A File System Optimized for Solid State Hard Drives

UNIVERSITY OF BATH
Department of Computer Science

Student: *Jordan Dinwiddy*

Supervisor: *Professor John Fitch*

Abstract

*This project investigates a number of techniques for improving file system performance on **Solid State Drives (SSDs).** The results of this work are demonstrated in a new simple file system called **Somix**.*

Introduction

Most computer file systems in use today have been developed with the inherent assumption that the underlying storage device be mechanical in nature. As a result such file systems incorporate clever optimisations targeting the unique physical properties of these devices. The use of these file systems on SSDs often leads to sub-optimal performance as a result of many such properties having changed. Further still, such changes give rise to the possibility of using new, SSD specific optimisations. This project discusses both the features of traditional file systems that no longer apply to these newer storage devices and also investigates potential new optimisations that can help improve performance. To demonstrate the results of my investigation I have ported over the simple **Minix** file system from MinixOS kernel code to the File System in Userspace (FUSE) framework and then retrofitted it with implementations of the ideas developed. I have called this new file system **Somix** meaning SSD Optimised Minix.

Background

SSDs, unlike mechanical hard drives (**HDDs**), consist of no moving parts. When data is requested or written to a mechanical hard drive a read/write head has to physically scan over the surface of a magnetic disk to locate the associated data region. Such a mechanical requirement introduces a reasonably significant latency in data access, typically 5-10ms on modern drives. With SSDs however all operations are entirely electrical and as a result access times are significantly reduced, usually to within 0.1ms. This improvement results in SSDs having much faster read times compared to their mechanical counterparts. Unfortunately the major pitfall for SSDs lies in their write performance. SSDs differ from HDDs in that they cannot simply 'overwrite' existing data. Instead data must be explicitly erased before the containing cell can be written to. The problem for SSDs is that erase operations can only be performed on large blocks of data, typically between 128 and 512 KB. As a result, the preceding erase operations make block rewrites significantly slower, often putting performance on a par with mechanical drives.
Among the optimizations being developed in this project, some build on the previous work of a similar project, IotaFS[1].

Comparison

The graphs below provide a comparison between performance metrics for typical SSD and HDD hardware available today.

Fig 1: Sequential R/W comparison chart.

Fig 2: Typical SSD component.

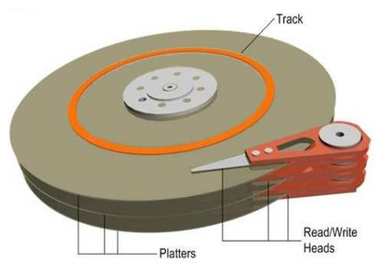

Fig 3: Typical HDD component.

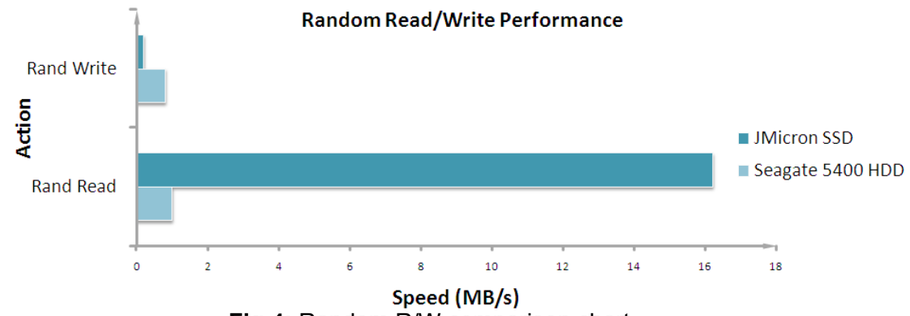

Fig 4: Random R/W comparison chart.

The most significant result above is the SSD random read performance. This particular drive has over 16X the random read throughput of the mechanical drive. It is primarily this property of SSD hardware that can be utilized in SSD specific file system optimizations.

Optimizations

The optimisations being investigated as part of this project make fundamental design changes to the Minix file system. These changes involve the data written to the file system being fragmented where convenient and necessary. Such changes would be catastrophic to file systems used on mechanical hard drives but are much less detrimental to SSDs. By allowing ourselves to fragment data in such a way we can develop optimisations that result in both improved write performance and reduced metadata storage requirements.

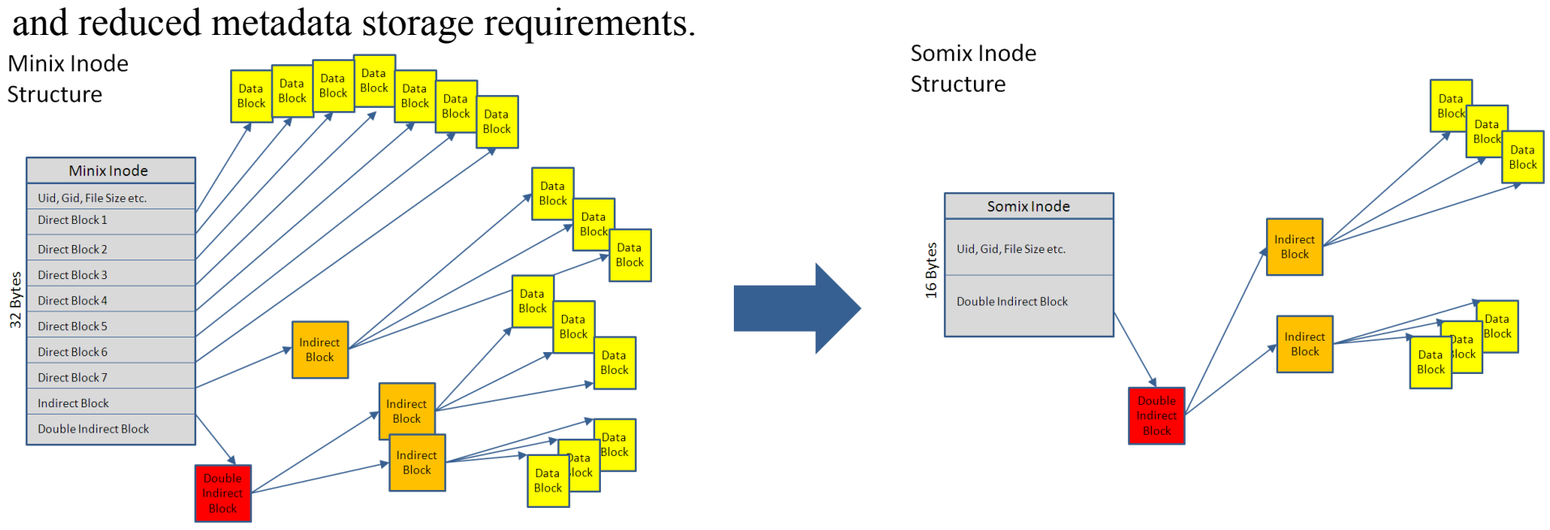

Fig 5: Example of one optimisation that removes all direct and indirect block pointers from the file systems Inode structure. This reduces Inode size by half and translates to a storage saving of just over 1% the total volume size. The main cost of this is a doubling in the number of random reads per data access. However, for SSDs these extra accesses are not expected to result in any significant performance hit.

Implementation

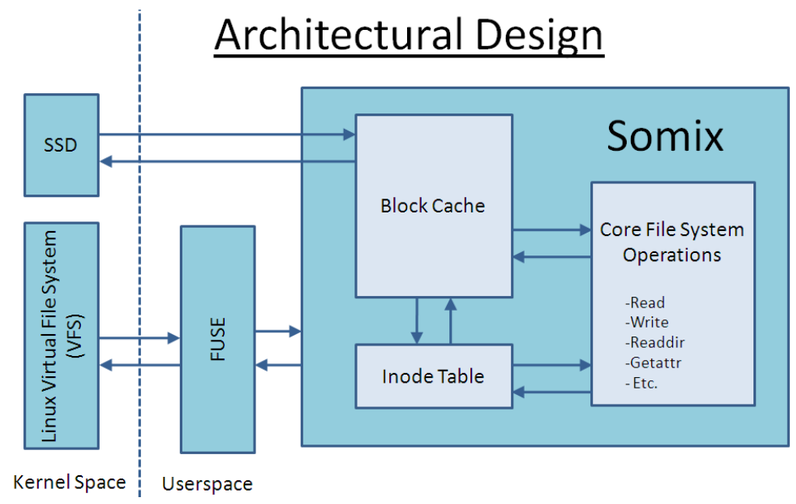

Fig 6: Somix architectural design.

Somix is being developed in C and is designed for use on Linux. It interfaces with the FUSE framework to communicate with the kernel and handle file I/O. Once a request is received from the kernel, FUSE passes it to the Somix core code which then issues read/write requests to the SSD device through the internal block cache. Where required, Somix responds to the system query with file data.

Conclusions

The final implementation of Somix is expected to demonstrate improved random write performance and a more efficient use of storage space when operating on SSDs. However, it is possible that these improvements may come at the expense of reduced read performance and additional processing overhead. Further still, the onboard microcontrollers on SSD drives are becoming increasingly sophisticated to the extent where they may start to negate some of the positive effects these optimisations provide.

References

[1] Understanding SSDs and New Drives from OCZ. URL http://www.anandtech.com/storage/showdoc.aspx?i=3531&p=1

[2] IotaFS: Exploring File System Optimizations for SSDs. URL http://www.eecs.berkeley.edu/~laurak/iotafs.pdf

[Comparison Data] URL http://www.anandtech.com/storage/showdoc.aspx?i=3531&p=24