

# Energy-Efficiency and Storage Flexibility in the Blue File System



Salke Hartung

Advanced Topics in  
Computer Networking  
2007/2008



## Contents

- 1) Introduction
- 2) Related work and other approaches
- 3) Basic ideas and design goals
- 4) Implementation details
- 5) Evaluation
- 6) Conclusion and open questions



## Introduction – Today's storage situation on mobile systems

- Past: „Simple“ mobile systems with internal harddisk and network connection (FileServer access)
- Today: Additional storage possibilities like USB-Sticks, flash memory cards, external disk drives, etc.



## Introduction – Existing problems

- Limited battery lifetime:  
Additional storage devices consume lots of power
- Power management can result in performance lacks or delays  
e.g.: Spinup of harddisk, reenabling of network interface
- Switching power mode transitions consumes power
- Inconsistent file copies  
Manual file copies done by the user can lead to inconsistent data

Needed:

A method to allow storage-flexibility that assures low power consumption



### Related work

- Other approaches just focused on flexibility OR low power consumption, but never both
- Hardwarebased power-saving approach: Hybrid-Disk, uses large cache memory and writes data back from time to time → disk doesn't need to run all the time
- Distributed file systems like Coda[2] and AFS[3] make it possible to create flexible storage, but don't have energy-efficiency as a prior design goal
- PersonalRAID[4] can be used to keep data consistent between different computers for the same user, but neither this is a distributed file system which improves performance nor it focuses on energy-efficiency

## Basic ideas of the Blue File System

- Distributed file system
- Primary replica of each file resides on a fileserver
- Local storage keeps secondclass copies of all files and is just used to improve performance
- Multiple storage devices keep the same files, modifications must be written to all storage devices and the fileserver
  - data can be retrieved from any device, BlueFS will choose the least costly device
  - „write to many – read from any“ – strategy
- It seems counterproductive to write modifications of one file to many storage devices with regard to the aim to save energy
  - find out in Evaluation section



## Design goals - Overview

- 3 basic design goals in BlueFS:
  - Constant change of storage devices:  
Adapt easily to performance and energy characteristics of all storage types (local, portable and network storage)
  - Extend battery lifetime
  - Transparent mobile storage with automatic synchronization



## Design goals – Constant Change

- Performance of different storage devices varies due to:
  - Deployment of new storage devices  
e.g.: Characteristics of flash memory differs from characteristics of a fixed hard disk
  - Impact of power management  
Power management leads to delays when reaccessing a storage device, e.g.: spinup time for a hard disk, reenabling network interface
  - Network variability  
Performance of remote storage depends on the network conditions
  
- To handle device performance variations, BlueFS monitors their performance at runtime and the power state of each device, see implementation section



## Design goals – Battery Lifetime

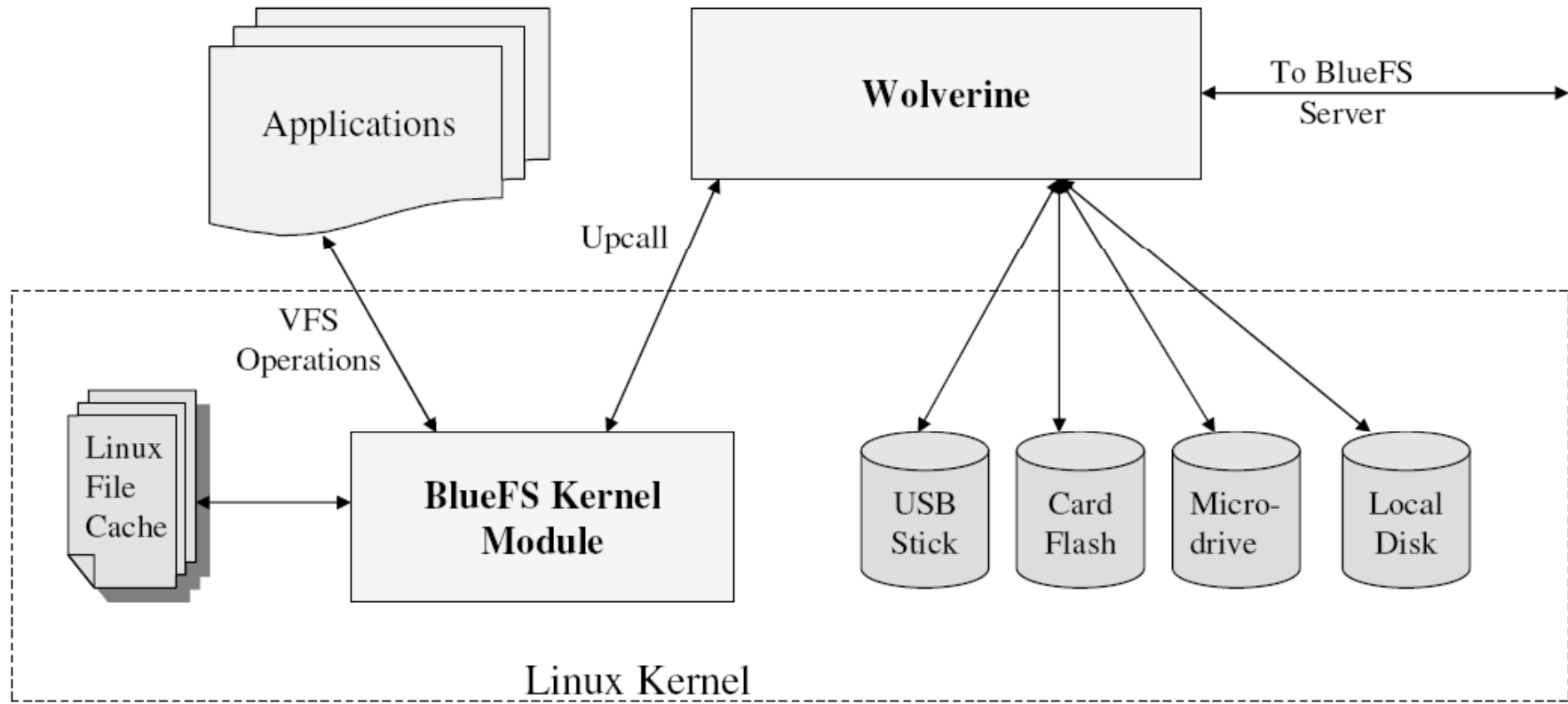
- Energy is mostly the bottleneck on mobile computers
- Constantly added features and capabilities for mobile computers consume more and more power
- In contrast: Improvement of battery capacity develops slowly
- Current file systems don't focus on energy consumption and therefore energy is wasted on data access
- BlueFS contains mechanisms to reduce energy consumption:
  - Aggregation of write operations
  - Consideration of energy costs when trying to read data
- Details follow in implementation section



## Design goals – Transparent Storage

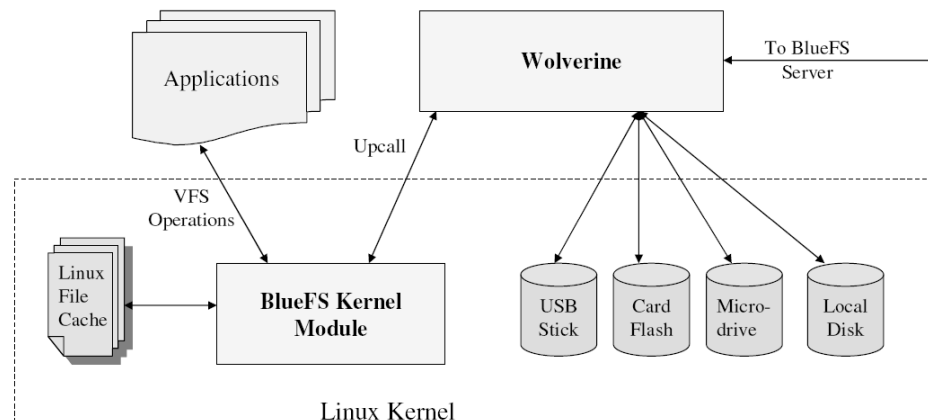
- Performance and capacity of portable storage devices rapidly increased in the last years, but:
- Users manage files on portable storage devices with manual copying or special synchronization applications
  - Data is lost if the device is damaged or stolen
  - Manual copying can lead to inconsistent file copies
- BlueFS only stores second-class replicas of files on portable and local storage and just uses these for performance improvements
- BlueFS manages attachment and detachment of portable storage and guarantees that latest versions of all files are written to it
  - minimizing potential of inconsistent file copies
  - possible to take your work home

# Implementation



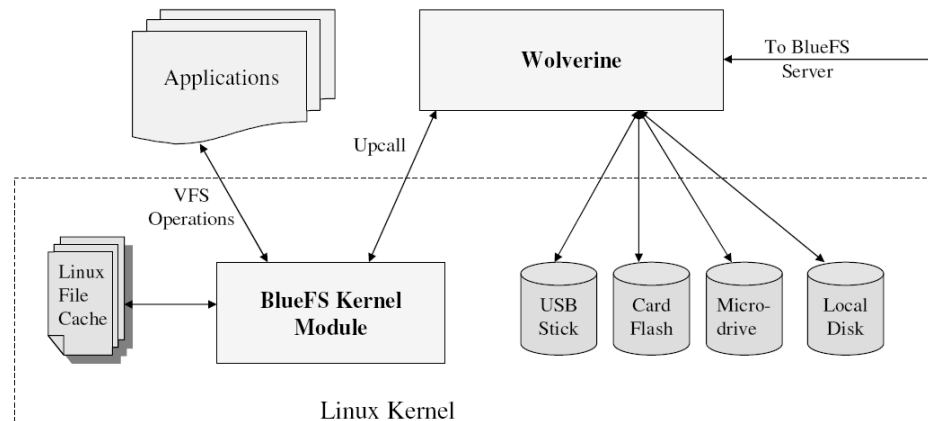
## Implementation – Kernel Module

- Minimal kernel module
- Intercepts VFS calls and redirects them to „Wolverine“, a user level daemon that is responsible for reading and writing data in BlueFS
- Kernel module just acts as a pipe to the daemon and as an interface to VFS
- Interacts with the linux file cache



## Implementation – „Wolverine“

- Functionality of BlueFS resides in the userlevel daemon Wolverine
- Receives operation calls from the kernel module
- Communicates with the file server and all storage devices that are directly connected to the machine
- Responsible for determining the least cost (=least energy) device when reading data and aggregating write operations





## Implementation – „Wolverine“ – Write Queues

- Wolverine keeps a write queue in memory for every connected storage device, basically it's an operation log, every operation instruction from the kernel module is added to the tail of the queue and removed by the FIFO principle
- Write queues can be used to aggregate several write operations to create bursty write accesses that don't force the storage device to be always active → creates long periods of idle time for storage devices in which they can go to sleep mode and save power
- Write queue records can be shared between different queues to save memory, when a queue exceeds a maximum of 75% of max queue size, the queue is flushed immediately

## Implementation – „Wolverine“ – Reading data

- Wolverine tries to read data from the least energy consuming device with the best performance
- Wolverine keeps an estimate value of the current time to access a storage device using the formula:  
$$\text{new\_est.} = (a)(\text{this\_msmt.}) + (1-a)(\text{old\_est.})$$
- $a$  is chosen based on empirical observation of practise tests:  
network storage:  $a = 0.1$ , local storage:  $a = 0.01$
- In addition to that: device power mode of each device is monitored and for each power mode an estimate is calculated using the formula above
- Calculation of estimates just focuses on performance issues, not saving energy



## Implementation – „Wolverine“ – Reading data (ctd.)

- Wolverine considers energy usage when accessing storage devices for reading data
  - Energy consumption when accessing a device for reading is (nearly) constant for each device
  - Wolverine uses „hardcoded“ characterization to predict energy costs
  - For each device that is integrated into BlueFS this characterization has to be configured once based on benchmarks or vendor device specifications
- 
- Wolverine checks the write queue of a device for file modifications and has to write the updates first if data is modified that must be read

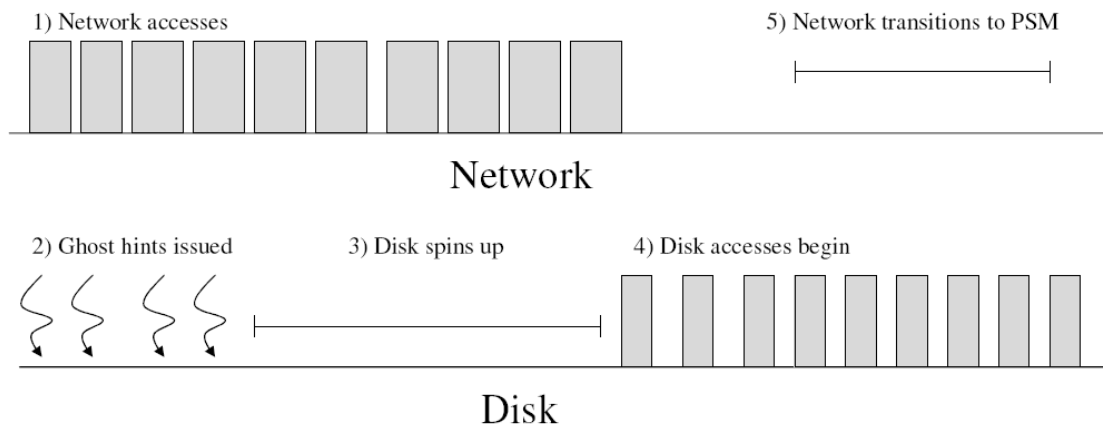


## Implementation - „Wolverine“ – Reading data (ctd.)

- To avoid that Wolverine accesses devices that don't contain the needed files BlueFS uses an „enode cache“
- A complete cache in memory that stores which files reside on which devices would be too large
- Instead just information about accessed files is stored
- Wolverine first checks the enode cache and skips devices that don't have entries for the files to read in the cache

## Implementation – Hiding access delays

- Wolverine integrates itself into the power management of the client to decide which devices should be accessed for fetching data
- Accessing storage devices in sleep mode would result in huge delays since the devices need to change to active-mode first
- To avoid these delays, BlueFS (Wolverine) fetches data from the server while it wakes up other devices
- When other devices are ready BlueFS stops fetching data from the file server and uses the cached files on the local storage devices





## Implementation – Cache consistency

- Since multiple clients can work on the same files it is possible that conflicts occur
- BlueFS file server uses a metadata version system to tag different file versions and recognize conflicts → version number of a file object in BlueFS is increased on every modification
- Whenever a conflict is recognized by BlueFS the user has to resolve it
- On every modification of a file by a client the server notifies all other devices of the other clients to invalidate their file caches since their versions of the modified file are no longer valid and have to be updated



## Implementation – Affinity

- User can define a set of files that is always present with latest versions on a storage device
- Good for „taking work home“
- BlueFS calls this feature „affinity“
- Commandline tool provides supports for adding and removing files or whole subtrees to a per-device refetch list
- Wolverine scans this list every 5 minutes and stores missing files and latest versions on the device



### Implementation – „Wolverine“ – Add/Remove-ing devices

- Basically, when a device is detached Wolverine stops recording new operations for the change log and flushes the write queue of the device
- For affinity purposes a „clean shutdown record“ is written that indicates that latest versions of all files were written to the device
- Detachment message is sent to server
  
- On device attachment Wolverine checks for the shutdown record, if it is not found, the device is synchronized
- Attachment message is sent to server

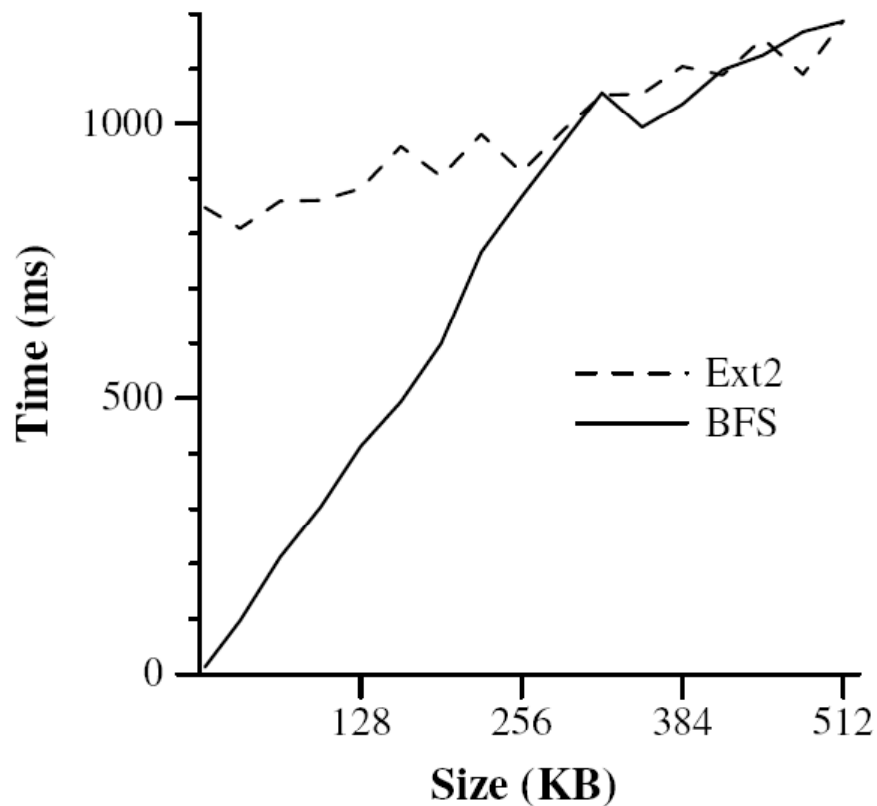


## Evaluation – Test Environment

- 2 test systems:
  - Laptop, Pentium 3 700Mhz, 128MB, 10GB disk drive, Linux 2.4.28-30
  - Handheld, 206Mhz StrongARM, 64MB Dram, 32MB Flash memory
- Both use 11 Mbit 802.11b adapter and have a portable 1GB disk drive attached
- Both use the same file server, Pentium 4 3000 Mhz, 1GB, 120GB disk drive, 802.11b

## Evaluation – Results

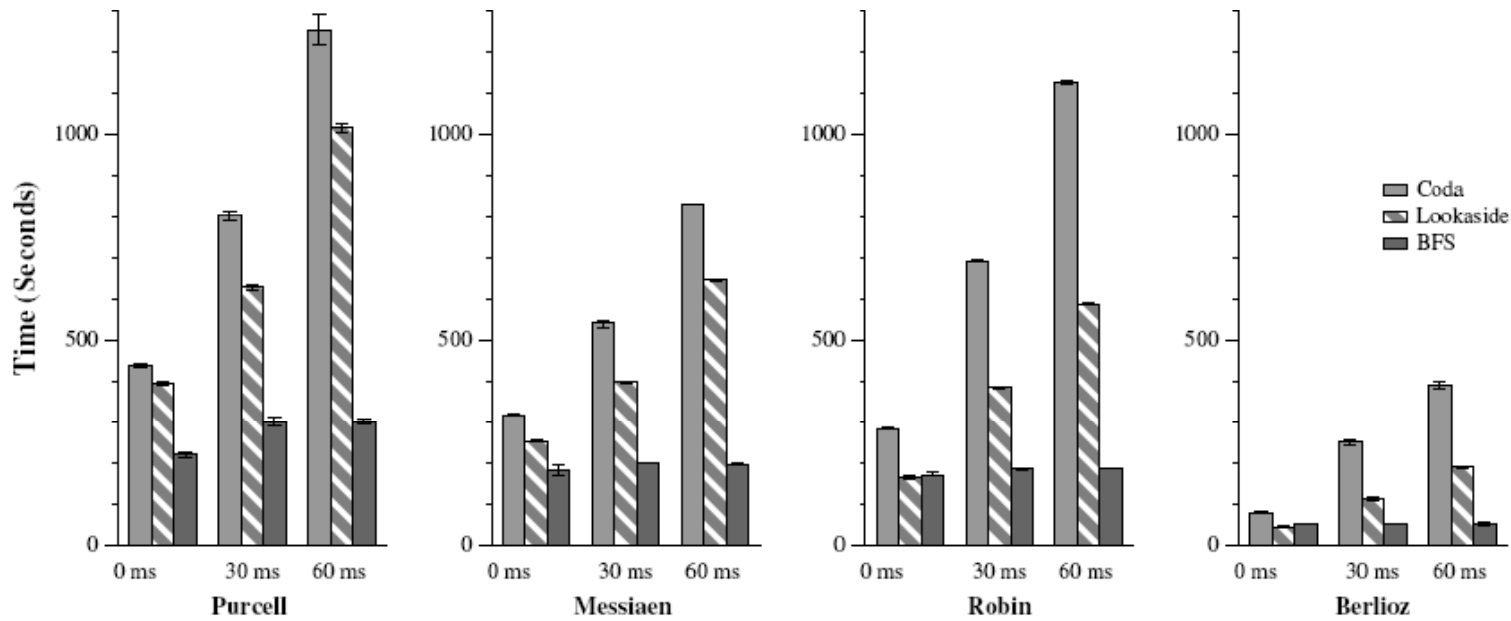
„How well does BlueFS hide access delays caused by power management?“





# Evaluation – Benefiting from local cache

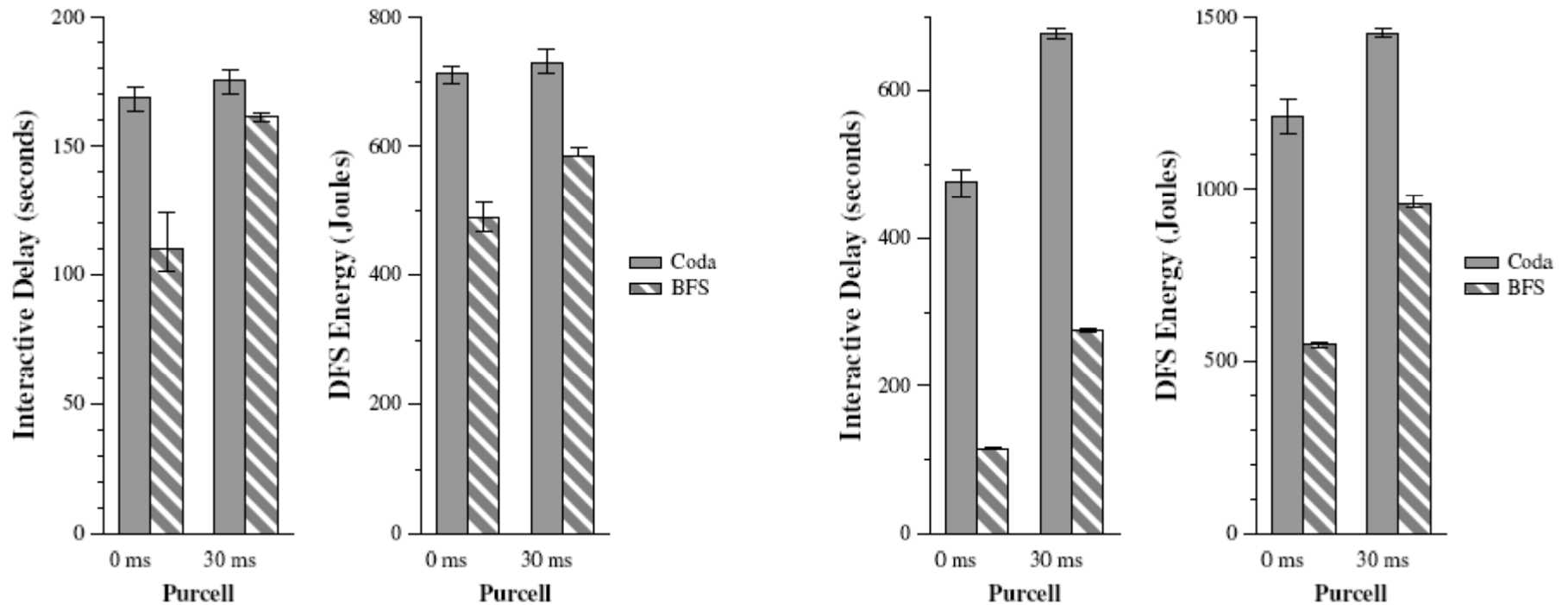
„How well does BlueFS benefit from portable storage as caching devices?“



Trace	Number of Ops.	Length (Hours)	Update Ops.	Working Set (MB)
purcell	87739	27.66	6%	252
messiaen	44027	21.27	2%	227
robin	37504	15.46	7%	85
berlioz	17917	7.85	8%	57

## Evaluation – Energy-efficiency

„How well does BlueFS save energy when accessing the file system?“



Left: with all files cached

Right: just 50% of files cached, files not found on local storage are retrieved from the server without accessing the local devices



## Conclusion and open questions

- BlueFS is a distributed file system that focuses on performance and energy consumption
- Maybe oversized for home applications
- Test environment should have considered more different storage devices connected to a machine
- Power consumption analysis just done for the handheld, what about other mobile systems, more tests would be useful
- BlueFS considers the least cost device when reading data, wouldn't the same idea apply when writing data?



## References

- [1] Edmund B. Nightingale, Jason Flinn. *Energy-Efficiency and Storage Flexibility in the Blue File System*. University of Michigan, 2004
- [2] J.J. Kistler and M. Satyanarayanan. *Disconnected operation in the Coda file system*. ACM Transactions on Computer Systems, Feb 1992
- [3] *OpenAFS project*, <http://www.openafs.org/>
- [4] S. Sobti, N. Garg, C. Zhang, X. Yu, A. Krishnamurthy, R. Wang *PersonalRAID: Mobile Storage for Distributed and Disconnected Computers*, In Proceedings of the 1st Conference on File and Storage Technologies, Monterey, California, pages 159.174, Jan. 2002