Florida International University

TFS: A Transparent File System for Contributory Storage

James Cipar Mark D. Corner Emery D. Berger

Luis Useche
lusec001@cs.fiu.edu

4/3/07

Overview

- Introduction
- TFS: Design
- TFS: Implementation
- Evaluation
- Conclusions
- References

Introduction

Users share resources from local machine to different uses:

Introduction

- The sharing of CPU and memory is widely use. In contribution of storage is not the case.
- Reasons:
  - Low performance.
  - Users are generally reluctant to relinquish their free space.

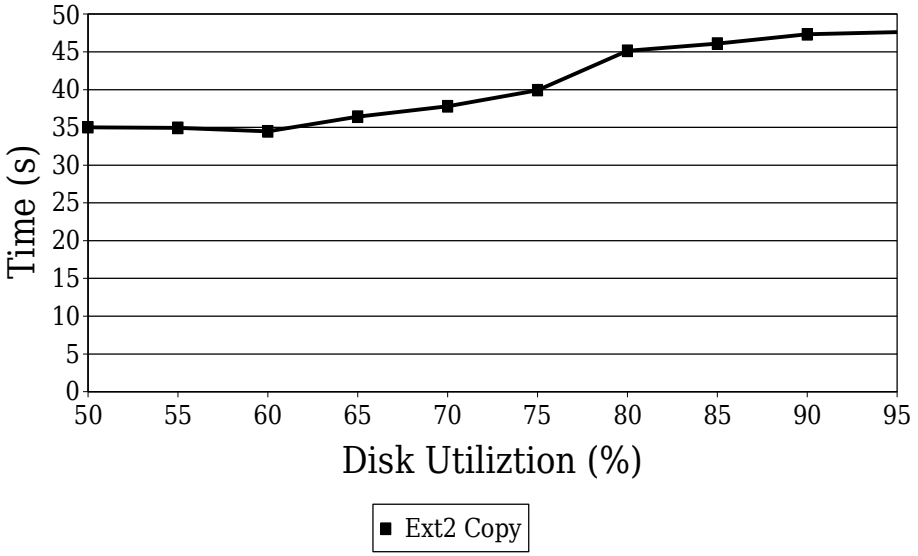
How is the I/O performance affected?

- Two types of contributions: static and dynamic.
- Watermarking: Preset a percent of disk to use.
- Problem: The file system should delete the contributory files when space needed.

How is the I/O performance affected? II

- Problem: When the disk is almost full the FS allocation doesn't do a good work.
  - E.g. In an FFS throughput can drop 77% is the 75% of the disk is full.

Contributory effects



- When the disk utilization increase so does the file fragmentation
- More contribution, lower performance

Solution: Transparent File System (TFS)

- Goal: Design a transparent file system that does not affect the normal function of the local FS.
  - Performance: does not affect the allocation policy.
  - Capacity: does not decrease the size of the local FS space.

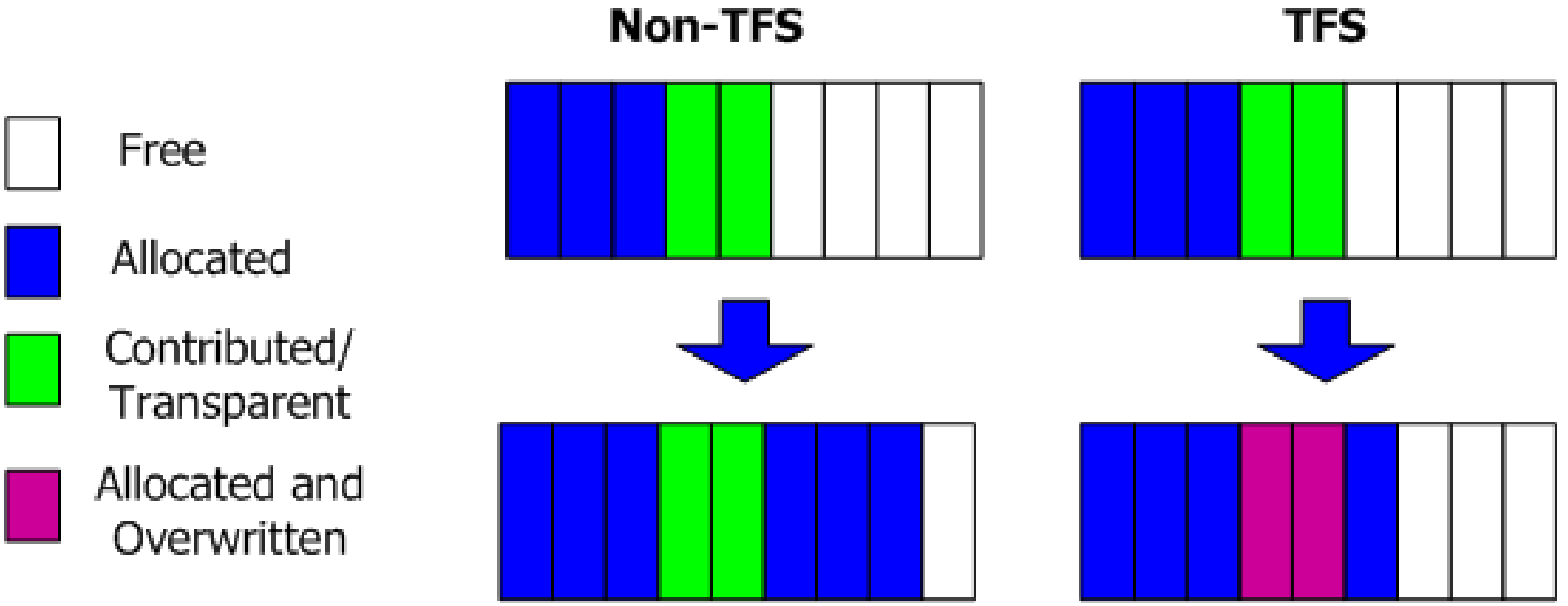
TFS Design

- Designers started TFS from a popular FS, ext2.
- New blocks type added:
  - transparent
  - free-and-overwritten
  - allocated-and-overwritten

Allocation policy

- Allocation policy treats transparent and free blocks equally.
  - Benefit: The allocation policy is not affected.
  - Drawback: Overwrite of the contributory files. Contributory applications should use replication to prevent data loss.

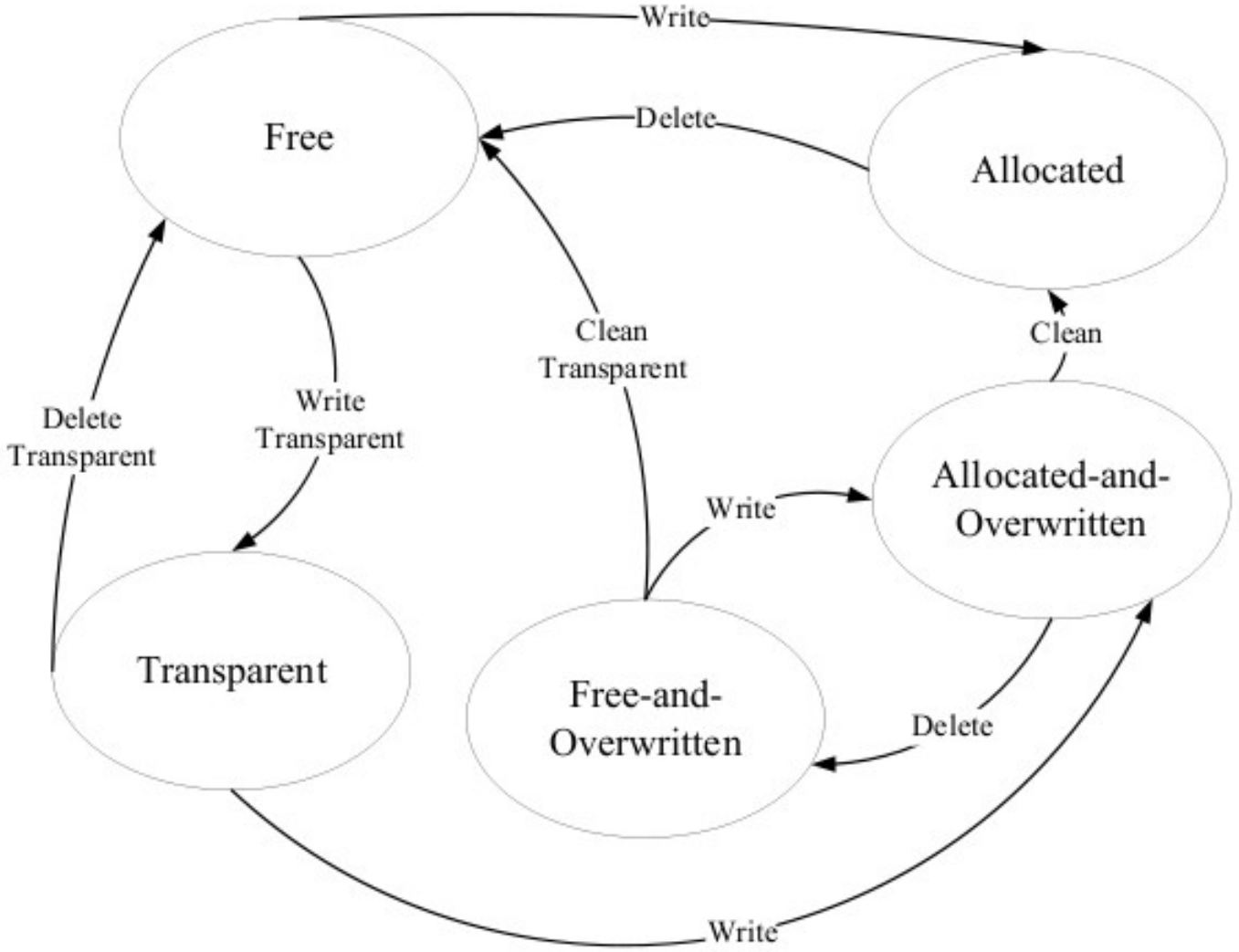
Allocation: An example



Management of states

- When an ordinary file writes to a transparent block, the block is marked as: allocated-and-overwritten.
- Transparent data can only be written to free blocks.

States Transition



Cleaning of Overwritten blocks

- The cleaning is done when the transparent files are open.
- TFS scans the blocks of the opened transparent file:
  - if some block is overwritten, returns an error and deletes the corresponding inode.

Cleaning of Overwritten blocks II

- Transparent files with overwritten blocks and never opened leads to unused space.
- Solution: user space application that periodically opens transparent files.

TFS states overhead

- For each block TFS store its state.
- Representation of state: 3 bits.
- In a 100GB FS with blocks of size 4kB, the array of states occupies 6.25MB.
- Just part of this array will be in memory at any given time.

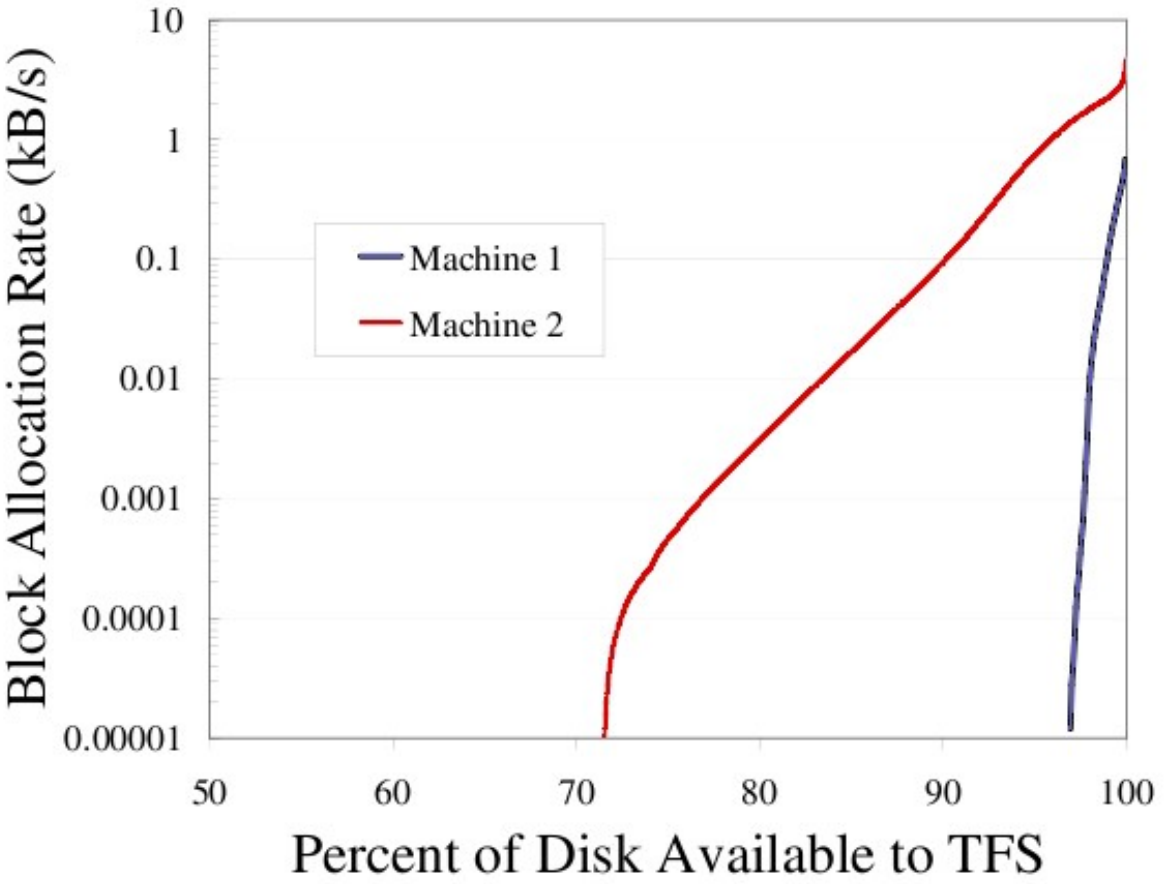
Performance concession

- What happen is a transparent file is open and a user process overwrites it?
  - TFS yields to open files.
- What happen if a transparent meta-data is overwritten?
  - The transparent meta-data is stored in the FS regular space.

Transparent data Allocation

- The ordinary data overwrites transparent blocks.
- TFS wants to avoid FS hot spots and minimize the overwritten transparent blocks.

Hot Spots

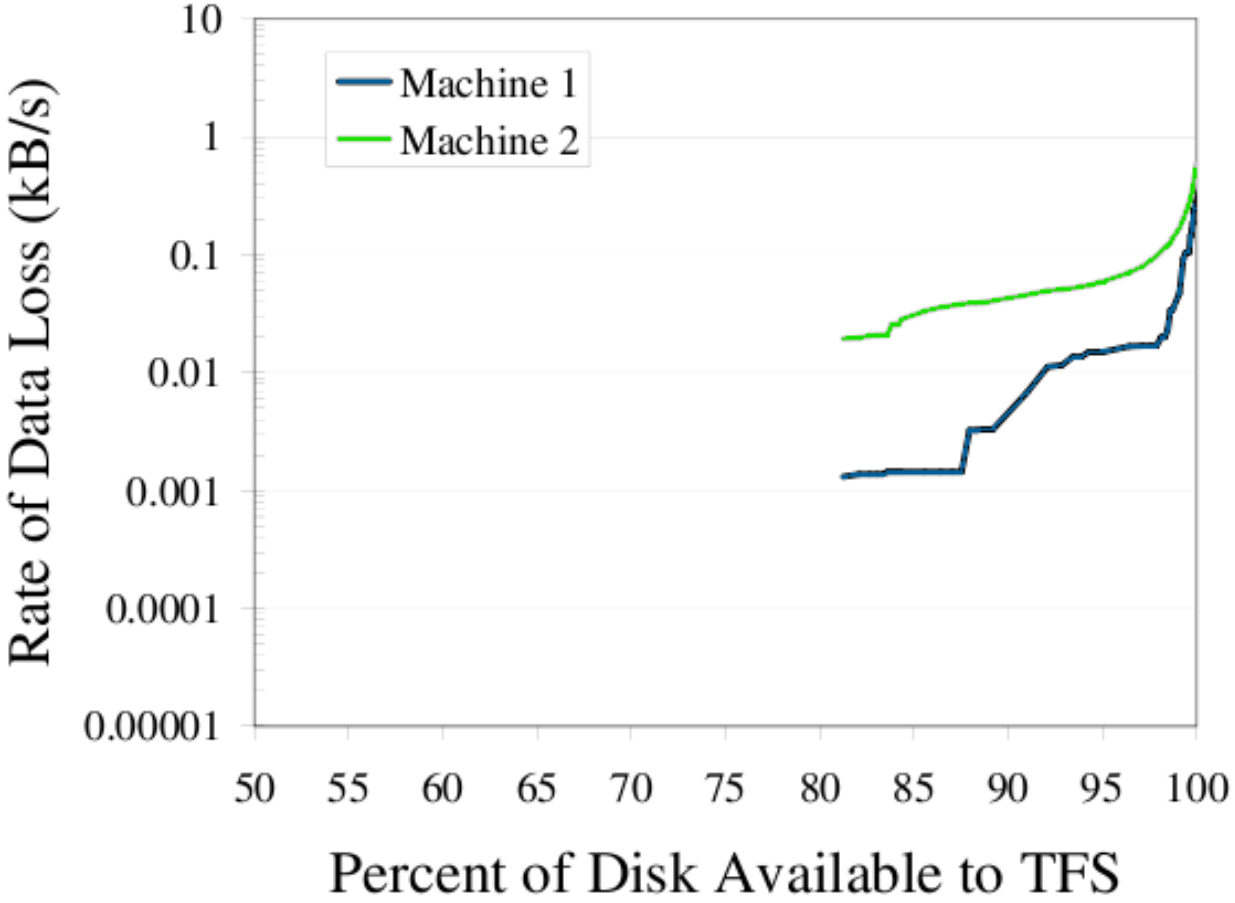


- Block allocation shows high locality

Avoiding High locality

- It keeps a histogram of the usage of each block.
- With this information and some desire rate of transparent overwriting is easy to obtain the fraction of blocks to avoid, *f*.

Data loss vs. TFS disk usage



- When TFS avoid 3% of disk, the used disk will be overwritten at 0.1kB/s

TFS: Implementation

- It was implemented for the Linux kernel 2.6.13.4.
- Used by one of the developers for 6 months to store his home and use Freenet.
- mke2fs, fsck, open, df modified.
- Aprox 900 lines of new code.

Evaluation

- Goal: Measure each of the contributory storage systems with different metrics:
  - The amount of storage contributed.
  - The effect on the block allocation policy.
  - Overall performance effect.

Evaluation: Contributed Storage Capacity

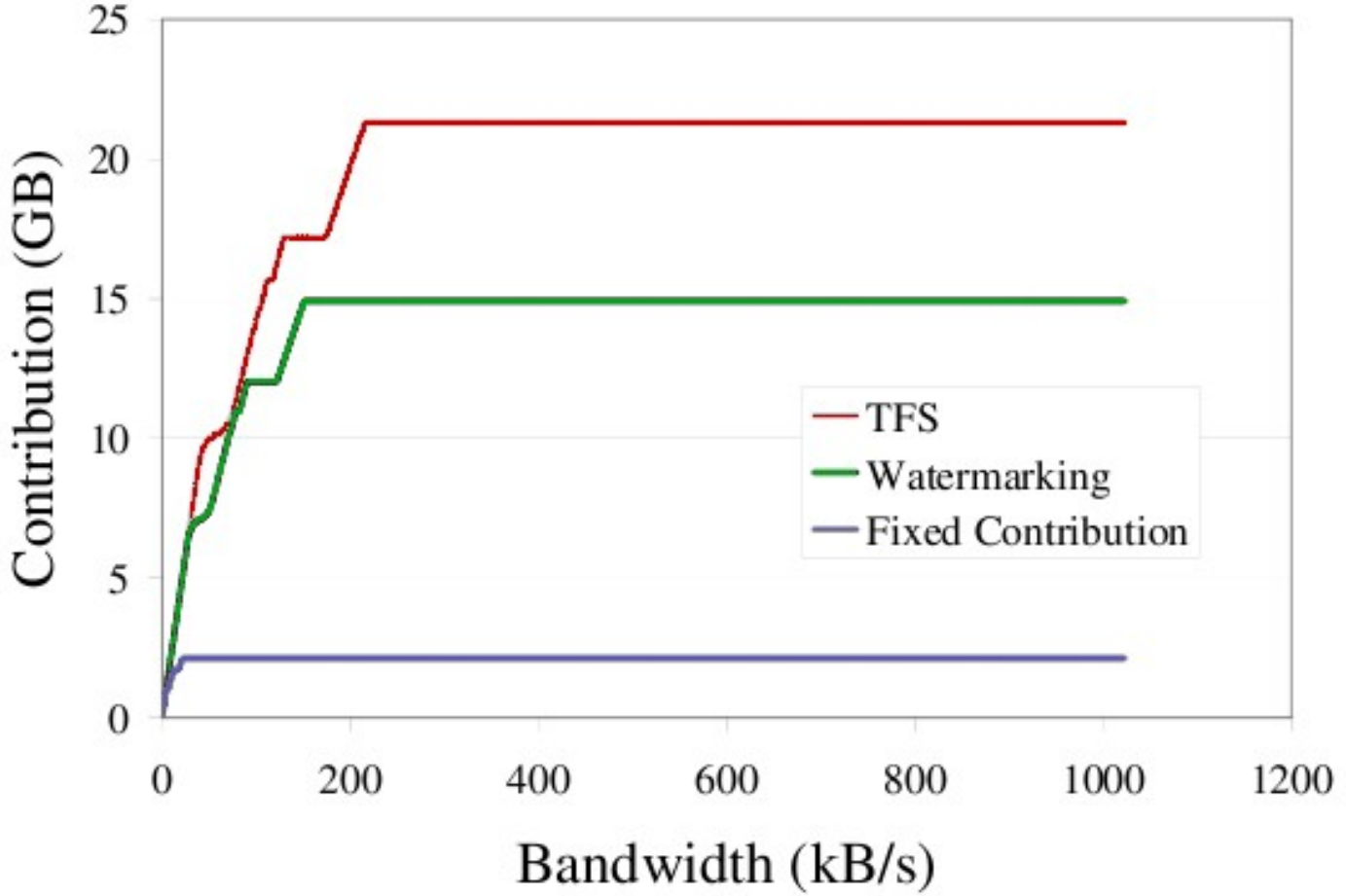
- Assuming a network of identical machines each with 100GB and 50% of disk full.
- 3 contributory systems analyzed:
  - Static contribution: 5% of contribution.
  - Dynamic system (watermarking): 35% of contribution.
  - TFS: 47% of contribution. Determined using the results from the data loss graph.

Evaluation: Contributed Storage Capacity II

- Two traces evaluated from different type of networks:
  - Microsoft corporate network
  - Skype super-peers
- The contributed capacity calculated by:

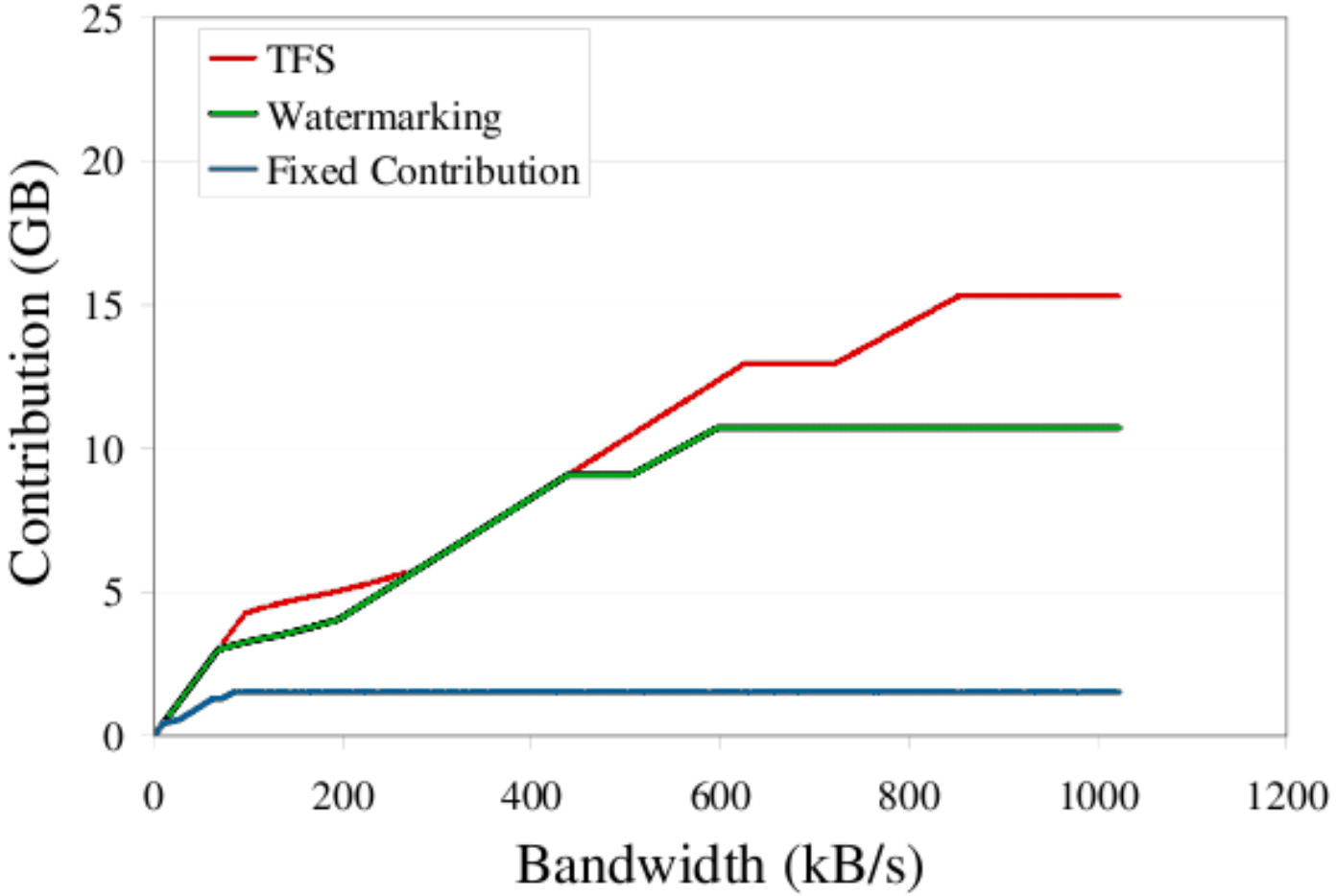
$$C = A(T,r)(B - F)$$

Contributed Capacity: Microsoft corporate network



- In reliable systems, TFS contributes much more than other solutions.

Contributed Capacity: Skype super-peer

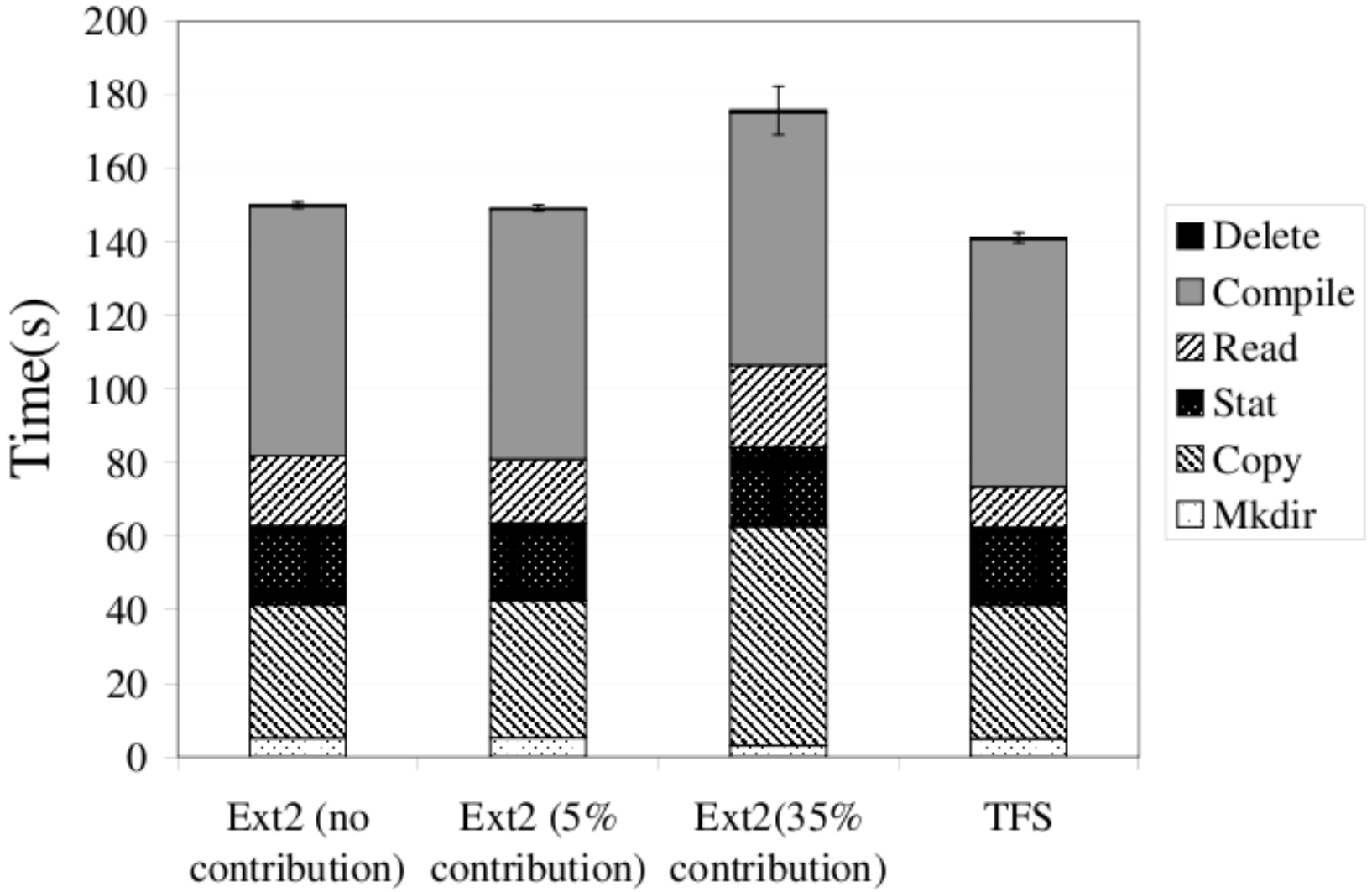


- In unreliable networks, TFS behaves at least as good as other solutions.

Evaluation: Performance Impact

- Disk fill 50% with the `/usr` directory files.
- Four cases measured:
  - No contribution (baseline)
  - 5% of contribution
  - 35% of contribution (Dynamic case)
  - TFS: The disk is filled with transparent data.
- Andrew benchmark ran.

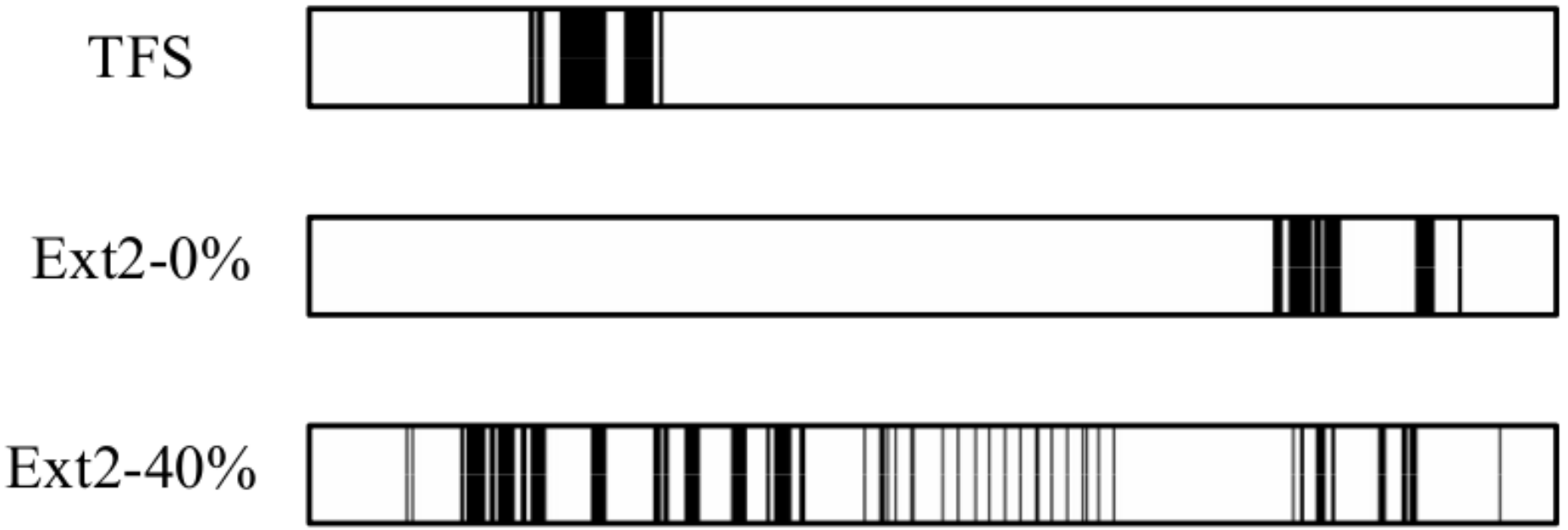
Andrew benchmark results



Evaluation: Block allocation layout

- Two Ext2 machines with 0% and 40% of contributory space and one with TFS.
- Machines 50% filled with ordinary data, rest with transparent data and finally Andrew data.

Block allocation II



- Different disk layouts

Conclusions

- The key benefit of TFS is that it does not affect the local allocation.
- TFS avoids hot-spots keeping track of each block.
- In the worst case TFS behaves as good as other solutions.

Conclusions II

- In the best case TFS provides 40% more storage than the best user-space technique.

References

- James Cipar, Mark D. Corner and Emery D. Berger, “TFS: A Transparent File System for Contributory Storage”, *FAST '07: 5th USENIX Conference on File and Storage Technologies*, (2007)
- Robin Harris, “Transparent File System – Can You See It?”, *http://storagemojo.com/?p=408*, (2007)