

Anticipatory scheduling: a disk scheduling framework to overcome deceptive idleness in synchronous I/O

Proceedings of the 18th ACM symposium on Operating systems principles, 2001

Anticipatory Disk Scheduling

Sitaram Iyer

Peter Druschel

Rice University

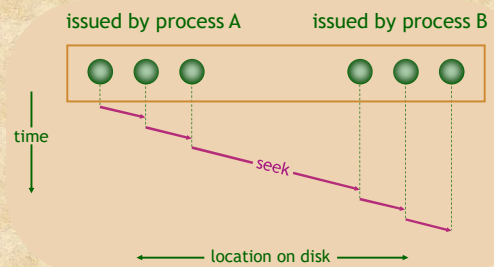
Disk schedulers

Reorder available disk requests for

- performance by seek optimization,
- proportional resource allocation, etc.

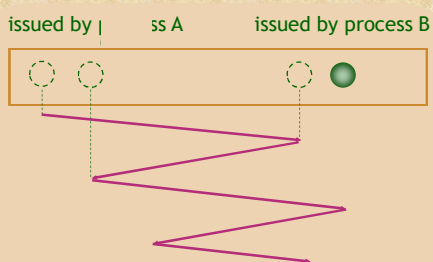
Any policy needs multiple outstanding requests to make good decisions!

With enough requests...



E.g., Throughput = 21 MB/s (IBM Deskstar disk)

With synchronous I/O...



E.g., Throughput = 5 MB/s

Deceptive idleness

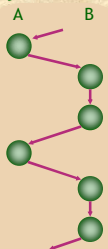
Process A is about to issue next request.

but

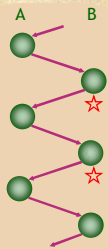
Scheduler hastily assumes that process A has no further requests!

Proportional scheduler

Allocate disk service
in say 1:2 ratio:



Deceptive idleness
causes 1:1 allocation:



Prefetch

Overlaps computation with I/O.

Side-effect:
avoids deceptive idleness!

- Application-driven
- Kernel-driven

Prefetch

- Application driven - e.g. `aio_read()`

aio

- `aio_read()` Start an asynchronous read operation
- `aio_write()` Start an asynchronous write operation
- `lio_listio()` Start a list of asynchronous I/O operations
- `aio_suspend()` Wait for completion of one or more asynchronous I/O operations
- `aio_error()` Retrieve the error status of an asynchronous I/O operation
- `aio_return()` Retrieve the return status of an asynchronous I/O operation and free any associated system resources
- `aio_cancel()` Request cancellation of a pending asynchronous I/O operation
- `aio_fsync()` Request synchronization of the media image of a file to which asynchronous operations have been addressed

Aio usage patterns

Blocking

```
aio_read()
aio_read()
aio_read()
aio_read()
aio_read()
aio_read()
aio_suspend()
```

Polling

```
aio_read()
aio_read()
aio_read()
aio_read()
aio_read()
aio_read()
do {
    aio_error()
} until (completed)
```

Aio usage patterns

Signals

```
aio_read()
aio_read()
aio_read()
aio_read()
aio_read()
aio_read()
.
other()
stuff()
.
```

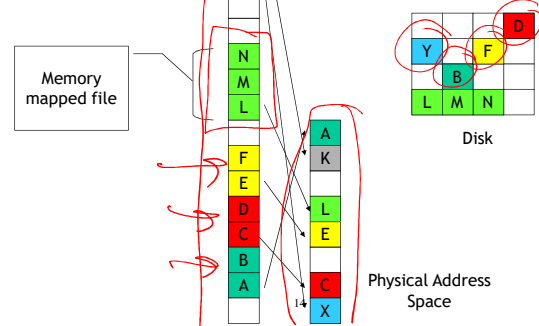
Signal handler

```
process_data()
```

Prefetch

- Application driven - e.g. aio_read()
 - Application need to know their future
 - Cumbersome programming model
 - Existing apps need re-writing
 - aio_read() optional
 - May be less efficient than mmap

Memory-mapped files and paging



Prefetch

- Kernel driven
 - Less capable of knowing the future
 - Access patterns difficult to predict, even with locality
 - Cost of misprediction can be high
 - Medium files too small to trigger sequential access detection



Anticipatory scheduling

Key idea: Sometimes wait for process whose request was last serviced.

Keeps disk idle for short intervals.

But with informed decisions, this:

- Improves throughput
- Achieves desired proportions

When, How, How Long

- When should we or shouldn't we delay disk requests?
- How long do we delay disk requests, if we do delay?
- How do we make an informed decision?
 - What metrics might be helpful?

Cost-benefit analysis

Balance expected benefits of waiting against cost of keeping disk idle.

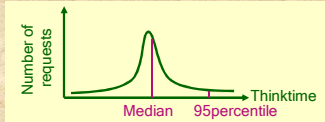
Tradeoffs sensitive to scheduling policy

- e.g.,
1. seek optimizing scheduler
 2. proportional scheduler

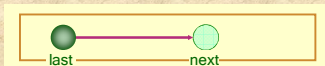
Statistics

For each process, measure:

1. Expected median and 95percentile thinktime



2. Expected positioning time



Cost-benefit analysis for seek optimizing scheduler

best := best available request chosen by scheduler

next := expected forthcoming request from process whose request was last serviced

Benefit =

best.positioning_time - next.positioning_time

Cost = next.median_thinktime

Waiting_duration =

(Benefit > Cost) ? next.95percentile_thinktime : 0

Proportional scheduler

Costs and benefits are different.

e.g., proportional scheduler:

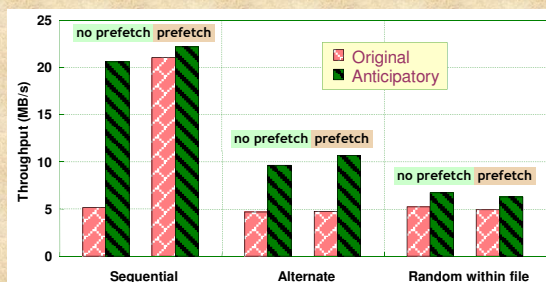
Wait for process whose request was last serviced,
 1. if it has received less than its allocation, and
 2. if it has thinktime below a threshold (e.g., 3ms)

Waiting_duration = next.95percentile_thinktime

Experiments

- FreeBSD-4.3 patch + kernel module (1500 lines of C code)
- 7200 rpm IDE disk (IBM Deskstar)
- Also in the paper: 15000 rpm SCSI disk (Seagate Cheetah)

Microbenchmark



Real workloads

What's the impact on real applications and benchmarks?

Andrew benchmark

Apache web server (large working set)

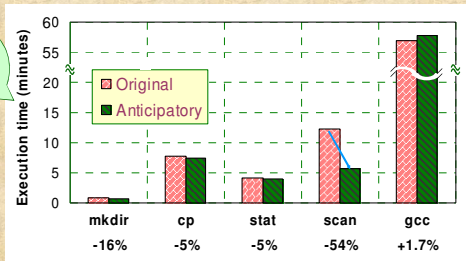
Database benchmark

- Disk-intensive
- Prefetching enabled

Andrew filesystem benchmark

2 (or more) concurrent clients

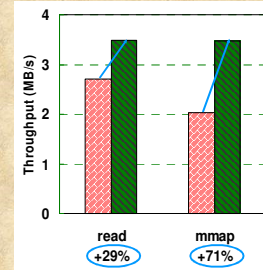
Lower is better



Overall 8% performance improvement

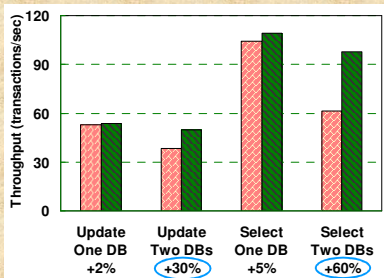
Apache web server

- CS.Berkeley trace
- Large working set
- 48 web clients



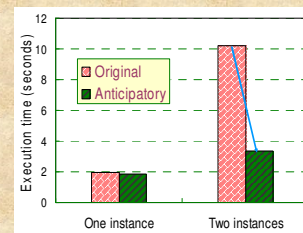
no prefetch

Database benchmark



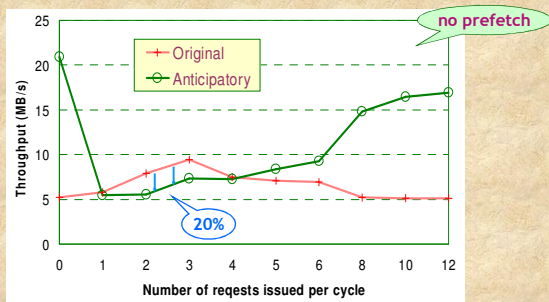
- MySQL DB
- Two clients
- One or two databases on same disk

GnuLD

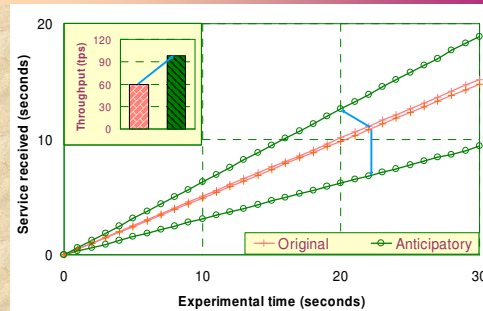


Concurrent: 68% execution time reduction

Intelligent adversary



Proportional scheduler



Database benchmark: two databases, select queries

Conclusion

Anticipatory scheduling:

- overcomes deceptive idleness
- achieves significant performance improvement on real applications
- achieves desired proportions
- and is easy to implement!



Anticipatory Disk Scheduling

Sitaram Iyer

Peter Druschel

<http://www.cs.rice.edu/~ssiyer/r/antsched/>