1st International Conference on Recent Trends in Engineering & Technology, Mar-2012
Special Issue of International Journal of electronics, Communication & Soft Computing Science & Engineering, ISSN: 2277-9477

Icra – a Unified Tool to Model the Performance of Linux Block Device Layer using Queuing Theory

Apoorva Pathak **Reshma Raju** **Ashish Kumar Singh** **Aarti Goel**

Abstract: In recent years, Linux block device layer has evolved into an efficient design and implementation. However, the improvements like request queues, IO schedulers, mapped device framework, etc have made it sufficiently complicated to understand and model this layer. A clear understanding of block device layer becomes critical for developing scalable and high performing applications on top of it. This complexity urged us to develop an integrated tool – Icra. Icra is a self- contained tool which is developed by integrating Linux tools such as fio, blktrace, btt and blkparse. We would also attempt to come up with general guidelines for choosing block device configuration for a particular IO load and pattern.

Index Terms — blkparse, blktrace, btt, device driver ,fio .

I. Introduction

An operating system is an important part of every computer system, it acts as an intermediary between a user and the computer hardware, managing resources and allocating them to specific programs. A device driver is basically a program which provides an easy to use and common interface to a hardware device while protecting the hardware from bad user programs. A block driver provides access to devices that transfer randomly accessible data in fixed-size blocks—disk drives, primarily Device drivers are located in the heart of the Linux OS - The kernel Linux is aimed at the grey area that lies between desktop computers and distributed embedded systems, Linux boxes are extremely fast, because the operating system is very efficient at managing resources such as memory, CPU power and disk space.

Linux is *only the operating system kernel*, code written from scratch, which supports 32-bit and 64-bit multitasking, virtual memory, shared libraries and multi-user capabilities. The Linux kernel sees block devices as fundamentally different from character devices, as a result block drivers have a distinct interface and their own particular challenges. Device drivers are part of the Linux kernel and are responsible for the communication between hardware and other parts of the kernel and user software, respectively. Much of the design of the block layer is centered on performance. Many char devices can run below their maximum speed, and the performance of the system as a whole is not affected. The system cannot run well, however, if its *block I/O subsystem* is not well-tuned.

The Linux block driver interface allows you to get the most out of a block device but imposes, necessarily, a degree of complexity that you must deal with, the 2.6 block interface is much improved over what was found in older kernels. If your device uses a different size, the kernel adapts and avoids generating I/O requests that the hardware cannot handle. There are large number of devices like keyboard, mouse, hard drives, sound cards, printers , PCI, USB etc. Without device driver programs it would be difficult to talk to these devices on an individual basis. We intend to develop Icra so that no user is deprived of the versatility of the Linux block device layer.

II. Fundamentals of Icra

A. Linux Operating System

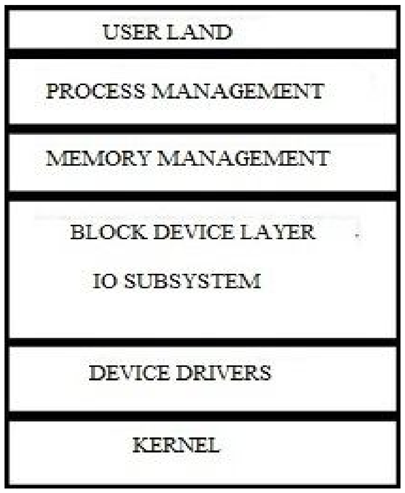

Fig. 1 Split view of Linux Operating System

The layout of Linux operating system is as follows:

a. User Land

The user land comprises of the user defined applications and user interfaces which are provided to access the services of operating system.

b. Process Management

A process needs certain resources including CPU time, memory, files and IO devices to accomplish its task. The management of these resources by the operating system is termed as process management.

c. Memory Management

Main memory is a repository of quickly accessible data shared by the CPU and IO devices. The central processor needs instruction from main memory during the instruction fetch cycle and reads and writes data from main memory during data fetch cycle.

d. Block Device Layer

This layer is of main concern for Icra as it comprises of the IO subsystem layer.IO subsystem is generally the messiest part of the operating system.IO is the major factor in overall system performance. Improving IO performance is a major operating system challenge.

e. Device Drivers

A device driver is the set of kernel routines that makes a hardware device respond to the programming interface defined by the canonical set of VFS functions that control a device. The actual implementation of all these functions is delegated to the device driver.

f. Kernel

In computing, the kernel is the main component of most computer operating systems .It is a bridge between applications and the actual data processing done at the hardware level. Usually as a basic component of an operating system a kernel can provide the lowest level abstraction layer for the resources that application software must control to perform its function.

B. Request Queue

When a kernel component wishes to read or write some disk data, kernel actually creates a block device request. However, the kernel does not satisfy the request as soon as it is created. The IO operation is just scheduled which will be performed at a later time. This artificial delay is paradoxically the crucial mechanism for boosting the performance of block devices. Each block device driver maintains its own request queue, which contains the list of pending request for the device. Essentially, a request queue is a doubly linked list whose elements are request descriptors. The ordering of the elements in the queue list is specific to each block device driver.

C. IO Scheduler

When a new request is added to a request queue, the generic block layer invokes the IO scheduler to determine that exact position of the new element in the queue. The IO scheduler tries to keep the request queue sorted sector by sector. Currently, Linux 2.6 offers four types of IO schedulers :

a. The “Noop” elevator

This is the simplest IO scheduling algorithm. There is no ordered queue: new request can be added either at the front or at the tail of the dispatch queue.

b. The “CFQ” elevator

The main goal of “Complete Fairness Queuing” elevator is to ensure fair allocation of the disk IO bandwidth among all the processes that trigger IO requests. To achieve this result, the elevator makes use of 64 sorted queues to store the request coming from different process.

c. The “Deadline” elevator

The “Deadline” elevator makes use of four queues. Two of them are sorted queues include read and write requests, respectively ,ordered according to their initial sector numbers. The other two are the deadline queues which include same read and write requests sorted according to their “deadlines”. These queues are introduced to avoid request starvation. A request deadline is an expire timer that starts ticking when the request is passed to the elevator. By default the expire time of read request is 500ms,while the expire time for write request is 5s.

d. The “Anticipatory” elevator

Basically .it is an evolution of the “Deadline” elevator. The key difference between “Deadline” elevator and “Anticipatory” elevator lies in the expire time. In “Anticipatory” elevator the default expire time for read request is 125ms, while the default expire time for write request is 250ms.Thus, the “Anticipatory” elevator is the most sophisticated IO scheduler algorithm offered by Linux.

III. Design and Implementation Constraints

The design of Icra is as shown in the diagram below:

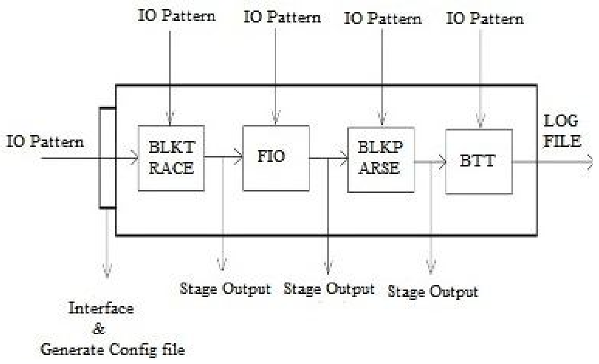

Fig. 2 Design Of Icra

Icra involves integrating the following tools:

A. Fio (Flexible IO tester)

Fio is a tool that will spawn a number of threads or processes doing a particular type of I/O action as specified by the user. The typical use of fio is to write a job file matching the I/O load one wants to simulate.

a. Job File Format

Job files are in 'ini' format. They consist of one or more job definitions, which begin with a job name in square brackets and extend to the next job name. The job name can be any ASCII string except 'global', which has a special meaning. Following the job name is a sequence of zero or more parameters, one per line, that define the behavior of the job. Any line starting with a ';' or '#' character is considered a comment and ignored. If *job file* is specified as '-', the job file will be read from standard input.

b. Global Section

The global section contains default parameters for jobs specified in the job file. A job is only affected by global sections residing above it, and there may be any number of global sections. Specific job definitions may override any parameter set in global sections.

B. Blktrace (Block Trace)

Blktrace is a block layer IO tracing mechanism which provides detailed information about request queue operations up to user space. There are three major components: a kernel component, a utility to record the i/o trace information for the kernel to user space, and utilities to analyze and view the trace information. The blktrace utility extracts event traces from the kernel. Some background details concerning the run-time behavior of blktrace are :

i. blktrace receives data from the kernel in buffers passed up through the debug file system.
ii. blktrace defaults to collecting all events that can be traced.
iii. blktrace stores the extracted data into files stored in the local directory.
iv. The default behaviour for blktrace is to run forever until explicitly killed by the user (via a control-C, or sending SIGINT signal to the process via invocation the *kill (1)* utility). Also you can specify a run-time duration for blktrace via the **-w** option -- then blktrace will run for the specified number of seconds, and then halt.

C. Blkparse (Block Parse)

The blkparse utility will attempt to combine streams of events for various devices on various CPUs, and produce a formatted output of the event information. Some background details concerning the run-time behavior of blkparse are :

i. By default, blkparse expects to run in a post-processing mode – one where the trace events have been saved by a previous run of blktrace, and blkparse is combining event streams and dumping formatted data.
ii. blkparse may be run in a liv*e* manner concurrently with blktrace by specifying -*i* - to blkparse, and combining it with the live option for blktrace.
An example would be:
% blktrace -d /dev/sda -o - | blkparse -i –
iii. By default, blkparse sends formatted data to standard output.

D. Btt (Block Trace Timeline)

Btt is a post-processing tool for the block layer IO tracing tool called blktrace. Btt will take in binary dump data from blkparse, and analyze the events, producing a series of output from the analysis. It will also build .dat files containing "range data" -- showing things like Q activity (periods of time while Q events are being produced), C activity (likewise for command completions), and etc. Included with the distribution is a simple 3D plotting utility, *bno_plot*, which can plot the block numbers btt outputs if the -*B* option is specified. The display will display each IO generated, with the time (seconds) along the X-axis, the block number (start) along the Y-axis and the number of blocks transferred in the IO represented along the Z-axis.

IV. Product Functions

i. Opal will provide a GUI to the end user for selecting the various tunable parameters.

ii. The user is then presented with a configuration file

iii. (that will be generated by the tool) which contains the current status of the tunable parameters provided by the end user.

iv. After the processing is complete , the user is provided with the details of the resources which were utilized during the processing of IO pattern in the form of the log file. The resources are:

- CPU time
- Number of request queues
- Time taken by each request queue
- Seek time
- Response time

V. Conclusion

In this paper we have explored the functionality of the major open source tools that affect the Linux block device layer, and suggested the details of the tool that we intend to come up with so as to make this task easier. This would in turn make it easier to trace the block layer operations and will help to build applications on top of this layer, We have shown that there is potential for improvements in modelling the Linux block device layer. Thus, we have introduced an integrated tool, and described its potential.

Acknowledgment

We would like to acknowledge the motivation and support provided to us by Prof.N.F.Shaikh, our Internal Project Guide (Head of Department of the Computer Dept.). Their efforts towards helping us in our work even in their busy schedules are highly appreciated.

We would also thank lab assistants for helping us out with our problems in lab .Taking this opportunity we would also like to thank to the staff of MESCOE (Modern Education Society's College of Engineering) library that have always helped us to find all the material related to Icra.

References

[1] Bovet &Macro, Understanding the Linux Kernel,O'Reilly,2001.
[2] Robert Love. Linux Kernel Development, pages 245-249 Novell Press, second edition, 2005.
[3] Avantika Mathur, Mingming Cao, Suparna Bhattacharya, Andreas Dilger, Alex Tomas, and Laurent Vivier. The new ext4 filesystem: current status and future plans. In Proceedings of the Linux Symposium,.
[4] Stephen C. Tweedie. Journaling the Linux ext2fs filesystem. In The Fourth Annual Linux Expo, Durham, North Carolina, May 1998.

Mr. Ashish Kumar Singh
Bachelors in Computer Engineering
At Modern Education Society's College of Engineering.Pune-411001.
Contact:8983315075
Email_Id: ashish061291@gmail.com

Ms.Aarti Goel
Bachelors in Computer Engineering
At Modern Education Society's College of Engineering.Pune-411001.
Contact:8968845150
Email_Id: 037aarti@gmail.com

Ms.Reshma Raju
Bachelors in Computer Engineering
At Modern Education Society's College of Engineering.Pune-411001.
Contact:8149439412
Email_Id: reshma.raju021@gmail.com

Ms. Apoorva Pathak
Bachelors in Computer Engineering
At Modern Education Society's College of Engineering.Pune-411001.
Contact:9028989561
Email_Id: apoorva.pathak10@gmail.com