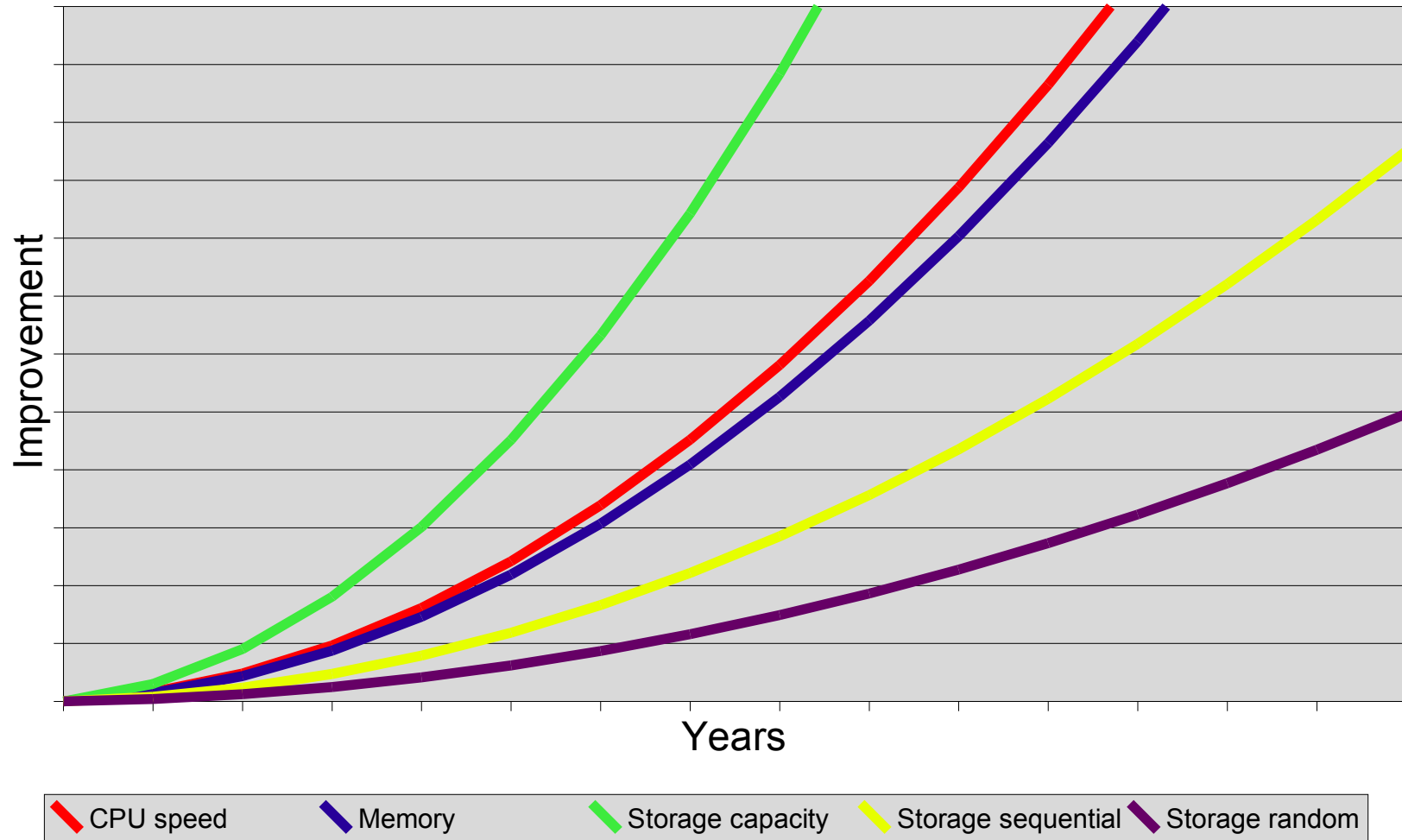


# Using prioritized I/O to improve storage bandwidth in MySQL

Christoffer Hall-Frederiksen  
and Philippe Bonnet

VLDB 2005

# Hardware trends



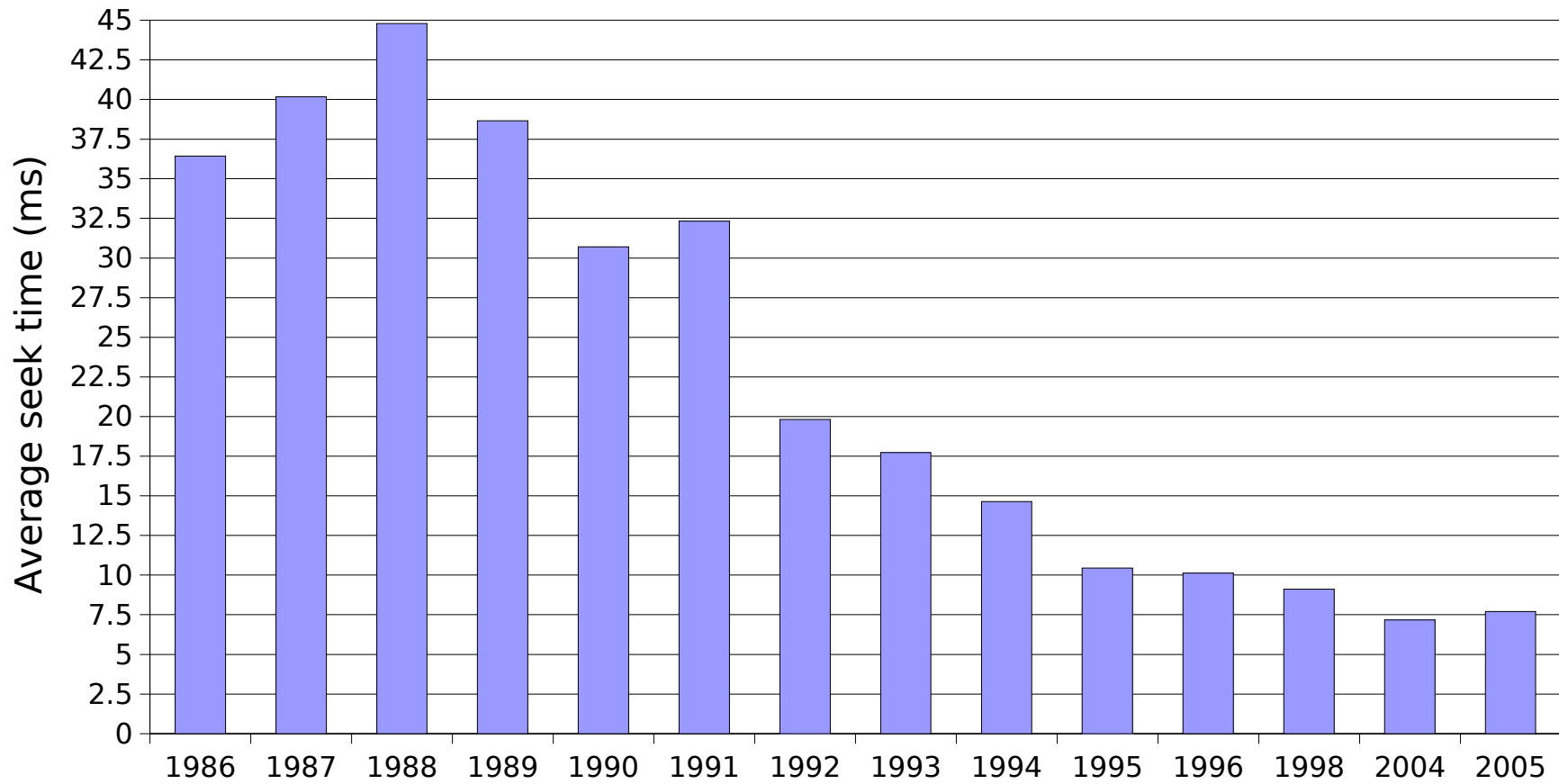
# Slow random I/O, why?

- A random I/O requires a seek
  - The disk arm needs to reposition
  - The platters need to spin to the right sector
- It's a mechanical movement
  - Less improvement made each year



Picture from  
[www.howstuffworks.com](http://www.howstuffworks.com)

# Seek times 1986-2005



# Mind the gap!

- Database workloads can often be disk bound
  - Utilizing the available storage bandwidth is key
- Databases produce random I/O
  - Access through non-clustered index
  - Flushing of pages from buffer pool
- Random I/O will increasingly dominate
  - Amdahl's law.
- Mind the access gap!!

# Random I/O is important!

## What can be done?

- Common answers are:
  - Increase device parallelism
  - Overlap I/O with computation
  - Reorganize data to exploit locality
  - More efficient scheduling of access
    - Requires many pending I/Os

# Paper idea

- Submit more I/Os
- Control latency with prioritized I/O
- Achieve better bandwidth utilization

# Storage utilization in databases

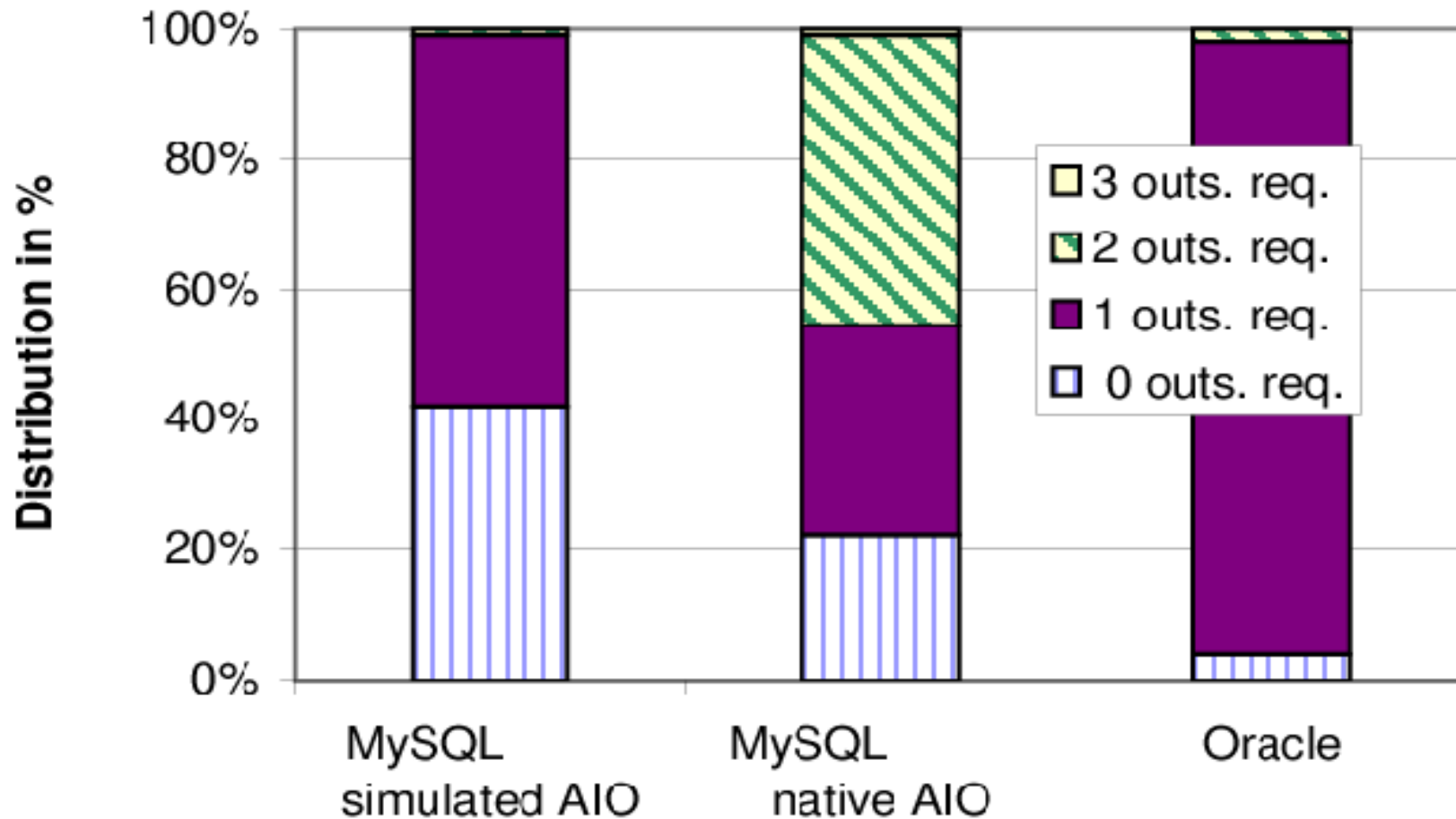
- How well do databases utilize storage bandwidth?
- Tuning experiments inspired this work
  - They showed a low number of pending I/Os

# Sequential access

- Table scans produce a sequential I/O pattern
- How well does Oracle and MySQL/InnoDB perform sequential access?
  - Oracle did a reasonably good job
    - Pending I/Os more than 90% of the time
  - MySQL didn't do as well
    - Pending I/Os less than 80% of the time

Setup: Dual P3, Adaptec controller,  
3 SCSI disks, Linux 2.6, Oracle 9, MySQL 4.1

# Sequential performance



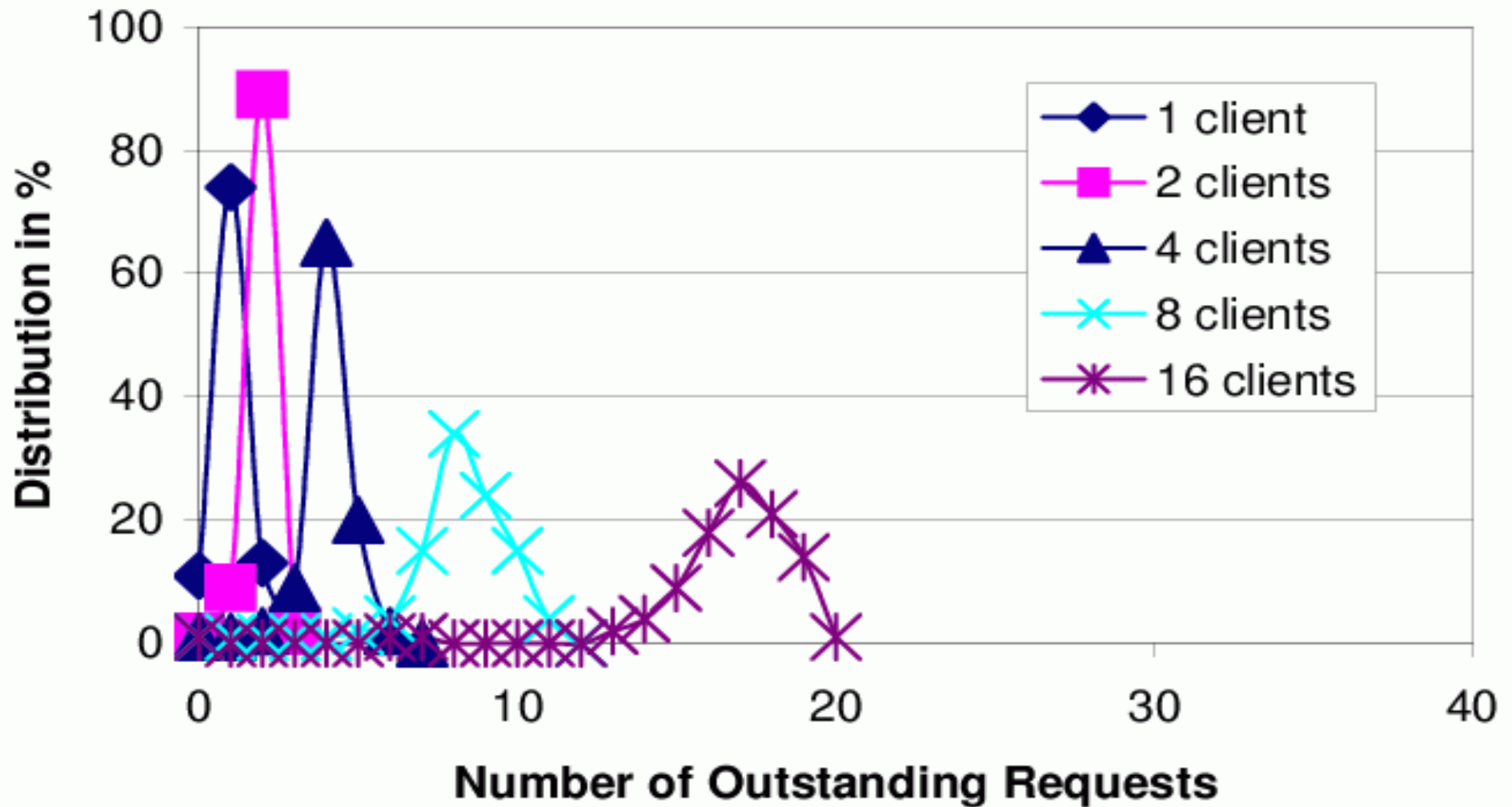
Data placed on a single IBM SCSI disk

# Random access

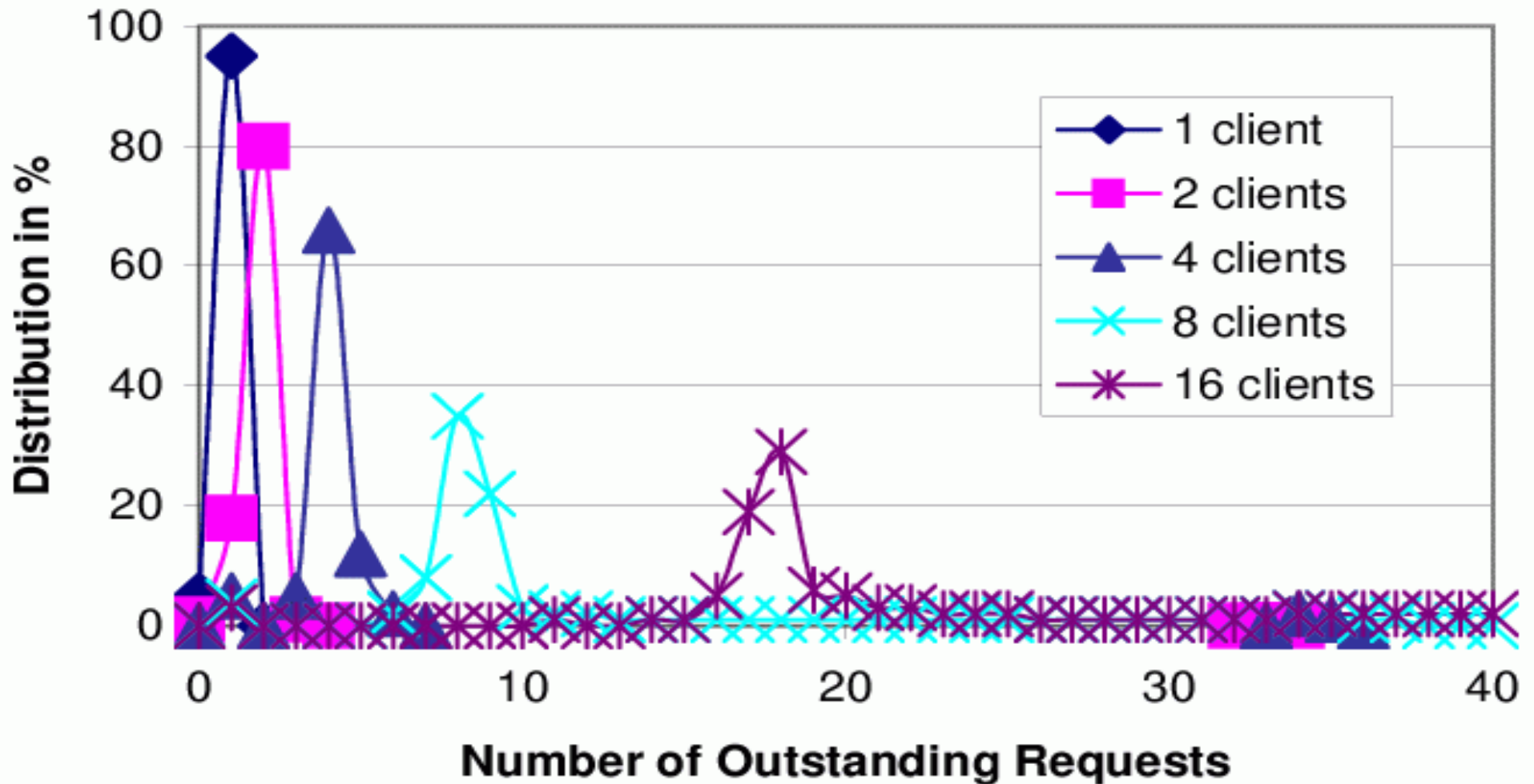
- How well does Oracle and MySQL/InnoDB perform random access?
  - Neither Oracle or MySQL did a good job

```
select avg(id2) from t where id between n and k  
(index on id, no index on id2)
```

# Random access in Oracle



# Random access in MySQL

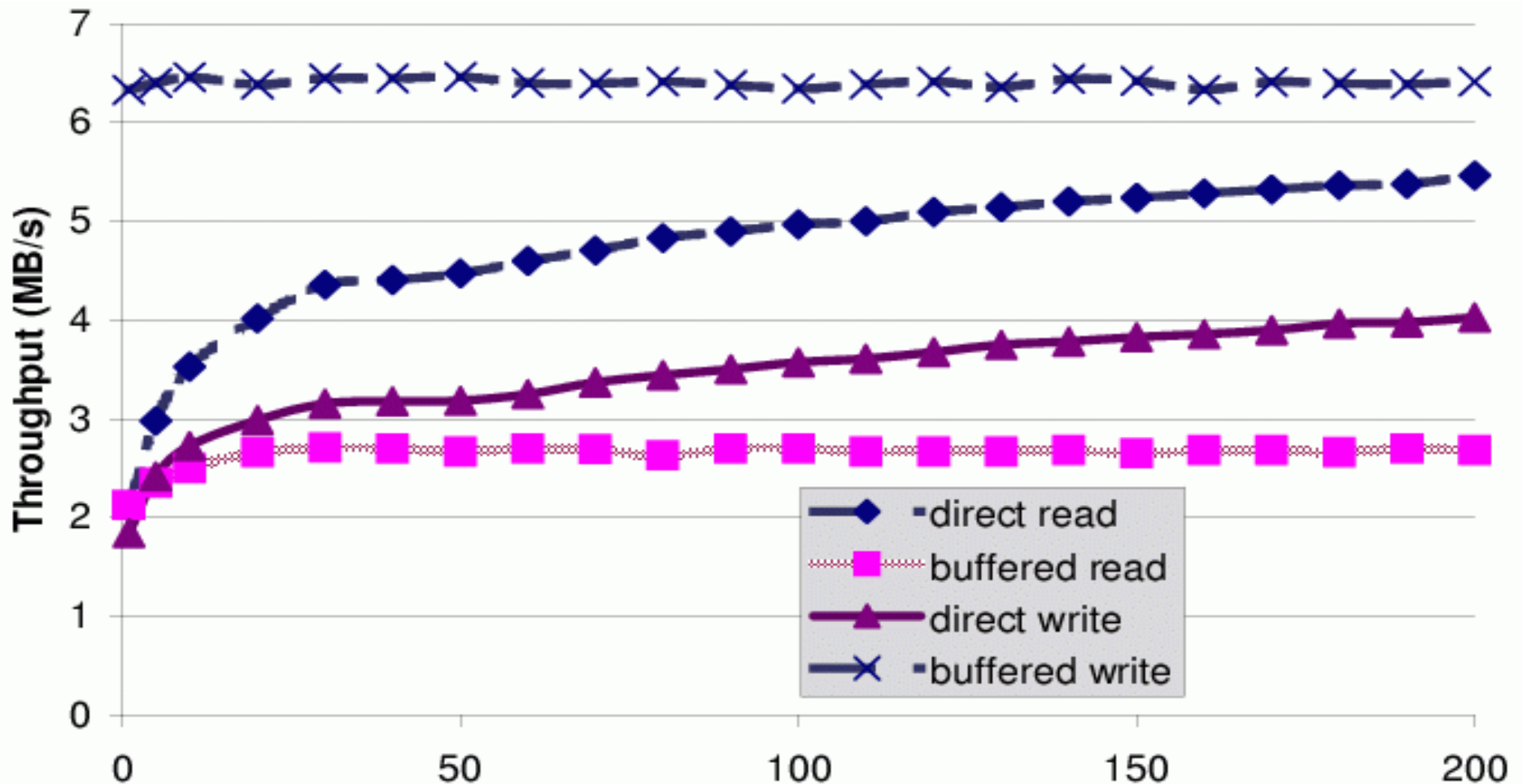


# Random access

## Random I/O summary

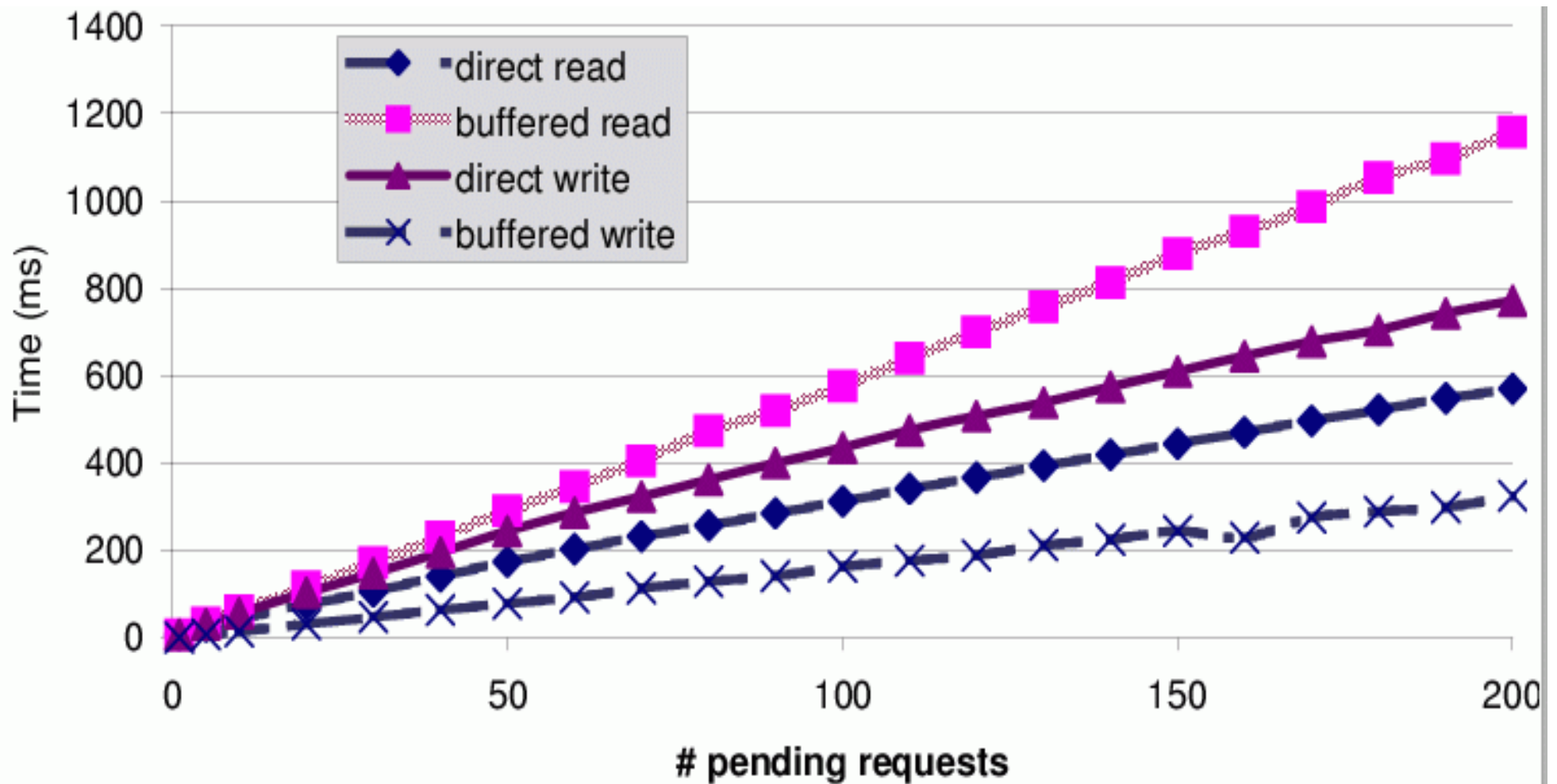
- Both Oracle and MySQL keeps few pending I/Os
- Disk scheduler cannot help
  - Few pending I/Os ties the disk schedulers hands
- More I/Os should be submitted!

# What could be gained?



Synthetic benchmark submitting async I/O

# What is the price?



# Why are databases so conservative?

- There are two classes of database I/O
  - Transaction synchronous
  - Transaction asynchronous
- Low latency for synchronous I/O is important
- Conservative policy
  - The only latency control available is to keep the number of pending I/Os low.

# Contribution

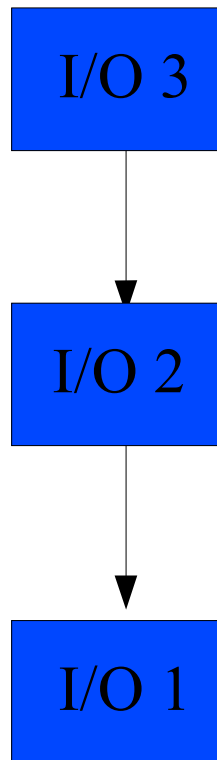
- Support for native async I/O in InnoDB on Linux
  - Will be in MySQL-5.1!
- A prioritizing disk scheduler for Linux 2.6
  - Submission of prioritized asynchronous I/Os
- Modified InnoDB lazy writer
  - Uses an aggressive I/O policy
  - Controls latency through prioritized I/O

# Linux disk schedulers

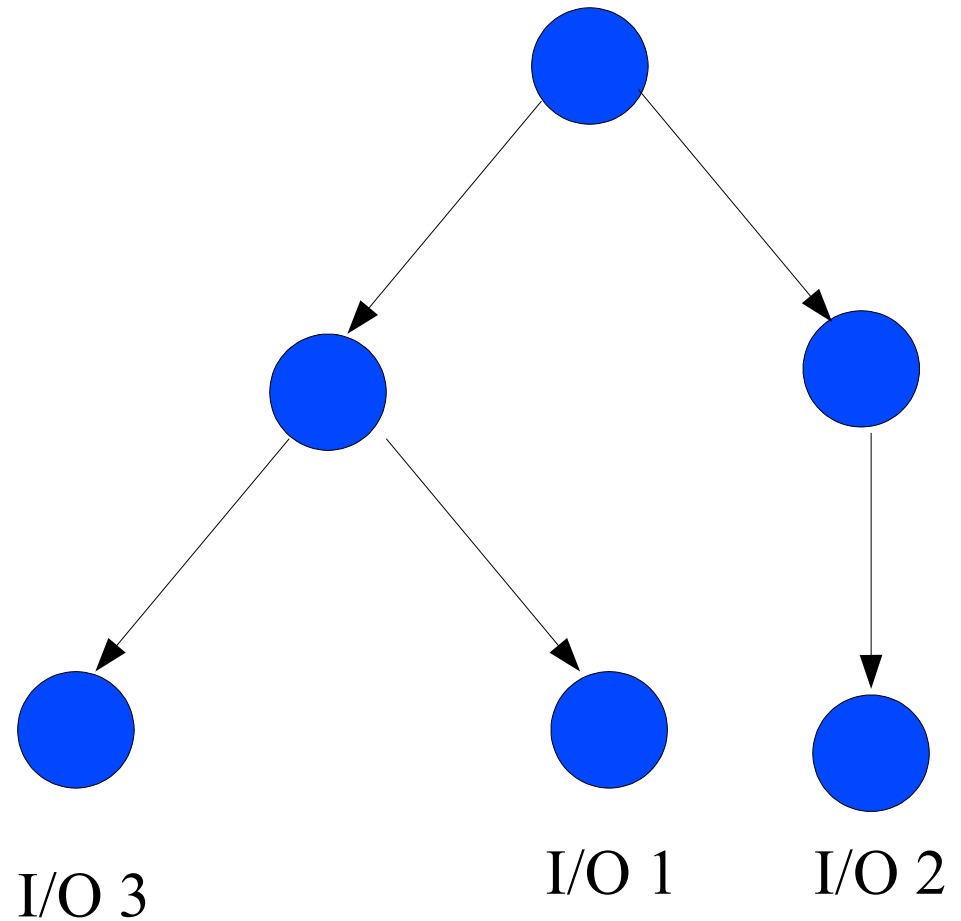
- Linux has 4 different disk schedulers
  - Deadline, anticipatory, CFQ and NOOP
- The deadline scheduler is by far the most well suited for databases
- Prioritization was introduced in deadline !

# The deadline scheduler

FIFO



Sector sorted



# Introducing priorities

- High priority I/O should not wait
- Low priority I/O can wait a bit
- Priority determines when the deadline is set
- More FIFOs are now used
  - One per priority level

# Current InnoDB lazy writer

- Slowly flushes pages during idle time
- Loops and flushes while
  - Less than 10% of pages are immediately replaceable
  - The database is not idle

# Modified InnoDB lazy writer

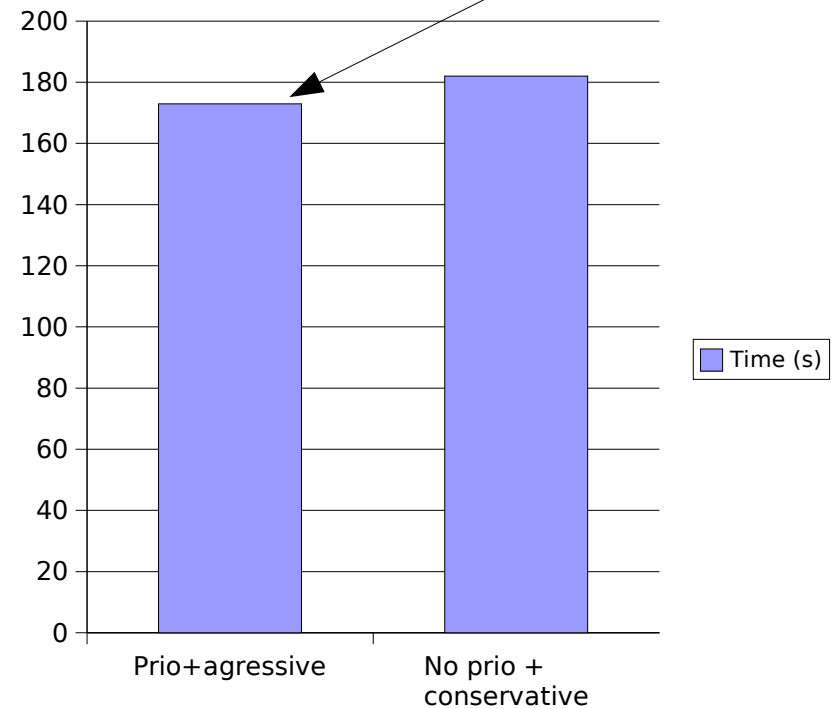
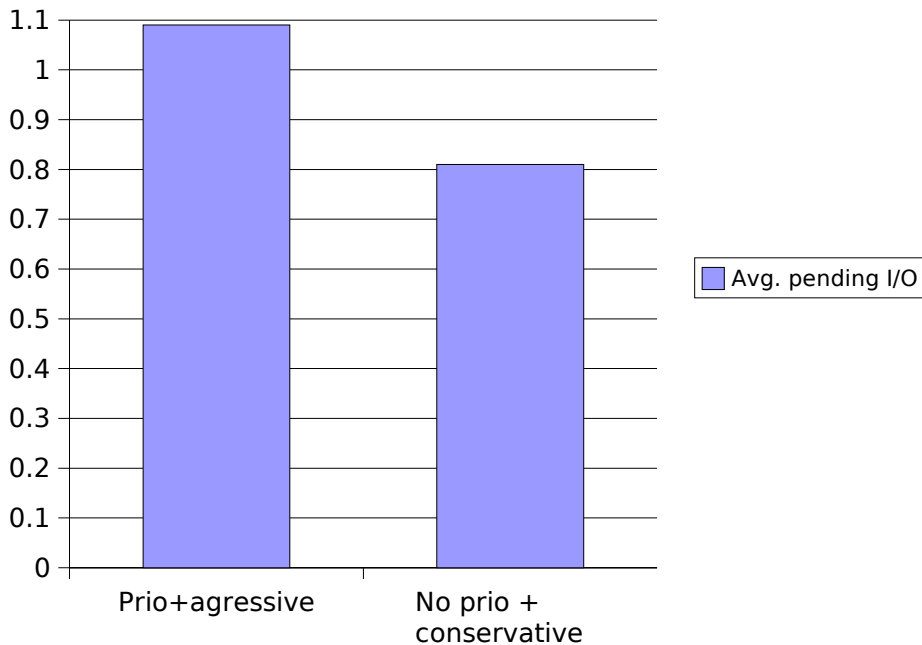
- Kicks in earlier and submits more
- The additional I/Os are submitted with low priority
- Submits I/Os depending on memory pressure

# Results

- Simple queries were used to evaluate performance
- Both updating and non-updating queries
- Storage utilization is measured as well as overall performance

# Example query

- Simple update used to create memory pressure
  - update T set number = number + 1



# Non-updating queries

- Queries doing scan and index traversal were not affected
- But the aggressive I/O-policy has a price
  - The lazy writer kicks in earlier
- On workloads where the working set just fits into memory and is updated continually, performance will suffer.

# Conclusion

- More I/Os are submitted
  - Annotated with priorities
- Disk scheduler prioritizes I/O
  - In accordance with application request
- A step towards an aggressive I/O policy

# Future work

- Prioritized I/O on SAN-sized storage
  - SCSI-3 and Fiber Channel has priorities
- Metadata for handling storage caches
- Prioritized aggressive read ahead
- Adaptivity
  - Instead of fixed sized I/O submissions, aiming for a high number of pending I/Os at all times
- Larger benchmarks
  - TPC-C, TPC-H etc.

# Questions?

- More info on <http://www.distlab.dk/badger>