CS270: Advanced Operating Systems

Course Project on File System Implementation

Gareth George Thomas Schibler Nazmus Saquib

Graduate Students
Department of Computer Science
University of California Santa Barbara

December 2, 2018

UCSB

Outline

1. Introduction
2. Architecture
3. Beyond Basics
4. Challenges
5. Performance Benchmark
6. Conclusion
7. Questions

Introduction: Design Goals

- Iterative approach to development
- Encapsulation
- Testable
  - object-oriented design
  - unit test for each object
  - integration test

Introduction: Design Goals

- Iterative approach to development
- Encapsulation
- Testable
  - object-oriented design
  - unit test for each object
  - integration test

Introduction: Design Goals

- Iterative approach to development
- Encapsulation
- Testable
    - object-oriented design
    - unit test for each object
    - integration test

Introduction: Design Goals

- Iterative approach to development
- Encapsulation
- Testable
  - object-oriented design
  - unit test for each object
  - integration test

Introduction: Design Goals

- Iterative approach to development
- Encapsulation
- Testable
  - object-oriented design
  - unit test for each object
  - integration test

Introduction: Design Goals

- Iterative approach to development
- Encapsulation
- Testable
  - object-oriented design
  - unit test for each object
  - integration test

Introduction: Design Goals *(Cntd.)*

- General data structures
  - superblock
  - inode table
  - segment manager (LFS)
- Memory mapped files
  - performance gain at the cost of reliability
  - acceptable for high performance system
  - can be seen in Mach

Introduction: Design Goals *(Cntd.)*

- General data structures
  - superblock
  - inode table
  - segment manager (LFS)
- Memory mapped files
  - performance gain at the cost of reliability
  - acceptable for high performance system
  - can be seen in Mach

Introduction: Design Goals *(Cntd.)*

- General data structures
  - superblock
  - inode table
  - segment manager (LFS)
- Memory mapped files
  - performance gain at the cost of reliability
  - acceptable for high performance system
  - can be seen in Mach

Introduction: Design Goals *(Cntd.)*

- General data structures
  - superblock
  - inode table
  - segment manager (LFS)
- Memory mapped files
  - performance gain at the cost of reliability
  - acceptable for high performance system
  - can be seen in Mach

Introduction: Design Goals *(Cntd.)*

- General data structures
  - superblock
  - inode table
  - segment manager (LFS)
- Memory mapped files
  - performance gain at the cost of reliability
  - acceptable for high performance system
  - can be seen in Mach

Introduction: Design Goals *(Cntd.)*

- General data structures
  - superblock
  - inode table
  - segment manager (LFS)
- Memory mapped files
  - performance gain at the cost of reliability
  - acceptable for high performance system
  - can be seen in Mach

Introduction: Design Goals *(Cntd.)*

- General data structures
    - superblock
    - inode table
    - segment manager (LFS)
- Memory mapped files
    - performance gain at the cost of reliability
    - acceptable for high performance system
    - can be seen in Mach

Introduction: Design Goals *(Cntd.)*

- General data structures
  - superblock
  - inode table
  - segment manager (LFS)
- Memory mapped files
  - performance gain at the cost of reliability
  - acceptable for high performance system
  - can be seen in Mach

Introduction: Memory Mapped Files

+ Efficient paging as kernel handles it

– Lesser direct control

– Restricts fine-grained control over writes

Introduction: Memory Mapped Files

+ Efficient paging as kernel handles it

– Lesser direct control

– Restricts fine-grained control over writes

Introduction: Memory Mapped Files

+ Efficient paging as kernel handles it

– Lesser direct control

– Restricts fine-grained control over writes

Outline

1. Introduction
2. Architecture
3. Beyond Basics
4. Challenges
5. Performance Benchmark
6. Conclusion
7. Questions

Architecture: Hybrid Log Structured File System (LFS)

- Superblock
  - holds references to all other file system data structures
  - initializes
    - the inode table
    - the segment controller
- Inodes
  - closely resembles inodes from "A Fast File System for UNIX"
  - each inode contains 8 direct, 1 indirect, 1 double indirect, and 1 triple indirect blocks
  - additionally complicated by the need for an algorithm that allows moving both data blocks and indirect pages belonging to an inode

Architecture: Hybrid Log Structured File System (LFS)

- Superblock
  - holds references to all other file system data structures
  - initializes
    - the inode table
    - the segment controller
- Inodes
  - closely resembles inodes from "A Fast File System for UNIX"
  - each inode contains 8 direct, 1 indirect, 1 double indirect, and 1 triple indirect blocks
  - additionally complicated by the need for an algorithm that allows moving both data blocks and indirect pages belonging to an inode

Architecture: Hybrid Log Structured File System (LFS)

- Superblock
  - holds references to all other file system data structures
  - initializes
    - the inode table
    - the segment controller
- Inodes
  - closely resembles inodes from "A Fast File System for UNIX"
  - each inode contains 8 direct, 1 indirect, 1 double indirect, and 1 triple indirect blocks
  - additionally complicated by the need for an algorithm that allows moving both data blocks and indirect pages belonging to an inode

Architecture: Hybrid Log Structured File System (LFS)

- Superblock
  - holds references to all other file system data structures
  - initializes
    - the inode table
    - the segment controller
- Inodes
  - closely resembles inodes from "A Fast File System for UNIX"
  - each inode contains 8 direct, 1 indirect, 1 double indirect, and 1 triple indirect blocks
  - additionally complicated by the need for an algorithm that allows moving both data blocks and indirect pages belonging to an inode

Architecture: Hybrid Log Structured File System (LFS)

- Superblock
  - holds references to all other file system data structures
  - initializes
    - the inode table
    - the segment controller
- Inodes
  - closely resembles inodes from "A Fast File System for UNIX"
  - each inode contains 8 direct, 1 indirect, 1 double indirect, and 1 triple indirect blocks
  - additionally complicated by the need for an algorithm that allows moving both data blocks and indirect pages belonging to an inode

Architecture: Hybrid Log Structured File System (LFS)

- Superblock
  - holds references to all other file system data structures
  - initializes
    - the inode table
    - the segment controller
- Inodes
  - closely resembles inodes from "A Fast File System for UNIX"
  - each inode contains 8 direct, 1 indirect, 1 double indirect, and 1 triple indirect blocks
  - additionally complicated by the need for an algorithm that allows moving both data blocks and indirect pages belonging to an inode

Architecture: Hybrid Log Structured File System (LFS)

- Superblock
  - holds references to all other file system data structures
  - initializes
    - the inode table
    - the segment controller
- Inodes
  - closely resembles inodes from “A Fast File System for UNIX”
  - each inode contains 8 direct, 1 indirect, 1 double indirect, and 1 triple indirect blocks
  - additionally complicated by the need for an algorithm that allows moving both data blocks and indirect pages belonging to an inode

Architecture: Hybrid Log Structured File System (LFS)

- Superblock
  - holds references to all other file system data structures
  - initializes
    - the inode table
    - the segment controller
- Inodes
  - closely resembles inodes from "A Fast File System for UNIX"
  - each inode contains 8 direct, 1 indirect, 1 double indirect, and 1 triple indirect blocks
  - additionally complicated by the need for an algorithm that allows moving both data blocks and indirect pages belonging to an inode

Architecture: Hybrid Log Structured File System (LFS)

- Superblock
  - holds references to all other file system data structures
  - initializes
    - the inode table
    - the segment controller
- Inodes
  - closely resembles inodes from “A Fast File System for UNIX”
  - each inode contains 8 direct, 1 indirect, 1 double indirect, and 1 triple indirect blocks
  - additionally complicated by the need for an algorithm that allows moving both data blocks and indirect pages belonging to an inode

Outline

1. Introduction
2. Architecture
3. Beyond Basics
4. Challenges
5. Performance Benchmark
6. Conclusion
7. Questions

Beyond Basics

- Segment controller
  - heavy lifting data structure
  - manages a pool of segments
  - tracks segment utilization for efficient GC
- Segment
  - each segment maps its data blocks to the inodes that own them
  - keeps a write head that allows for constant time chunk allocation
  - finding and garbage collecting low utilization chunks can be costly, however
- Experiences with LFS
  - + LFS dramatically decreased time to allocate chunks
  - + Faster sequential writes due to constant time chunk allocation
  - – Slower updates due to poor utilization of page cache when doing copy on write for updates
  - – Introduces costly GC operation

Beyond Basics

- Segment controller
  - heavy lifting data structure
  - manages a pool of segments
  - tracks segment utilization for efficient GC
- Segment
  - each segment maps its data blocks to the inodes that own them
  - keeps a write head that allows for constant time chunk allocation
  - finding and garbage collecting low utilization chunks can be costly, however
- Experiences with LFS
  - + LFS dramatically decreased time to allocate chunks
  - + Faster sequential writes due to constant time chunk allocation
  - – Slower updates due to poor utilization of page cache when doing copy on write for updates
  - – Introduces costly GC operation

Beyond Basics

- Segment controller
  - heavy lifting data structure
  - manages a pool of segments
  - tracks segment utilization for efficient GC
- Segment
  - each segment maps its data blocks to the inodes that own them
  - keeps a write head that allows for constant time chunk allocation
  - finding and garbage collecting low utilization chunks can be costly, however
- Experiences with LFS
  - + LFS dramatically decreased time to allocate chunks
  - + Faster sequential writes due to constant time chunk allocation
  - − Slower updates due to poor utilization of page cache when doing copy on write for updates
  - − Introduces costly GC operation

Beyond Basics

- Segment controller
  - heavy lifting data structure
  - manages a pool of segments
  - tracks segment utilization for efficient GC

- Segment
  - each segment maps its data blocks to the inodes that own them
  - keeps a write head that allows for constant time chunk allocation
  - finding and garbage collecting low utilization chunks can be costly, however

- Experiences with LFS
  - + LFS dramatically decreased time to allocate chunks
  - + Faster sequential writes due to constant time chunk allocation
  - − Slower updates due to poor utilization of page cache when doing copy on write for updates
  - − Introduces costly GC operation

Beyond Basics

- Segment controller
  - heavy lifting data structure
  - manages a pool of segments
  - tracks segment utilization for efficient GC
- Segment
  - each segment maps its data blocks to the inodes that own them
  - keeps a write head that allows for constant time chunk allocation
  - finding and garbage collecting low utilization chunks can be costly, however
- Experiences with LFS
  - + LFS dramatically decreased time to allocate chunks
  - + Faster sequential writes due to constant time chunk allocation
  - – Slower updates due to poor utilization of page cache when doing copy on write for updates
  - – Introduces costly GC operation

Beyond Basics

- Segment controller
  - heavy lifting data structure
  - manages a pool of segments
  - tracks segment utilization for efficient GC
- Segment
  - each segment maps its data blocks to the inodes that own them
  - keeps a write head that allows for constant time chunk allocation
  - finding and garbage collecting low utilization chunks can be costly, however
- Experiences with LFS
  - + LFS dramatically decreased time to allocate chunks
  - + Faster sequential writes due to constant time chunk allocation
  - – Slower updates due to poor utilization of page cache when doing copy on write for updates
  - – Introduces costly GC operation

Beyond Basics

- Segment controller
  - heavy lifting data structure
  - manages a pool of segments
  - tracks segment utilization for efficient GC
- Segment
  - each segment maps its data blocks to the inodes that own them
  - keeps a write head that allows for constant time chunk allocation
  - finding and garbage collecting low utilization chunks can be costly, however
- Experiences with LFS
  - + LFS dramatically decreased time to allocate chunks
  - + Faster sequential writes due to constant time chunk allocation
  - – Slower updates due to poor utilization of page cache when doing copy on write for updates
  - – Introduces costly GC operation

Beyond Basics

- Segment controller
  - heavy lifting data structure
  - manages a pool of segments
  - tracks segment utilization for efficient GC

- Segment
  - each segment maps its data blocks to the inodes that own them
  - keeps a write head that allows for constant time chunk allocation
  - finding and garbage collecting low utilization chunks can be costly, however

- Experiences with LFS
  - + LFS dramatically decreased time to allocate chunks
  - + Faster sequential writes due to constant time chunk allocation
  - – Slower updates due to poor utilization of page cache when doing copy on write for updates
  - – Introduces costly GC operation

Beyond Basics

- Segment controller
  - heavy lifting data structure
  - manages a pool of segments
  - tracks segment utilization for efficient GC
- Segment
  - each segment maps its data blocks to the inodes that own them
  - keeps a write head that allows for constant time chunk allocation
  - finding and garbage collecting low utilization chunks can be costly, however
- Experiences with LFS
  - + LFS dramatically decreased time to allocate chunks
  - + Faster sequential writes due to constant time chunk allocation
  - – Slower updates due to poor utilization of page cache when doing copy on write for updates
  - – Introduces costly GC operation

Beyond Basics

- Segment controller
  - heavy lifting data structure
  - manages a pool of segments
  - tracks segment utilization for efficient GC
- Segment
  - each segment maps its data blocks to the inodes that own them
  - keeps a write head that allows for constant time chunk allocation
  - finding and garbage collecting low utilization chunks can be costly, however
- Experiences with LFS
  - + LFS dramatically decreased time to allocate chunks
  - + Faster sequential writes due to constant time chunk allocation
  - – Slower updates due to poor utilization of page cache when doing copy on write for updates
  - – Introduces costly GC operation

Beyond Basics

- Segment controller
  - heavy lifting data structure
  - manages a pool of segments
  - tracks segment utilization for efficient GC

- Segment
  - each segment maps its data blocks to the inodes that own them
  - keeps a write head that allows for constant time chunk allocation
  - finding and garbage collecting low utilization chunks can be costly, however

- Experiences with LFS
  - + LFS dramatically decreased time to allocate chunks
  - + Faster sequential writes due to constant time chunk allocation
  - – Slower updates due to poor utilization of page cache when doing copy on write for updates
  - – Introduces costly GC operation

Beyond Basics

- Segment controller
  - heavy lifting data structure
  - manages a pool of segments
  - tracks segment utilization for efficient GC
- Segment
  - each segment maps its data blocks to the inodes that own them
  - keeps a write head that allows for constant time chunk allocation
  - finding and garbage collecting low utilization chunks can be costly, however
- Experiences with LFS
  - + LFS dramatically decreased time to allocate chunks
  - + Faster sequential writes due to constant time chunk allocation
  - – Slower updates due to poor utilization of page cache when doing copy on write for updates
  - – Introduces costly GC operation

Beyond Basics

- Segment controller
  - heavy lifting data structure
  - manages a pool of segments
  - tracks segment utilization for efficient GC
- Segment
  - each segment maps its data blocks to the inodes that own them
  - keeps a write head that allows for constant time chunk allocation
  - finding and garbage collecting low utilization chunks can be costly, however
- Experiences with LFS
  - + LFS dramatically decreased time to allocate chunks
  - + Faster sequential writes due to constant time chunk allocation
  - – Slower updates due to poor utilization of page cache when doing copy on write for updates
  - – Introduces costly GC operation

Outline

1. Introduction
2. Architecture
3. Beyond Basics
4. Challenges
5. Performance Benchmark
6. Conclusion
7. Questions

Challenges

- Memory mapped interface
- Dropped pointer
- Off by one

Language choice: C++ 11

+ proper exception handling
+ proper memory management

Challenges

- Memory mapped interface
- Dropped pointer
- Off by one

Language choice: C++ 11

+ proper exception handling
+ proper memory management

Challenges

- Memory mapped interface
- Dropped pointer
- Off by one

Language choice: C++ 11

+ proper exception handling

+ proper memory management

Challenges

- Memory mapped interface
- Dropped pointer
- Off by one

Language choice: C++ 11

+ proper exception handling

+ proper memory management

Challenges

- Memory mapped interface
- Dropped pointer
- Off by one

Language choice: C++ 11

+ proper exception handling

+ proper memory management

Outline

1. Introduction
2. Architecture
3. Beyond Basics
4. Challenges
5. Performance Benchmark
6. Conclusion
7. Questions

Performance Benchmark

- Synchronous memory mapped
- Asynchronous memory mapped
- Non-memory mapped

Outline

1. Introduction
2. Architecture
3. Beyond Basics
4. Challenges
5. Performance Benchmark
6. Conclusion
7. Questions

Conclusion

Outline

1. Introduction
2. Architecture
3. Beyond Basics
4. Challenges
5. Performance Benchmark
6. Conclusion
7. Questions

Questions?