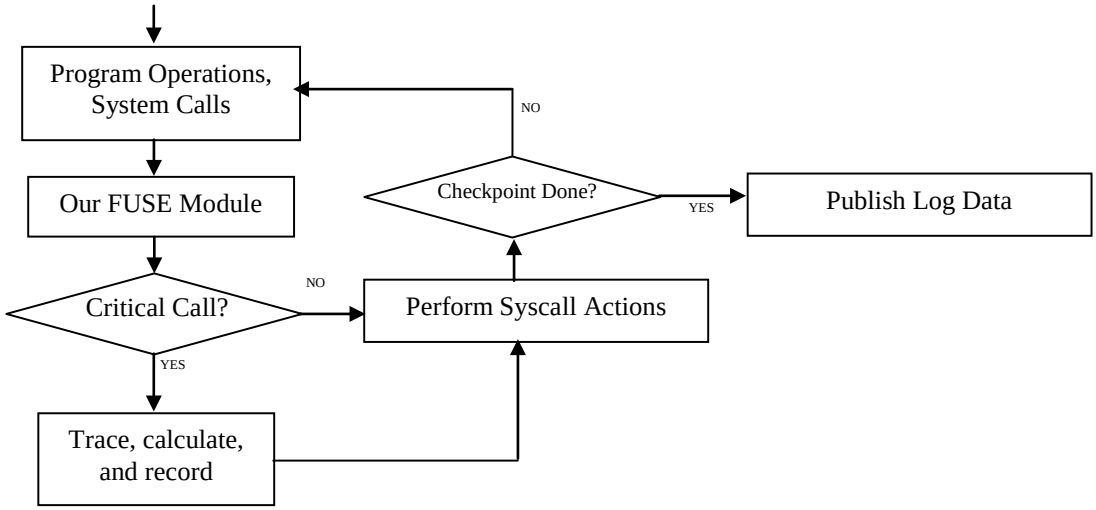
Scope and Requirements

One of the most important tools for helping storage researchers understand I/O workloads is utilizing tracing, which is needed to provide information about I/O workloads when researchers do not have the ability to run applications directly. However, I/O tracing an application in a large parallel machine is difficult, since oftentimes applications and processes storage varies from project to project, making one I/O tracing method unable to work in every circumstance and requiring information about the checkpointing format to decide which tracing method to use.

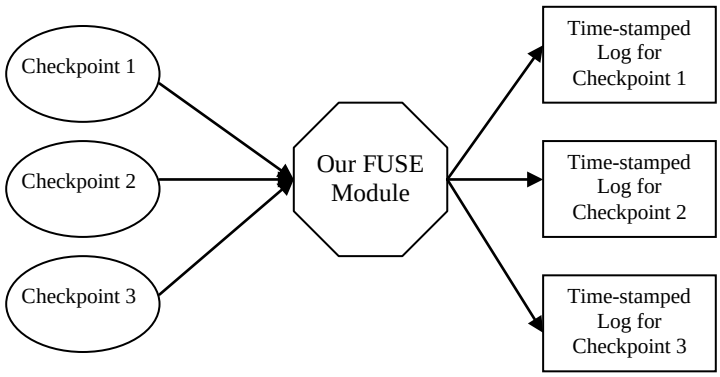
The end product of this project was to build a mountable file-based I/O tracing utility that meets several requirements. The I/O tracing utility had to be able to track and log I/O calls, identifying the specific storage format, data chunk size, and number of operations for any application utilizing it, and then outputting this information to a file. The product needed to contain extremely lightweight code with a very low overhead and not require any code changes to future or existing applications, with the only permissible change being the modification of the output directory. Our tracing utility also had to be able to run on any Linux operating system, and only leverage open source or original code.

Design:

Our module monitors the kernel for system calls and performs preprocessing functions for key calls without interfering with their functionality. Once a checkpoint is complete the module will log the collected data. Below is a control-flow diagram detailing the process.




Our module will perform these functions for any number of consecutive checkpoints, with each checkpoint's data being logged to a separate file. Simultaneous checkpointing is not currently supported.



Implementation and Results

In order to accurately trace the I/O calls of an application's processes, our tracing utility accesses the process information during checkpointing before it is written to disk. Using the FUSE library, which stands for Filesystem in Userspace, we implemented a fully functional file system that can intercepted I/O calls as middleware, before the process' checkpointing was completed. The FUSE file system was effective for this project because all system calls executed on the system first pass through the mounted module's interface. Each syscall is routed to a unique function that first performs any necessary default actions, and then executes additional actions specified by the module. It is within these additional actions that this module's tracing and calculations occur. Because of the nature of FUSE, we have to implement functions for many syscalls in addition to the ones needed to make the necessary calculations (open, read, write, and release), but they contain the absolute minimum code needed to maintain functionality. Within these algorithms, the utility gathers the data required for the trace (I/O count, checkpoint I/O methodology, and data chunk size) and then writes this information to a separate log file given a time stamp as its filename. One option we have implemented as a command line argument is the suppressing of the write function during checkpointing to avoid taking the time to do so during testing. Below is a sample log output for a 100 processor simulation with 10 data members each.

```
Fri Jun 12 13:52:44 2009
        I/O Count: 16000
        Chunk Size: 4096
        Total Bytes Written: 65536000
        Checkpointing type is N-1.
        Data is non-strided.
```

## Conclusions

One the biggest challenges we had was learning how FUSE worked and how to leverage FUSE to intercept and interpret necessary system calls. We also became better acquainted with C and developing modules within the Linux operating system. As a challenge mostly unrelated to our work with FUSE, we also had to learn how to implement the Message Passing Interface (MPI) to simulate the massive parallel computing environment used by Los Alamos National Laboratories. Our project required constant communication with our client via e-mail in order to accurately understand and modify the requirements. We faced several difficulties trying to get the Fedora-based machine we needed for this project to work correctly, including obtaining the required privileges in order to install the dependencies needed to use FUSE and MPI. In the end we persevered and delivered our I/O tracing utility to our client for testing.

## Future Directions

While our project meets the client's expectations and needs, there is room for additional future development and enhancements. Support for simultaneous checkpointing could be implemented, for example. Additionally, specifying a different log directory and/or publishing data to the /proc file system could also be implemented. Upon completion of these and/or other additional functionalities, LANL will, at their discretion, publish our code to the open source community.

Glossary

HPC – High Performance Computing, the use of supercomputers and computer clusters to solve advanced computation problems.

DISC – Data Intensive Super Computing, maintains continually changing data sets in addition to performing large-scale computations on the data

RAID – Redundant Array of Inexpensive/Independent Disks, a data storage methodology in which many hundreds of individual hard disks are used to store data and maintain data integrity in spite of hardware failures.

FUSE – Filesystem in Userspace, a collection of libraries that allow for the mounting of virtual file systems which can monitor and/or modify system calls on the kernel side, after they have been issued by system drivers.