Opportunistic Storage Maintenance

UNIVERSITY OF TORONTO

George Amvrosiadis
Angela Demke Brown
Ashvin Goel

What is storage maintenance?

- Tasks that run periodically in the background
  e.g., backup and virus scans once a week
- Access large amounts of data
  e.g., process all files, scan every block
- Operate in the user or kernel level
  e.g., user-level backup, file system garbage collector

Why perform storage maintenance?

- Maintenance tasks offer essential guarantees

Guarantee	Periodic maintenance task
Reliability	Scrubbing, Write Verification
Availability	Backup, Data reorganization
Performance	Layout optimization
Security	Virus scanning
Storage Efficiency	Deduplication, Garbage collection

- Maintenance provides insurance against disaster
  - U.S. immigration unable to process visas for 2 weeks
  - Backups disabled during high tourist season
  - Hardware failure occurs (Murphy strikes again!)

When should maintenance be run?

- **Problem:** maintenance tasks can significantly impact foreground applications (workload)
  - Run alongside workload and access lots of data
  - Consequence: Increased seeks, cache pollution
- **Solution:** schedule tasks *around* workload
  - Perform during scheduled downtime
  - Perform while device is idle, at low priority

Sufficient idle time is needed to reduce maintenance impact

Too little time, too much work

- Idle time is expensive
  - No one wants extended downtime for maintenance
  - Consolidation in the cloud reduces idle time
- Maintenance takes too much time
  - Full backups performed every 1-4 days
  - A 6TB hard drive takes 10 hours to scan
- **Too many tasks**
  - **Each task adds to total maintenance I/O**

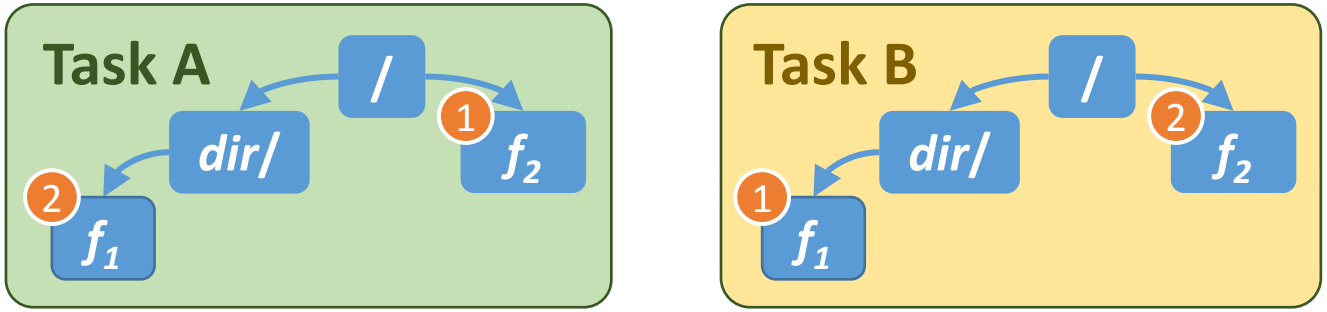
Too many tasks

- Tasks often access the same data
  - E.g., backup and defrag the same file system
  - Caching should be able to exploit data reuse

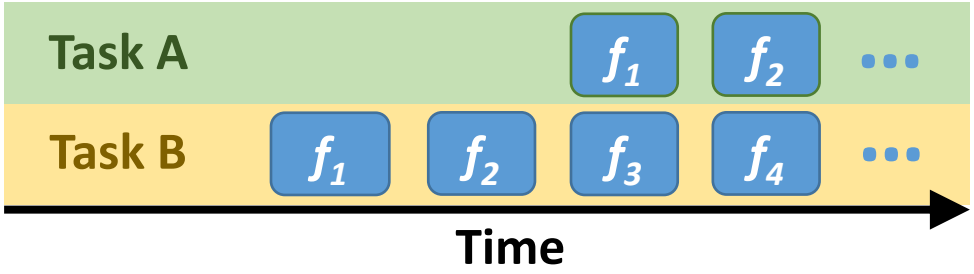
- **Problem:** cached data gets replaced before reuse
  - Each task accesses more data than fits in cache
  - Tasks process data independently

Effect of independent processing

- Different processing order: breadth- vs depth-first



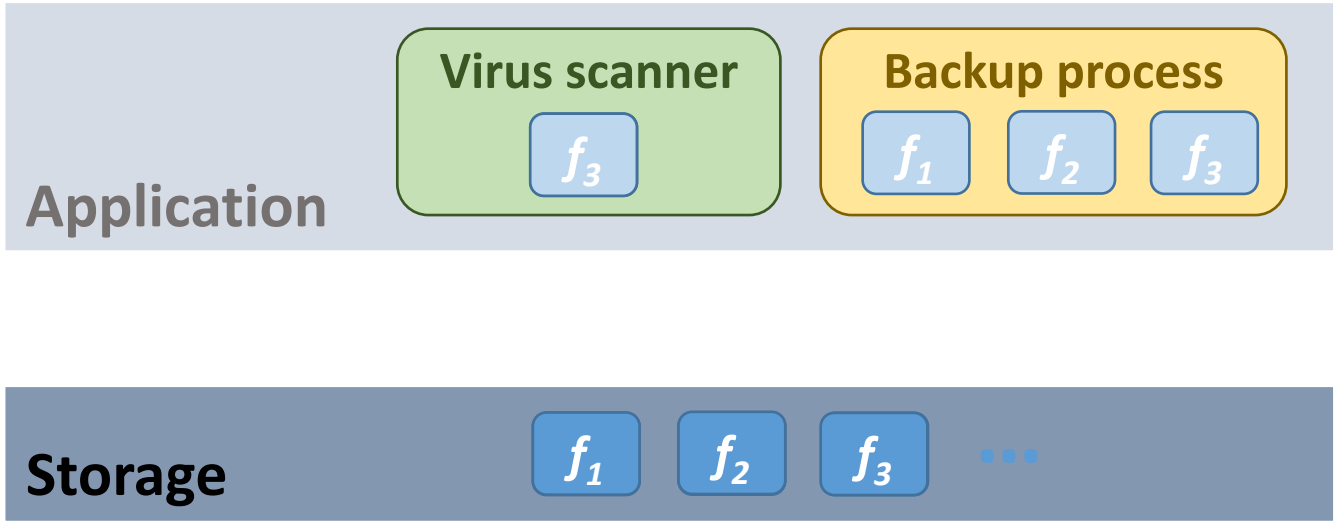
- Same processing order, run at different times



Maintenance I/O proportional to number of tasks, even when tasks access the same data

Reducing maintenance I/O

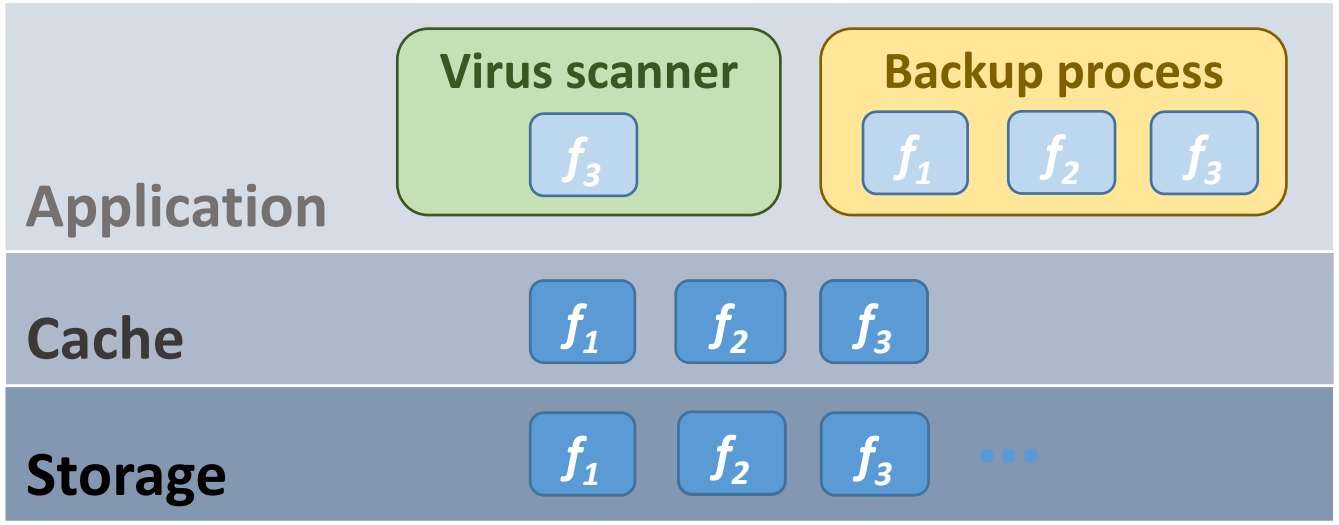
- Insight: correctness is not tied to processing order
  - E.g., backing up file f_1 before, or after f_2 doesn't matter
- Adapt order based on other tasks' accesses



How can a task determine what's accessed by other tasks?

Look in the cache!

- Recently accessed data is cached
- Opportunistically process cached data



How can a task find out what's in the OS cache?

Exposing cache information

- Operating systems cache data at page granularity
- Need to expose which pages are cached

1) We track changes to the status of cached pages

Page status	Description
Added	Added in cache
Removed	Removed from cache
Dirtied	Dirty bit set
Flushed	Dirty bit cleared

2) tasks register interest

Tell me about…

- ✓ **Added**
- ✓ **Removed**

Occurring at…

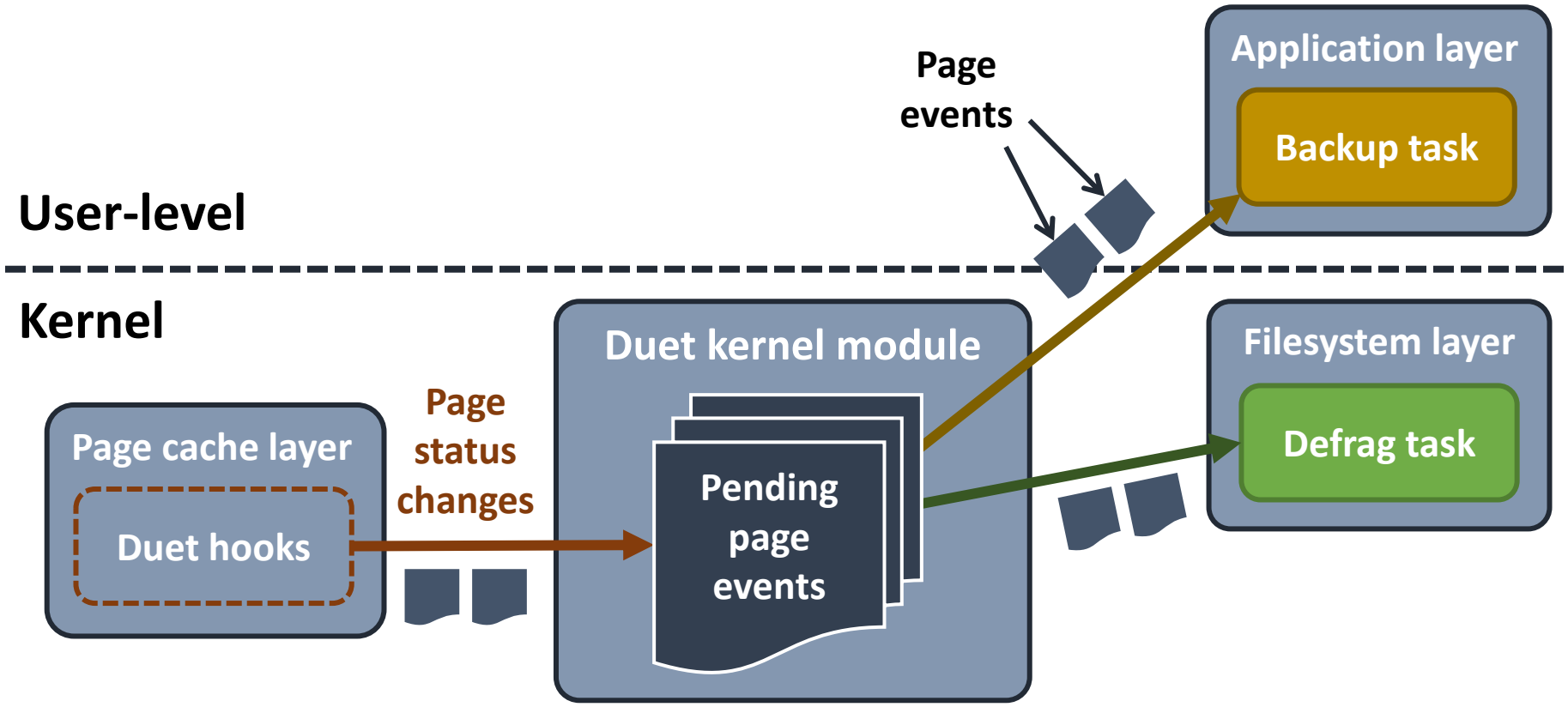
/home

3) tasks poll for events

Page event

- **Inode #**
- **Page offset**
- **Page status**

The Duet Framework



How do tasks use page events?

Using page cache events

- Goal: Use events to achieve more efficient order
- Tasks operate on **items** of different granularities

  e.g. files, extents, blocks, segments, …
- Page events expose what portion of an item is cached

- Example: file defragmentation task
  - Uses *Added, Removed* events to track cached file pages
  - Prioritizes processing of files with most cached pages
- **Opportunistic processing reduces required I/O**

An example opportunistic task

```
1  Algorithm example_task()
2      while True do
3          /* Process files opportunistically */
4          while True do
5              duet_fetch(sid, events)
6              prioqueue_update(sid, pq, events)
7              file ← prioqueue_dequeue(pq)
8              handle_file(file)
9          end
10         file ← pick_next_file()
11         handle_file(file)
12     end
13     return
1  Procedure handle_file(file)
2      if duet_check_done(sid, file) then return
3      process_file(file)
4      duet_set_done(sid, file)
5      return
```

Fetch any new page events

Use page events to update priority queue of files with most cached pages

Pick next file in task's standard order

Ensure file hasn't been processed before

Opportunistic maintenance tasks

Task	Processing Order	Opportunistic version	Modified Lines of Code
Scrubber (Btrfs)	By block number	Don't scrub recently read blocks	75 LoC (2%)
Backup (Btrfs)	By inode number	Backup in-memory blocks first	140 LoC (3%)
Defragmentation (Btrfs)	By inode number	Prioritize files with cached blocks	95 LoC (8%)
Garbage collector (F2fs)	Cost function	Adjust cost function to account for cached blocks	150 LoC (11%)
Rsync (User level app)	Depth-first order	Prioritize files with most cached pages	300 LoC (<1%)

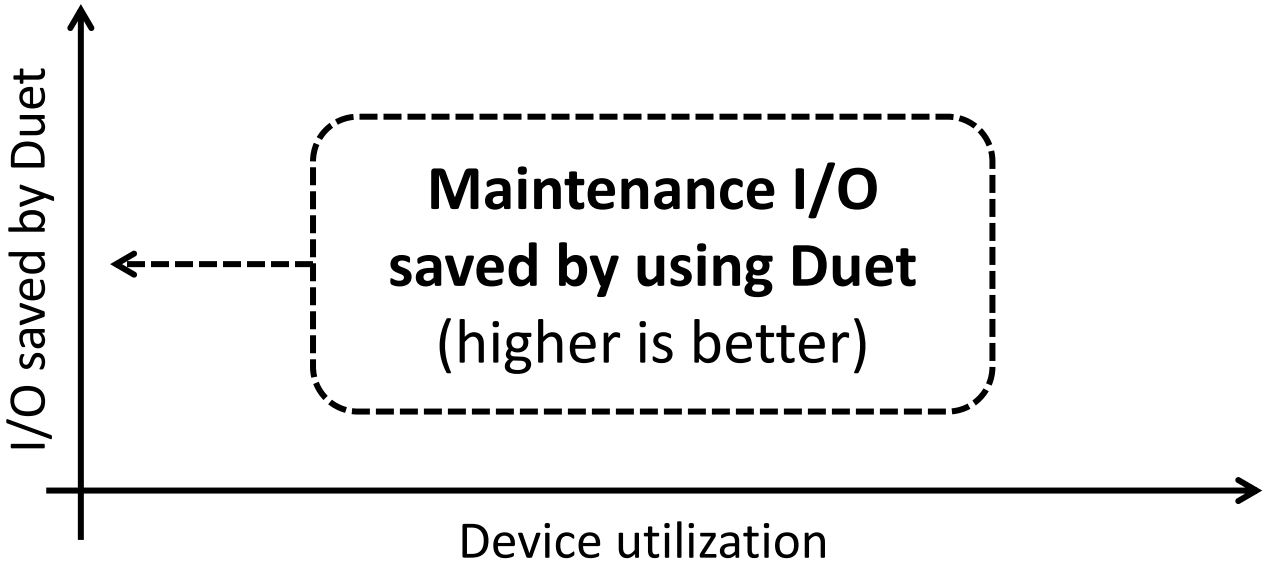
Evaluation Setup

Goal: Evaluate Duet's ability to reduce I/O

- Run maintenance tasks alongside Filebench workloads
- Measure maintenance I/O saved over 30 minutes
- Maintenance I/O is scheduled at idle times

Parameter	Description
Data overlap	Fraction of maintenance task data also accessed by workload
Device utilization	Amount of time the device is busy when running the workload alone

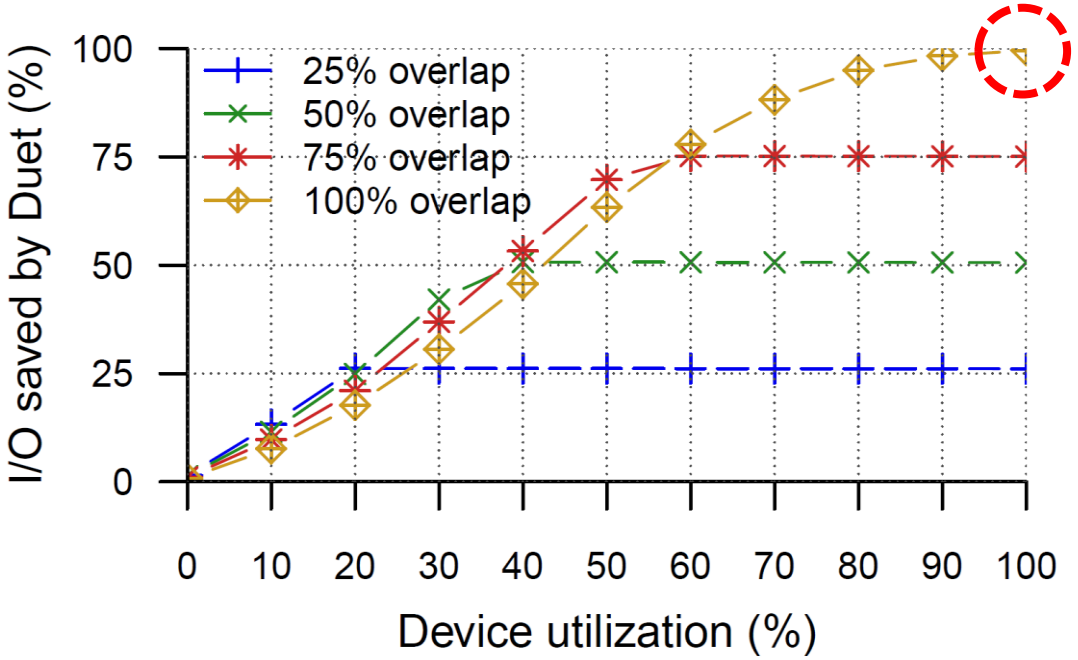
Evaluation metrics



Overlap w/ Maintenance	Maximum utilization	
	Baseline	Duet
25%		
50%		
75%		
100%		

Max. device utilization by workload to complete maintenance in 30mins
(higher is better)

Single task performance:
Btrfs backup

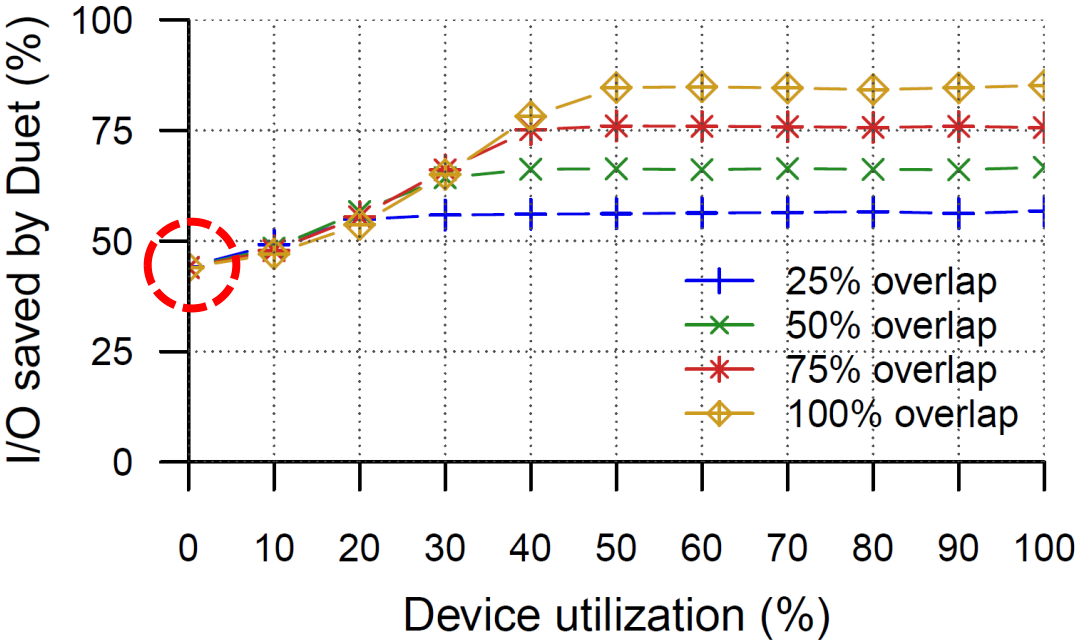


Workload overlap	Maximum utilization	
	Baseline	Duet
25%	40%	**50%**
50%	40%	**60%**
75%	40%	**70%**
100%	40%	**100%**

Duet exploits all opportunities to reduce I/O

Less idle time is needed for maintenance I/O

Running multiple tasks together:
Btrfs scrubbing, backup, and defragmentation



Workload overlap	Maximum utilization	
	Baseline	Duet
25%	**N/A**	**30%**
50%	**N/A**	**40%**
75%	**N/A**	**40%**
100%	**N/A**	**50%**

- Tasks can piggyback on one another
  - 3 tasks + idle device = up to 48% I/O reduction
  - 3 tasks + busy device = up to 80% I/O reduction

Duet overhead

- Low CPU overhead
  - Ran Filebench at full device utilization
    - Generates a high rate of page events
  - Registered for all page cache changes
  - Aggressively fetched events every 10 ms
  - Measured CPU overhead at ≤ 1.5%
- Small memory footprint
  - 64 bytes per page with pending events
    - 1.5% for 4K pages

Conclusion

- **Problem:** Current maintenance tasks work independently
- **Our approach:** Enable collaboration between tasks
  - Expose changes in cache state as events
  - Tasks poll for these events to detect cached data of interest
  - Tasks reduce I/O by reordering work to process cached data first

Source code: github.com/gamvrosi/duet