COEN 283 Term Project

Multilevel Queue Algorithm in Hybrid File System

By

Zhe Xu(Yishion)
W1095418
Xiaoguang Mo(Michael)
W1114182
Qianqian Zhong(Brady)
W1105442

In

Computer Science and Engineering

At

Santa Clara University

Date: 2014-11-19
DECLARATION

I hereby declare that all the work done in this Project is of my independent effort. I also certify that I have never submitted the idea and product of this Project for academic or employment credits.

___________________
Zhe Xu, Xiaoguang Mo, Qianqian Zhong

Date: ______________

Date: 2014-11-19
I would like to express my great gratitude towards my supervisor, Dr. Minghwa Wang who had given me invaluable advice to this project.
1. Abstract
2. introduction
   2.1 Objective
   2.2 What is the Problem
   2.3 Why this is a project related the this class
   2.4 why other approach is no good
   2.5 Why you think your approach is better
   2.6 Statement of the problem
   2.7 Area or scope of investigation
3. theoretical bases and literature review
   3.1 Theoretical background of the problem
   3.2 Related research to solve the problem
      3.2.1 Combo Drive: Optimizing Cost and Performance in a Heterogeneous Storage Device
      3.2.2 Fusion Drive
      3.2.3 SSD as a Cache Memory
      3.2.4 I-CASH: Intelligently Coupled Array of SSD and HDD
   3.3 advantage/disadvantage of those research
      3.3.1 Combo Drive: Optimizing Cost and Performance in a Heterogeneous Storage Device
      3.3.2 Fusion Drive
      3.3.3 SSD as a Cache Memory
      3.3.4 I-CASH: Intelligently Coupled Array of SSD and HDD
   3.4 Your solution to solve this problem
   3.5 Where your solution different from others
4. hypothesis (or goals)
   4.1 Positive/negative hypothesis
5. Methodology
   5.1 How to generate/collect input data
   5.2 How to solve the problem
      5.2.1 Algorithm design
      5.2.2 Language used
      5.2.3 Tools used
   5.3 How to generate output
   5.4 How to test against hypothesis
6. implementation
   6.1 Code (refer programming requirements)
       Makefile:
       Main.cpp
       Sample Input:
   6.2 Design Input and flowchart
7 data analysis and discussion
   7.1 output generation
   7.2 output analysis
   7.3 Compare output against hypothesis
   7.4 Abnormal case explanation (the most important task)
   7.5 statistic regression
   7.6 discussion
8. conclusions and recommendations
   8.1 summary and conclusions
   8.2 recommendations for future studies
9. bibliography

Date: 2014-11-19
1. Abstract

This paper describes the concepts and implementation of a hybrid storage system for an application of IOS platform. And this project takes an imaginary toy factory which produces Transformers as an example. Its main function is monitoring the flow of supply chain, keeping records and showing the data in order. This paper will also show the process of developing such an IOS application.

2. introduction

2.1 Objective
In modern computer science area, data is becoming increasingly important. Big data, cloud storage, security, even retailers start to acknowledge the importance of data. With the expansion of data, storage is also becoming a tough question to solve. Fortunately, with the development of SSD, people now have one more choice for data storage. In spite of the decreasing price and increasing speed of SSD NAND, most computer users still can not afford a SSD for the whole storage solution. According to the investigation of hard disk market, by Nov 28 2014, the HDD is generally 8 times slower and cheaper than SSD. So, the hybrid storage system could be the solution for this problem.

2.2 What is the Problem
To create a hybrid storage hierarchy, the essential algorithm and design is needed. The difference between SSD and HDD should be highlighted, the whole system should be designed to make the most of the advantage of both SSD and HDD.

2.3 Why this is a project related the this class
This project aims to optimize the file system, which is one of the core parts of the operating system. By developing such a file system we could increase the overall running efficiency of operating system.

2.4 why other approach is no good

Date: 2014-11-19
Some hybrid storage hierarchies already come out and take places in hard disk market. Like SSHD from Seagate, Fusion Drive from Apple, and some other hybrid system that already published but still waiting for implementation to market. Some hybrid storage system could significantly improve the overall performance of hard disk with a low cost, but they are still not good enough. SSHD and Fusion Drive focus on hard drive, they try to log the usage of sectors and move the more frequently used sectors to SSD, then move related files to SSD. They don’t care about the users’ habit, just mechanically do some statistics job then move the files. Other solutions like Combo Drive and Hybrid File System based on files but their algorithms are too simple, still not smart enough to make the most of the combination of SSD and HDD.

2.5 Why you think your approach is better

The popular solutions for hybrid storage system mainly focus on moving frequently used data to SSD. This algorithm is simple to implement but not intelligent enough. For example, if a student wrote homework, edit that document very frequently, then the document is moved to SSD. Then after the homework is done and uploaded, it will not be edited for years. However, the usage frequency is still very high, if the homework is a very big video file, it will take up a lot of space on SSD. Our approach is much more brilliant and thoughtful, it will create a priority system for all files, the whole file system is managed based on the file type, size and frequency. The new file will be allocated a priority based on its type and size, then edit the priority according to its access frequency. This algorithm could avoid some extreme situation that waste SSD’s capacity to store big files or make SSD keep a lot of free space.

2.6 Statement of the problem

The problem is we need to design an efficient algorithm that can beat the existing algorithm to make the most of SSD’s high speed and HDD’s high capacity.

2.7 Area or scope of investigation

The area for this project is hybrid file system and corresponding algorithm of operating system.

3. theoretical bases and literature review

3.1 Theoretical background of the problem

Many theories and solutions for this problem are already exist, but still need to be
3.2 Related research to solve the problem

3.2.1 Combo Drive: Optimizing Cost and Performance in a Heterogeneous Storage Device

Combo Drive aimed to regard the hybrid storage system as a single hard disk. Simply put the sectors of SSD in front of HDD, and put frequently used files in SSD. They designed two algorithm, one called static optimization and another called dynamic optimization. The static optimization organize the files according to their file type, it will put all executable files like .exe files and library files like .dll files SSD. The dynamic optimization will log the access frequencies of sectors, then find the corresponding file that take the place of the sector and move the file to SSD.

3.2.2 Fusion Drive

Fusion Drive is firstly published by Apple in 2012 and widely used on iMac. It is based on hardware and simply operates the hard disk to achieve their goal to speed up the hard disk

3.2.3 SSD as a Cache Memory

SSD as a Cache Memory is a solution that add another layer between hard disk and memory. File system will put file that will be accessed into SSD then transfer to memory. Make the SSD like another memory that won't lose data after power off.

3.2.4 I-CASH: Intelligently Coupled Array of SSD and HDD

I-CASH makes the combination of SSD and HDD like an array. Both SSD and HDD will maintain a log of all files’ read and edit history, then put the files with more read operation but less edit operation into SSD and files with more edit operation but less read operation into HDD.

3.3 advantage/disadvantage of those research

3.3.1 Combo Drive: Optimizing Cost and Performance in a Heterogeneous Storage Device

For static optimization of Combo Drive, it is inefficient when user try to edit some big and frequently used files. For example, when user wants to edit a video file, user want the editing process faster but since it is not an executable file nor library file, the file system will never put it into SSD. The only advantage for static optimization is it is easy to implement. For dynamic optimization, this algorithm seems to be perfect but it still need some improvement about take the file size into consideration and deal with

Date: 2014-11-19
the files which frequently edited within a small period of time but won’t be edit in the future.

3.3.2 Fusion Drive
The Fusion Drive is a success because it is perfectly implemented in industry area with the help of Apple’s support. So the biggest disadvantage is also it is supported by Apple, then the patent becomes the biggest problem for other companies. Further more, this algorithm only organize the hard disk and ignore the feature of files. If a file system ignore the size and type of files and simple consider the access frequency of sectors, it couldn’t be a eligible file system but more like a function of chip on hard disk.

3.3.3 SSD as a Cache Memory
The advantage for SSD as a Cache Memory is it is easy to design and probably will have a much more irradiant future usage. It is just like designed for a future product, a new kind of memory that won’t lose data after power off. But the main drawback for this algorithm is also because it doesn’t look like a solution for file system. It completely ignored the feature of SSD. Make it more like a hybrid memory system.

3.3.4 I-CASH: Intelligently Coupled Array of SSD and HDD
I-CASH system is very brilliant for it considered the read operation and edit operation separately but it ignored the feature of files’ size and type. It also doesn’t consider how to make the most usage of SSD’s capacity.

3.4 Your solution to solve this problem
Our solution is a synthesis of other solutions plus our own idea. Firstly, according to the update the chip of SSD. The modern SSD’s sector could endure over 1.5 PB’s edit operation. So the first feature of our solution is to make the most usage of the capacity of SSD. The SSD will be separated into two parts, one is called normal storage space, another is called buffer, their proportion is decided by the capacity of SSD and HDD according to our algorithm. The file system aim to keep the storage space of SSD always full to make more files could be accessed in high speed.

Secondly, we allocate a priority for every files in the file system. The priority will base on the file’s size and type when it is created at the first time. The smaller file will have a higher priority and executable and library files also have a higher priority.

Then, after files are created, the file system will log their access frequency. The files

Date: 2014-11-19
with higher access frequency will have a higher priority in the priority system, but it will also have a limitation of priority based on its size and type. For example, no matter how often you edit a document file, it won’t have a higher priority than the library file of operating system.

The most important part is how to decide which file will be move to SSD or HDD and how to move it. Based on our algorithm, all files have their own priorities, and the SSD’s storage space is almost full unless the number of files is too small. Then when a new file is created, it will firstly created in SSD to speed up the process of creating a file. If the storage space of SSD is full, it will be created in buffer space of SSD. When the buffer is almost full, the file system will check the priority of files in SSD, then move the files with lower priority to HDD from both storage space and buffer space. And the file system will also move HDD’s files with the priority that higher than the lowest priority in SSD. Then if the storage space of SSD still has free space, it will move files in buffer space into storage space.

3.5 Where your solution different from others
Our solution absorbed other solutions’ advantage and overcame their drawback. It is based on file system, considered the size, type and access frequency of file, and makes the most use of the capacity of SSD.

4. hypothesis (or goals)

4.1 Positive/negative hypothesis
The algorithm of FIFO, LRU will also be implement in the testing. The hypothesis is our algorithm could be more efficient than FIFO and LRU.

5. Methodology

5.1 How to generate/collection input data
The input generator is designed to generate virtual data based on the statistical data. Since the number of files in a computer is extremely giant, the input generator is designed to generate a set of files at one time. For example, it will be set to generate 500 files, type is .doc, size is between 10MB and 1GB. It could generate millions of files in one second then send it to file system to write then into the hard disk orderly.

5.2 How to solve the problem
To solve this problem, we need to design a simulator to simulate the real hardware because of the limitation of time. Give the data generated from input generator and run in our algorithm, LRU and FIFO to test the efficiency of each algorithm

Date: 2014-11-19
5.2.1 Algorithm design
The algorithm is like 3.4. Since it is run on simulator, the algorithm is redesigned to simulate the real situation faster. It will ignore the time consuming of read/write operation from hardware, using anticipated time consuming data calculated by the read/write speed from typical hardware data and file size.

5.2.2 Language used
C++

5.2.3 Tools used
Xcode, Github

5.3 How to generate output
The simulator will get the input data from input generator and run all testing algorithms, the output data could be seen in the simulated hardware file and the statistical data of running efficiency will be shown in console.

5.4 How to test against hypothesis
By comparing running efficiency of different algorithms, the statistical results could be used to illustrate the distinct comparison of different algorithm. The input and output should be same for all algorithms to make sure different situations are compared in same standard.

6. implementation

6.1 Code (refer programming requirements)
All code could be seen on https://github.com/bzhong/SSD-based-storage-system. The structure of this project is here:

Date: 2014-11-19
Makefile:

```makefile
TARGET = main
OBJ = common/epoch/structure.o disk/disk.o algorithm/replacement_algo.o input_generator/input_generator.o driver/main.o
CC = gcc
DEBUG = -g
CFLAGS = -Wall -std=c++11 -x C -l$(DEBUG)
LDFLAGS = -Wall $(DEBUG)
INC = -I./common -I./algorithm -I./disk -I./input_generator
all: input_generator/tinyxml2.h
cd common; make
cd disk; make
cd algorithm; make
cd input_generator; make
cd driver; make
$<($CC) $(CFLAGS) $(LIB) -o $(TARGET) $(OBJ) input_generator/tinyxml2.cpp

clean:
cd common; make clean
cd disk; make clean
cd algorithm; make clean
cd input_generator; make clean
cd driver; make clean
rm $(TARGET)
```

Main.cpp

```cpp
// main.cpp
// SSD-based storage system
//
// Created by Brady on 11/7/14.
// Copyright (c) 2014 Qianqian Zhong. All rights reserved.
//
#include "input_generator.h"
#include "disk.h"
#include "replacement_algo.h"
using namespace std;

int main() {
   Driver dr;
   string filename;
   cout << "Please indicate the absolute path of configuration file you want to choose (e.g. /user/config.xml):" << endl;
   cin >> filename;
   dr.RegisterFile(filename);
   dr.Run();
}
```

Date: 2014-11-19
Sample Input:

```xml
<InputGenerator>
  <SSD>64G</SSD>
  <test_length>3Y</test_length>

  <FileSet id="0">
    <filename>200000</filename>
    <minsize>1K</minsize>
    <maxsize>20G</maxsize>
    <distribute>Average</distribute>
    <writerate>20</writerate>
    <singlerequest>10</singlerequest>
    <frequency>1M</frequency>
    <start>0</start>
    <length>3Y</length>
    <type id="0">PY</type>
  </FileSet>

  <FileSet id="2">
    <filename>200000</filename>
    <minsize>1K</minsize>
    <maxsize>20G</maxsize>
    <distribute>Normal</distribute>
    <writerate>20</writerate>
    <singlerequest>10</singlerequest>
    <frequency>1M</frequency>
    <start>0</start>
    <length>3Y</length>
    <type id="0">PY</type>
    <type id="1">DOC</type>
  </FileSet>
</InputGenerator>
```

6.2 Design document and flowchart

This project gives an innovative replacement algorithm for SSD-based hybrid storage system. The project contains four parts: input generator, core algorithm, disk simulator and event driver. Input generator produces customized various kinds of workloads. Core algorithm contains traditional replacement algorithms like FIFO and LRU in addition to our innovative algorithm. Disk simulator mimics behaviors of HDDs and SSDs. Last, event driver triggers other three parts to coordinate with each other and make the system work well.

Date: 2014-11-19
7 data analysis and discussion

Sample Output Data:

```
print results: 1 for MQA, 2 for FIFO, 3 for LRU
Algo 1
algo exec time: 6.13e+94ms
ssd exec time: 5.95e+97ms
hdd exec time: 274468095.34878564545ms
hit count: 0.021349
Algo 2
algo exec time: 488000ms
ssd exec time: 5.951e+97ms
hdd exec time: 278743291.39536396964ms
hit count: 0.03185
Algo 3
algo exec time: 50495ms
ssd exec time: 5.951e+97ms
hdd exec time: 279743270.23072440751ms
hit count: 0.03185
```

7.1 output generation

Output are generated as txt files. It contains the logging message of the run-time and the overall performance of three algorithm including the total execution time of algorithm, time consuming of SSD’s operation, time consuming of HDD’s operation and hit rate. The output data is collected and transferred to a numbers file to generate a chart so that the differences of result could be shown more obviously.

7.2 output analysis

Date: 2014-11-19
From the output we could see that our algorithm is much more efficient than FIFO and LRU. Generally, the test results from bigger SSD are much better than smaller SSD because it can store more data in high speed SSD and decrease the operation of moving data from SSD to HDD. Then, for almost all the time our algorithm is better than FIFO and LRU except some situation that at the beginning of the test LRU could be a little bit faster than our algorithm according to the hit rate.

HDD request table indicated that the operation done on HDD, since HDD is much slower than SSD, HDD request is better to be lower. In the chart the HDD request of

**Date:** 2014-11-19
MQA is always lower than FIFO and LRU with the same capacity. That meets the goal of MQA to make the most usage of SSD’s high speed.

Algorithm Delay chart shows the delay time of running different algorithms, delay is the lower the better. MQA is still the best among three algorithms, the delay of MQA is the lowest for the most of time, sometimes FIFO is better than MQA because the complexity of FIFO is lower than other two algorithm. Since FIFO has the worst hit rate and worse than MQA for the most of time on delay, it can also prove that using MQA is worth the tiny fault on delay.

7.3 Compare output against hypothesis
The test result could show that the hypothesis is achieved, MQA is the best algorithm among three algorithms

7.4 Abnormal case explanation (the most important task)
The abnormal case in hit rate is at the beginning of the test LRU is sometimes better than MQA, because MQA will keep some blank space for buffer, its real capacity will be smaller than SSD with LRU.
The abnormal case in HDD request is when SSD is big enough (1024 GB), its request is extremely low. That’s because only when OS try to write some file that is too big will MQA work and put the big file into HDD. Since the number of big files is not large, after write several big files into HDD, the OS will no longer write any more files to it. At the same time, LRU and FIFO still have to move files frequently.
The abnormal case in algorithm delay is a little bit complicated, since all algorithm has its own best situation. The only thing that could be figured out easily is in average MQA is the best among three algorithm, then FIFO sometimes became the best one because of its low complexity of algorithm.

7.5 statistic regression
The test result is highly depend on the capacity of SSD, all algorithms work better on SSD with larger capacity. When capacity of SSD doubled, the performance of each algorithm is increasing more than twice of original performance. MQA increased the most significant, it meets its design goal, to make the most usage of SSD’s speed.

7.6 discussion
The performance of MQA is obviously stronger than FIFO and LRU. However FIFO and LRU are very old algorithms and didn’t aim to optimize the performance of hybrid storage system. The comparison of MQA with other algorithm that aim to optimize heterogeneous storage system will be more interesting and challenging.

8. conclusions and recommendations

8.1 summary and conclusions

Date: 2014-11-19
In conclusion, this project is a big success since it beats FIFO and LRU in overall performance and meets our design goal. The algorithm of file system for hybrid storage system is very important if user want to improve the overall performance of their computer.

8.2 recommendations for future studies

For future studies, MQA could be compared with other file system that designed for hybrid storage system. And if time is enough, MQA could leave the simulator and run on real file system to test its real performance on computer. Then if the performance is still as good as it on simulator, it could be implemented to market, take the place of fusion drive and other existing file systems.

9. bibliography


[5] Raja Appuswamy, David C. van Moolenbroek AND Andrew S. Tanenbaum. Integrating Flash-based SSDs into the Storage Stack


Date: 2014-11-19