Reliability analysis of ZFS.

Asim Kadav and Abhishek Rajimwale

November 2, 2007

Abstract

This project performs a reliability analysis of Sun's ZFS. ZFS is a next generation file system that provides simplicity, transactional semantics and immense scalability. ZFS also claims to be very robust and provides "end to end" data integrity with features like storing checksum of data recursively in a tree format and data self-healing facility. We aim to perform a suitable reliability analysis of ZFS to test how robust this corruption handling technique is. We also infer important lessons from this exercise on the protection mechanisms of ZFS from various types of data corruption.

1 Introduction

Pointers are extensively used in creating interesting and efficient data structures. All operating system components including file systems use pointers to maintain data. File systems extensively use on-disk pointers to make data available in files. Due to the inherent unreliable nature of disks, these on-disk pointers are susceptible to corruption and the reliablity of the file systems depends on the detection and recovery from such pointer corruptions.

ZFS claims to provide novel and robust reliability mechanisms like using checksums in a self validating tree structure. In this project, we aim to perform a type aware pointer corruption analysis of ZFS and evaluate the need of further types reliability analyses. In particular, by performing pointer corruption analysis we can reduce the large search space of data corruption possible on disk to a representative subset of interesting faults possible and analyse the file system's policy to deal with them.

2 Existing design and claims

ZFS uses the concept of a common storage pools for different filesystems on a system instead of using the traditional concept of volumes. The pool can be viewed as a giant tree sized structure comprising of data at its leaf nodes and metadata at its interior nodes. The stroage pool allocator which manages this pool is responsible for data integrity amongst its host of other functions. This allocator uses a strong checksum mechanism for detection of corruption. For each node (of the tree structure) in the storage pool, the 64 bit checksum of all its children(block pointers) are stored. This mechanism has several advantages like eliminating separate I/O for fetching checksums, fault isolation by separating data and checksum. It also gives a mechanism to store checksum of checksums because of the inherent tree nature of the storage pool. ZFS uses 64-bit second order Fletcher checksums to all user data and 64-bit fourth order Fletcher checksums to all metadata. Using this checksum mechanism, ZFS is able to detect and correct bit rot, misdirected reads, phantom writes, misdirected writes and some user errors. In addition to the above mechanism, ZFS also provides automatic repairs in mirrored configurations and also provides a disk scrubbing facility to detect the latent sector errors while they are recoverable [2]. By providing these comprehensive reliability mechanisms, ZFS claims to eliminate the need of fsck. To summarize, ZFS presents a data integrity model comprising of an always valid on-disk state therby giving "no-window of vulnerability".

3 Related work

There has been significant research in failure and corruption detection for file systems and many fault injection methodologies and techniques have been developed [1] [5] . The techniqies range from dynamic injection of errors [4] to static and model checking errors [6] for detection of failure points and scenarios. Our study of ZFS uses type information for fault injection to understand the failure behaviour of file systems. ZFS also provides a ztest utility to perform various types of tests including data corruption and is shipped with the file system [3] .

4 Methodology of evaluation

We develop a block device called as the “corrupter” driver which is a standard Solaris block device driver exposing the LDI (Layered Driver Interface). This block device interposes between the ZFS Virtual devices (VDEVs) and the disk driver beneath. The kernel interface of the LDI exposed by the corrupter driver is called in by the VDEVs to perform the actual I/O for the ZPOOL. The corrupter in turn calls into the actual disk driver using its LDI for performing the I/O. At the interception, the corrupter understands the semantics of the physical blocks which are being given by the VDEV for I/O to disk. It identifies the set of block and pointers to corrupt on a read. When these blocks are read by ZFS, the corrupter driver on interception corrupts the identified pointer.

We use an application as the “filesystem consumer” which calls into ZFS using the POSIX filesystem APIs. For different filsystem operations performed by our application such as a mount, read, write, the corrupter driver injects the faults in the blocks read by ZFS. We then analyse the behavior of ZFS to recover from such faults. Typically when writes are performed back to the identified blocks, the corrupter driver monitors these writes and doesn’t further corrupt them on reads. This behavior is representative of the transient errors on disks. To analyse the behavior of ZFS, we see the return values, system logs and also trace the the system calls using the Solaris dtrace.

5 Expected conclusions

The pointer corruption detection and recovery techniques of ZFS under various scenarious are studied. The conclusions will be drawn on the robustness of its fault handling techniques according to its behaviour on our tests.

References

[1] J. H. Barton, E. W. Czeck, Z. Z. Segall, and D. P. Siewiorek. Fault Injection Experiments Using FIAT. In *IEEE Transactions on Computers*, 1990.

[2] J. Bonwick. ZFS(Zettabyte FileSystem). In *One pager on ZFS: http://www.opensolaris.org/os/community/arc/caselog/2002/240/onepager/*, 2002.

[3] M. Byrne. ZTEST. In *http://www.opensolaris.org/os/community/zfs/ztest/*.

[4] W. Gu, Z. Kalbarczyk, R. K. Iyer, and Z. Yang. Characterization of Linux Kernel Behavior under Errors, (DSN’03). In *2003 International Conference on Dependable Systems and Networks*, page 459, 2003.

[5] G. Kanawati, N. Kanawati, and J. Abraham. FERRARI: a flexible software-based fault and error injection system. In *Transactions on Computers.*, 1995.

[6] J. Yang, P. Twohey, B. Pfaff, C. Sar, and D. Engler. EXPLODE: A Lightweight, General Approach to Finding Serious Errors in Storage Systems. In *In OSDI ’06, Seattle, WA*, November 2006.