Lorenzo Gomez
Jin Huang

## Abstract

File systems such as ext4 are great for servers, power users and everyday users doing general purpose tasks. However, as somebody that has used computers for writing on a daily basis;to write poetry, fiction, create worlds and characters, computers feel *almost* like the perfect tool for the job. And to some extent they are. They can take our writing and guarantee(as long as we hit the “save” button) that our characters and worlds will never disappear. This is great. However, when writers such as myself grab their laptop, go to their coffee shop happy place(this is usually anywhere quiet for me), start building worlds and then no more than four hours later, I find out that my laptop gives me some kind of notification saying I only have 15 minutes left of battery life. This is disruptive to the process of writing, to the creative process. Looking into the way ext4 works, we believe that that it could be better designed for the job of writing. ext4 has many drawbacks that impact writers in significant and negative ways.

The main drawback we’ll focus on this paper is power. ext4, aside from being overly complex for a simple task such as writing, has also mechanisms that push our hardware to consume more power than it needs to. A big example of this is journaling. When running a commodity linux system, the default setting is to update filesystem status such as the state of the journal every 5 seconds. While this makes our data reliable and crash-consistent, it is not a solution that takes into consideration power consumption. This means that if someone opens up their novel and starts writing, in just 1 minute it has made sixty commits. In just one hour(which is not much when one is writing given that that time might be spent just thinking, which is crucial to the creative process), ext4 has made 3600 commits. Such high amount of commits just incurs more writes in our linux systems—our hardware is being forced to use more power because of all these writes. It should be mentioned that mechanisms such as fsync make this power problem even worse as it forces us to go to disk, which is another power nightmare(including SSDs and Hard Disks). For the purposes of this paper, we will focus on tackling issues file system structure-specific issues—such as block sizes. After discussing our idea in detail with the professor, we realized that ext4 was not a good vantage point for us  as starting point. As mentioned before it is overly complex, and most importantly, does not support, multiple block sizes while it is running—it has very little context. Multiple block sizes(along with some context) can allow us to design a filesystem that will not incur as many unnecessary writes as ext4 as it will be able to make better decisions about *when* to commit, give some context from user space.

## Introduction

In this workshop paper, we propose the Script File System(scriptfs). scriptfs is a file system that knows a little more about the user than your typical ext4-like filesystem. In scriptfs, our users will be able to  tell the file system what they are writing. scriptfs will provide a syscall interface that will allow application developers to pass explicit parameters to scriptfs, so that scriptfs can make a smart decision about block sizes schemes. For example, after running some stories such as Sherlock Holmes and Alice in Wonderland through a python script, we found that most words are no more than 5 characters(5 bytes) on average. This is rather convenient for us as even a 4,000 word poem (which is a very *high* bound for contemporary poetry), would not be larger than 20KB! This means that, we can safe to allocate 32KB block sizes in memory since the very first moment our users *open* a file.  Poems aren’t the only keywords that tell us context, later on we’ll explore the details of allocating blocks of other sizes(for Short Stories, Novels, etc) in later sections.

A paper that has influenced our idea significantly is the idea of an *Infinite Cache* that [Li] demonstrate so well on his Towards A Low Power File System paper. The basic idea is that our filesystem goes to disk only when it *absolutely* has to. Take the Poem, which we consider to be only 32KB. For common household caches, this is nothing. We can do whole-file prefetching just like [Li] suggests, given that our biggest file in the context of scriptfs is no more than 2MB.

## Background

scriptfs was thought of as a part of a whole other operating system—TypeOS. Like I described in my proposal, TypeOS is an OS for writers that is power-centric. A core unit of this OS would be a file system like scriptfs that understands context. In TypeOS, scriptfs will provide an infrastructure for a file system that is crash-consistent, reliable and fast. However, it will prioritize power consumption over performance. This will imply creating a new blocksize scheme that will support multiple blocksizes at runtime, a journaling system that uses power as a unit of time. For instance, if you can imagine, only backing file and data to disk when and only if our OS has lost 25% percent of power. This will allow scriptfs to stay in memory *most* of the time, avoiding power-hungry mechanisms like ext4's aggressive commit journaling or too many fsync calls. The goal with a file system like sriptfs is to be able to grab one's laptop. go out in the woods and write a 40-page short story without having to worry about charging one's laptop, which will disrupt the delicate creative process of writing.

## Motivation

As a developer who writes, writing software can be disheartning to use as such like MS Word are bloated with annoying features, especially auto-save features that go to disk and shorten the lifespan of a battery cycle. When one starts writing, computers should be able to last a lot longer given that the act of writing itself is and *should not* be computationally expensive(which leads to power expensive).

## Analysis

Since different categories of novels have their own size range, we could just allocate enough pages for the specific novels at once, thus reducing the kernel crossing; and we can also change the pagesize to reduce of the overheads of page allocating. Because of all the context we have, we can also exploit the idea of Infinite Cache as most of our files are expected to not be bigger than 2MB(with the exception of the Super Novel). As mentioned before, ext4 is not a good starting point to implement scriptfs. We will be modifying ramfs so that it can support multiple blocksizes. Our current plan to do this is create our own functions for ramfs such as simple_write_begin so that we can contextualize these functions and, say, allocate Poems. The way we are going to contextualize these functions is by creating a syscall interface that will interact with user space so that users(application developers) can help scriptfs(a ramfs implementation) can smarter decisions and reduce MMU interaction, disk transfers and kernel crossings and ultimately save power.

For our scripts and current work visit Github:

| Category | Size | Pagenum |
|---|---|---|
| Poem | 32KB | 8 |

| | | |
|---|---|---|
| Short Story | 128KB | 32 |
| Small Novel | 256KB | 64 |
| Medium Novel | 512KB | 128 |
| Large Novel | 2MB | 512 |
| Super Novel | 256MB | 64K |