Cooperative I/O
A Novel I/O Semantics for Energy-Aware Applications

Andreas Weissel • Björn Beutel • Frank Bellosa

Department of Computer Science 4 (Operating Systems)
University of Erlangen
Martensstr. 1, 91058 Erlangen, Germany

{weissel, bnbeutel, bellosa}@cs.fau.de
Outline

■ A New I/O Semantics
■ Power Management of Hard Disks
■ Cooperative I/O
■ Implementation
■ Measurements
■ Conclusion
Cooperative I/O: Principle of Operation

- “Traditional” OS power management policies:
  - times of disk operations issued by user applications are unknown and cannot be influenced

- Cooperative I/O: more flexible timing of disk operations
  - *deferrable* and *abortable* I/O requests
  - new system calls (in addition to the original interface)
    - `read_coop(int fd, void *buf, size_t count, int time-out, int abort);`
    - `write_coop(int fd, void *buf, size_t count, int time-out, int abort);`
    - `open_coop(const char *pathname, int flags, int time-out, int abort);`
  - the OS can decide when to serve these requests
Device States of Hard Disks

- Hard disks support several modes with low power consumption
- Drawback of low-power modes: access delays
- Modes of operation of an IBM Travelstar:

<table>
<thead>
<tr>
<th>Mode</th>
<th>Properties</th>
<th>Power consumption</th>
<th>Access delay</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active</td>
<td>read and write operations</td>
<td>2.1–4.7 W</td>
<td>—</td>
</tr>
<tr>
<td>Idle</td>
<td>entered after I/O operation</td>
<td>1.85 W</td>
<td>—</td>
</tr>
<tr>
<td>Low-Power Idle</td>
<td>heads on parking ramp</td>
<td>0.7 W</td>
<td>300 ms</td>
</tr>
<tr>
<td>Standby</td>
<td>spindle motor off</td>
<td>0.25 W</td>
<td>1.0–9.5 s</td>
</tr>
<tr>
<td>Sleep</td>
<td>(almost) all electronics off</td>
<td>0.1 W</td>
<td>3.0–9.5 s</td>
</tr>
</tbody>
</table>
Mode Transitions

- Mode transitions consume time and energy:
  - parking and positioning of heads
  - spindle motor activation and slow down

- Definition of the break-even time:
  
  \[
  \text{energy consumption in standby mode} + \text{transition to standby mode and back} = \text{energy consumption in idle mode}
  \]

Travelstar: break-even time = 8.7 s
Mode Transitions (2)

- No energy savings in standby mode if idle period is too short (< break-even time)
- The OS does not know whether a mode transition will save energy
- Traditional approach of “spin-down policies”:
  - keep track of disk operations
  - try to predict the time of future I/O operations according to the access pattern
Outline

- A New I/O Semantics
- Power Management of Hard Disks
- Cooperative I/O
- Implementation
- Measurements
- Conclusion
Cooperative I/O

**Deferrable and abortable I/O requests**

```c
read_coop(int fd, void *buf, size_t count,
         int time-out, int abort);
write_coop(int fd, void *buf, size_t count,
           int time-out, int abort);
open_coop(const char *pathname, int flags,
          int time-out, int abort);
```

- Hard disk in active or idle mode:
  - deferrable operations are executed immediately

- Hard disk in standby mode:
  - operations are deferred until hard disk is activated by another process
  - or until user-defined time-out is reached
  - then: force activation of hard disk or cancel the operation
Cooperative I/O (2)

Disk operations are clustered/grouped together

- generate long periods of inactivity
- fewer mode transitions
- hard disk can be kept longer in standby mode

Examples:
- audio-/video player
- web browser
- background processes
- auto-save
Outline

- A New I/O Semantics
- Power Management of Hard Disks
- Cooperative I/O
- Implementation
- Measurements
- Conclusion
Implementation

- Integration of all layers—from the hardware to the application

Implementation

- Linux Kernel 2.4.19
- Modifications to the IDE device driver, VFS (block buffer cache and update mechanism) and ext2 file system
Implementation (2)

1. Transition to standby mode if hard disk is idle
   - simple, efficient and proven algorithm:
     Device-Dependent Time-out [Lu, Micheli 2001 (Stanford)]
   - spin down if:
     current time - time of last access > break-even time
2. Energy-aware caching & update

- goal: clustering of disk operations
- update writes all “dirty” blocks to disk, independent of their age
- updates are attached to other disk accesses
- If device driver decides to switch to standby mode, force update before the mode transition
Outline

■ A New I/O Semantics
■ Power Management of Hard Disks
■ Cooperative I/O
■ Implementation
■ Measurements
■ Conclusion
Measurements

- **DAQ system**
  - measurement of voltage drop at defined resistors in the 5V supply line of the hard disk
  - resolution: 256 steps; 20000 samples per second

- **MP3 player AMP using deferrable `read_coop()` requests**

- **Player with two read buffers**
  - audio data is read from one buffer
  - thread fills other buffer with deferrable read calls.
  - modifications: ~ 150 lines

- **Mail reader stores new mails in file on hard disk using `write()`**
MP3-Player + Mail-Reader

- Write requests of mail reader and read requests of AMP are grouped together

unmodified AMP using `read()` (373 J)

mail reader using `write()` (164 J)

Cooperative-I/O kernel,
AMP using `read_coop()` + mail reader (227 J)
MP3-Player + Mail-Reader

- Linux kernel w/o power management (373 J):

- Cooperative-I/O kernel, AMP using `read()` (265 J):

- Cooperative-I/O kernel, AMP using `read_coop()` (210 J):
Two `read_coop()` requests to fill the two read buffers

If device is in standby mode:
- first request is deferred until active buffer is empty
- force spin-up to serve request; device is in idle mode
  ➜ `read_coop()` request for second buffer is executed immediately
  ➜ effectively two read operations are grouped together

Write requests are attached to read operations of AMP
Synthetic Tests

- Five processes wake up periodically and issue cooperative read or write requests after random wait times
- Comparison of four policies:
  - Linux kernel without power management
  - Cooperative-I/O kernel, test programs use `read()` or `write()`
  - Cooperative-I/O kernel, use `read_coop()` or `write_coop()`
  - Oracle
- “Oracle” spin-down policy
  - knows timing of future disk operations (traces!)
  - transition to standby mode immediately if energy savings are possible
  - optimal strategy with respect to energy consumption
  - no influence on times of disk operations!
Synthetic Tests: \texttt{read\_coop()}

- Higher energy savings than (uncooperative) oracle policy
- Cooperative I/O clusters accesses
  - \textit{\smaller reduction of mode transitions}
  - \textit{\smaller more time in standby mode}
- Oracle policy does not defer or cluster requests!
Synthetic Tests: `write_coop()`

- `write()` and `write_coop()` requests are already deferred by the update mechanism
  - only little additional savings when using `write_coop()`
- Oracle has no influence on the times of disk operations
  - requests are not deferred (synchronous writes)
Varying number of cooperative processes

- 5 processes run in parallel, 0–5 of them using `read_coop()`, the other `read()`

- The more processes using `read_coop()` instead of `read()`, the higher the energy savings
Conclusion

- Higher energy savings than oracle policy
  ➔ higher energy savings than any “traditional” spin-down policy
- Applicable to all types of devices with rotating media
- Starting & stopping the spindle motor causes wear:
  ◆ common reason for device failures
  ◆ 50,000–300,000 mode transitions max.