

# Anticipatory Disk Scheduling

Sitaram Iyer

Peter Druschel

Rice University

# Disk schedulers

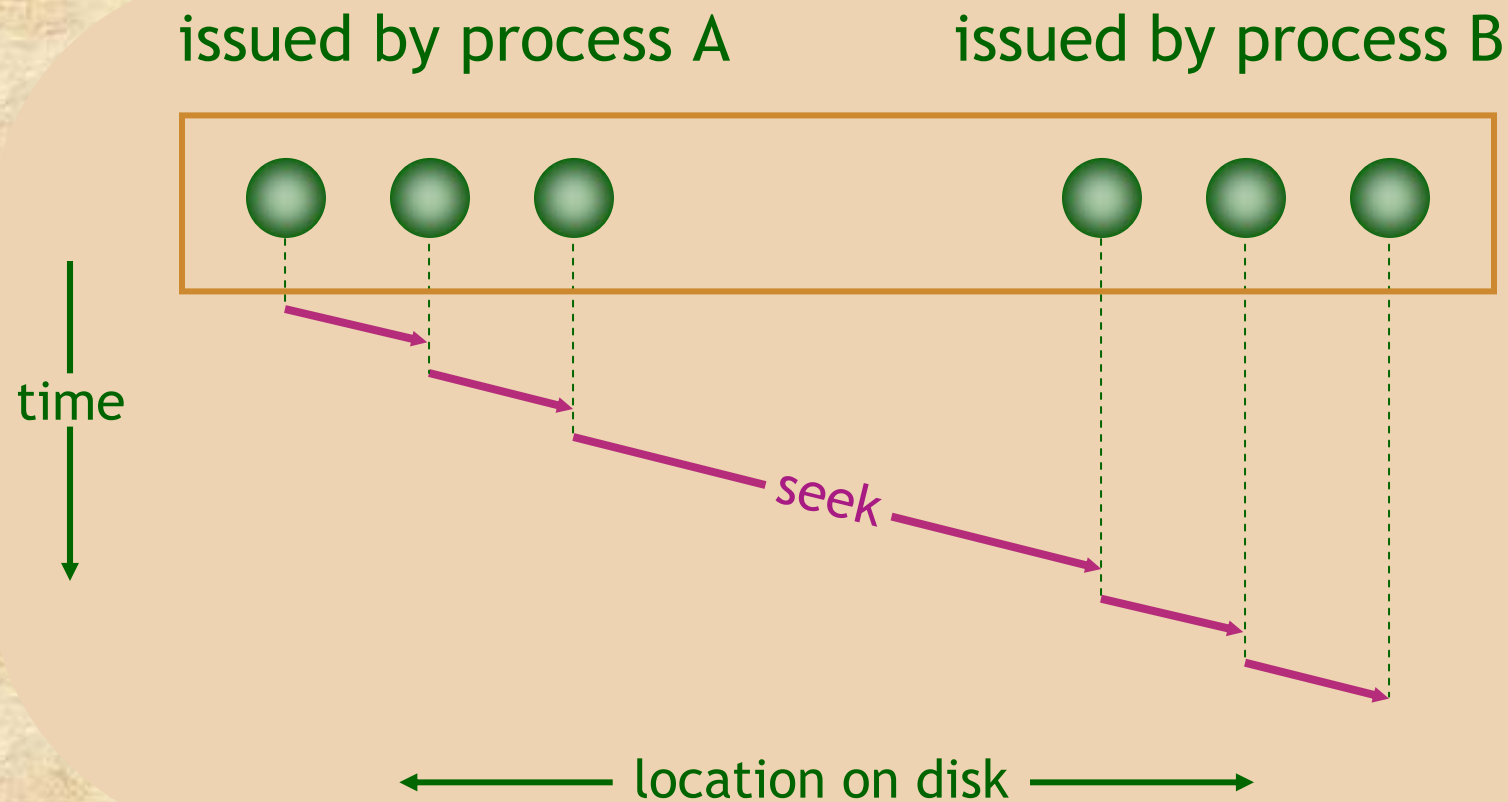
---

Reorder available disk requests for

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

Any policy needs multiple outstanding requests to make good decisions!

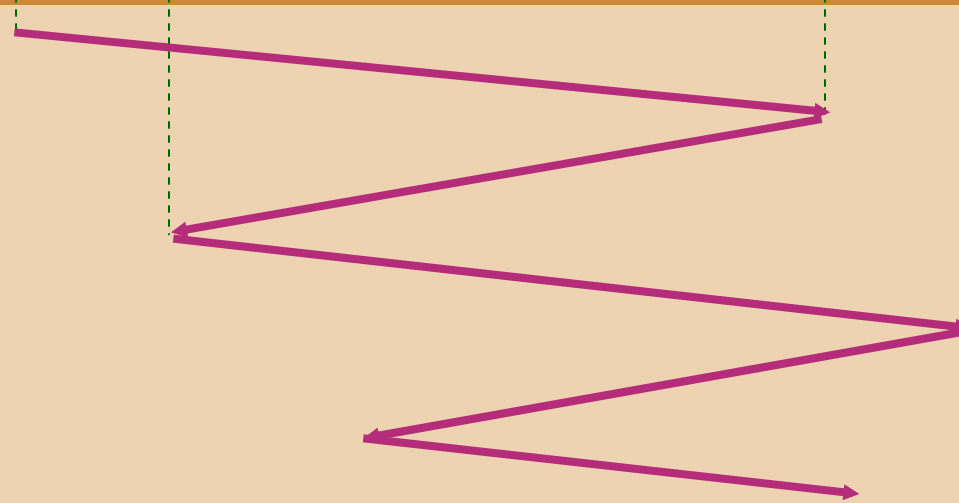
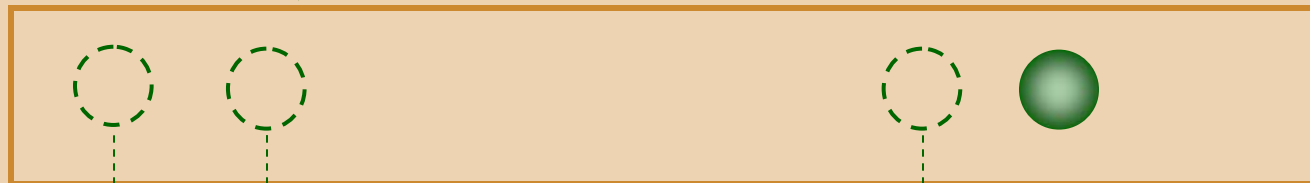
# With enough requests...



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

# With synchronous I/O...

issued by process A      issued by process B



E.g., Throughput = 5 MB/s

# Deceptive idleness

---

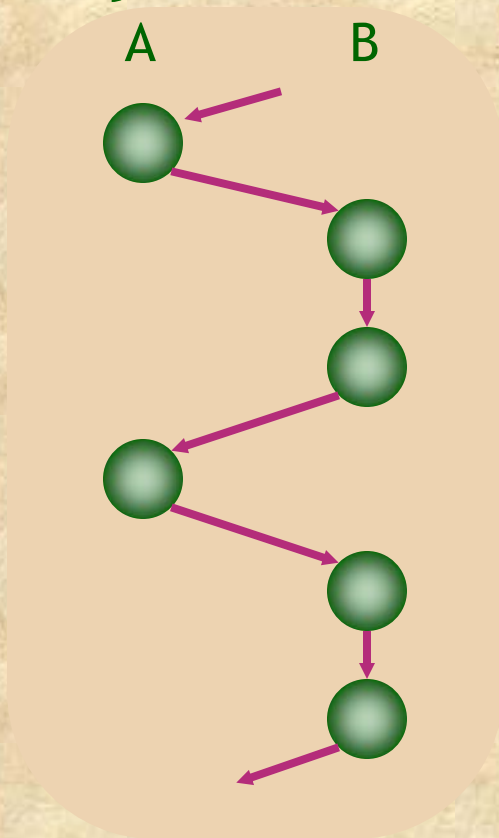
**Process A is about to issue next request.**

**but**

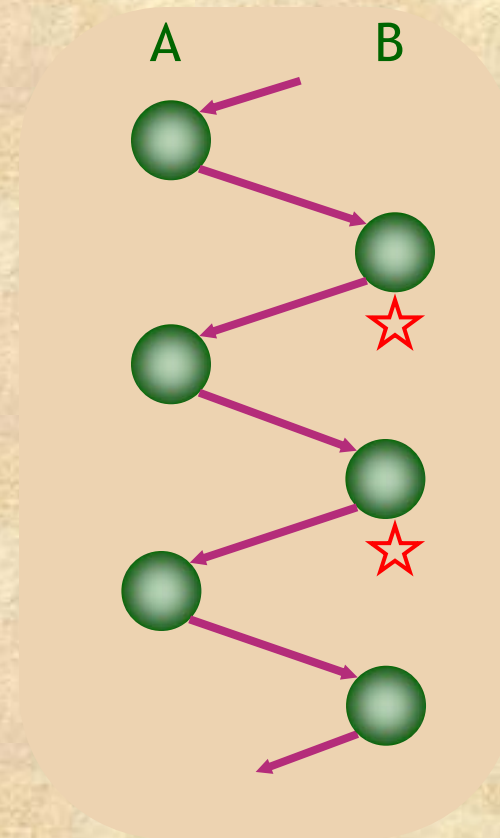
**Scheduler hastily assumes that process A  
has no further requests!**

# Proportional scheduler

Allocate disk service  
in say 1:2 ratio:



Deceptive idleness  
causes 1:1 allocation:





# 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

# Cost-benefit analysis

---

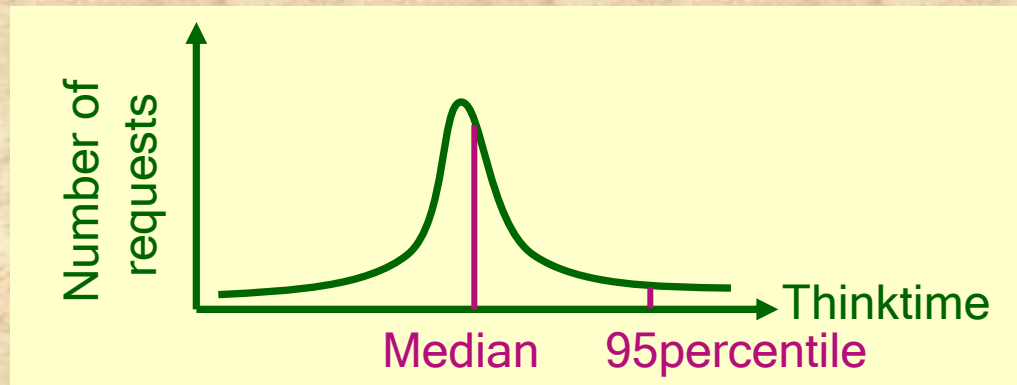
Balance expected benefits of waiting against cost of keeping disk idle.

Tradeoffs sensitive to scheduling policy  
e.g., 1. seek optimizing scheduler  
2. proportional scheduler

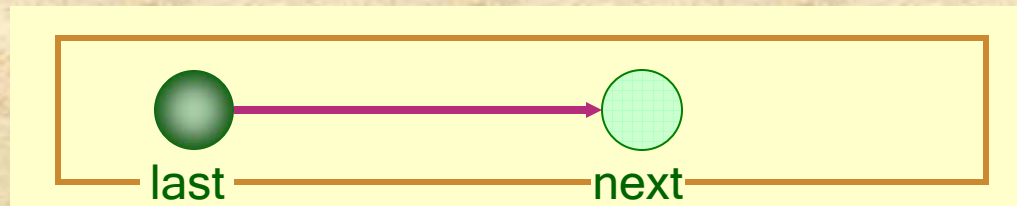
# Statistics

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**

# Prefetch

---

Overlaps computation with I/O.

Side-effect:

avoids deceptive idleness!

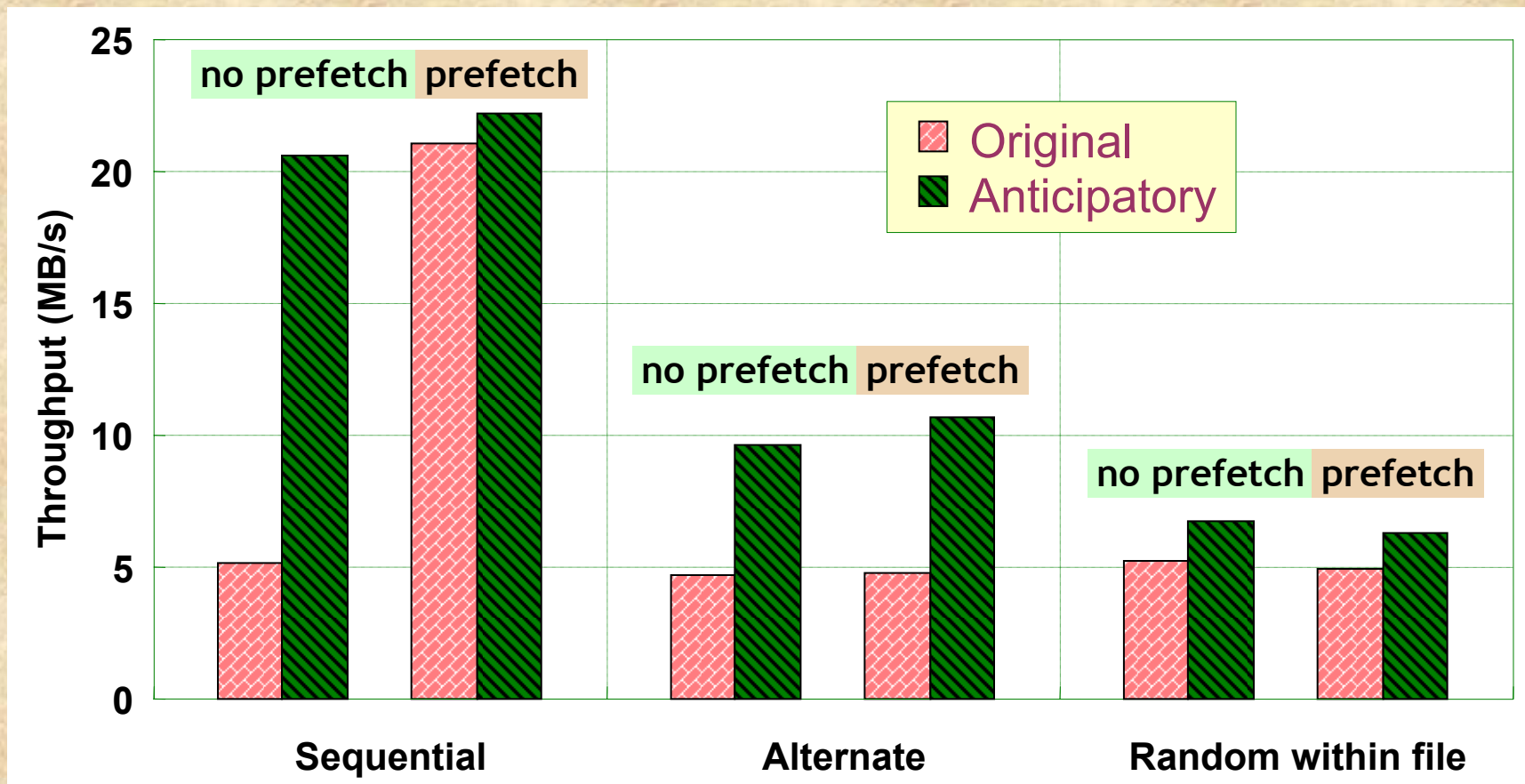
- Application-driven
- Kernel-driven

# 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)

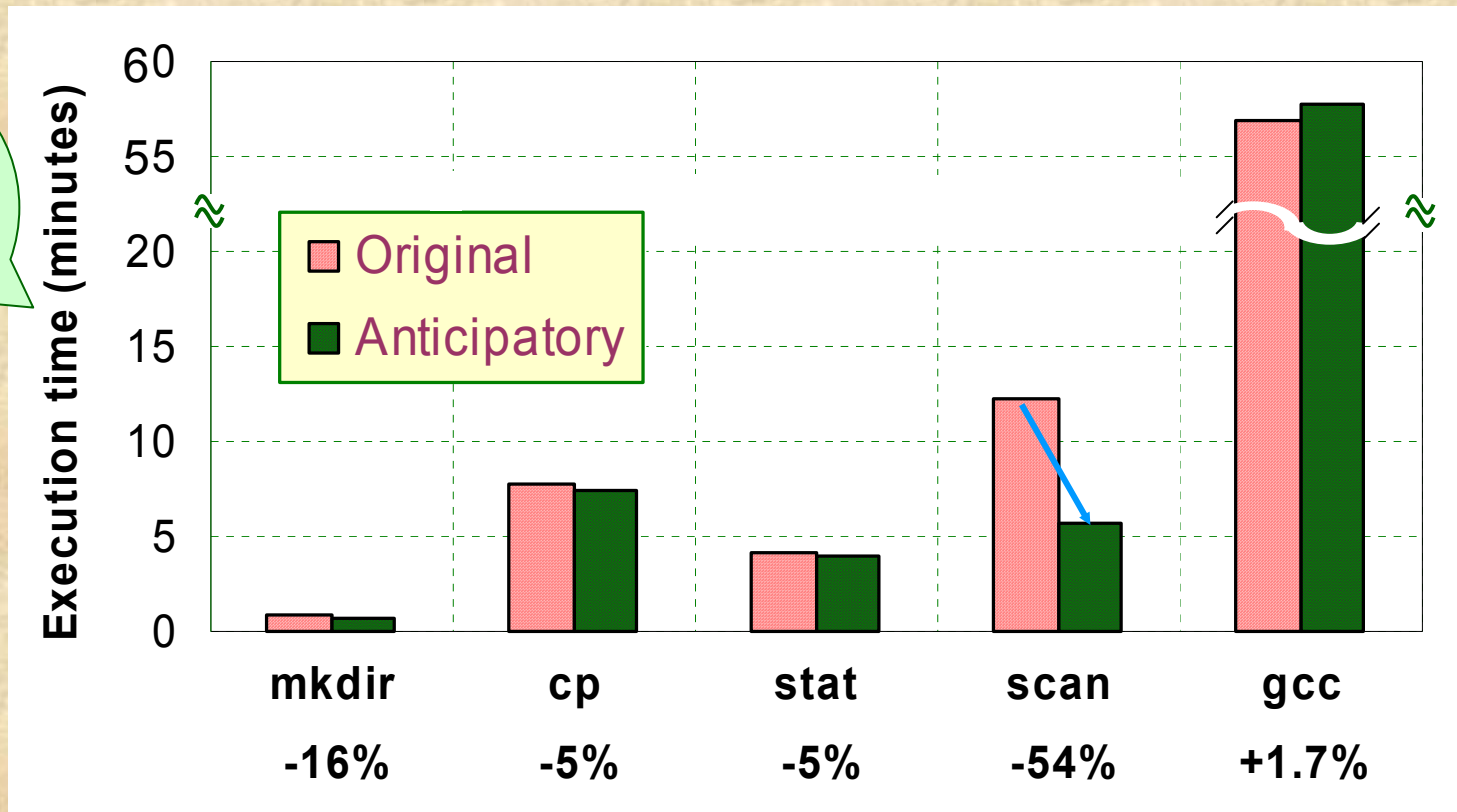
Database benchmark

- Disk-intensive
- Prefetching enabled

# Andrew filesystem benchmark

2 (or more) concurrent clients

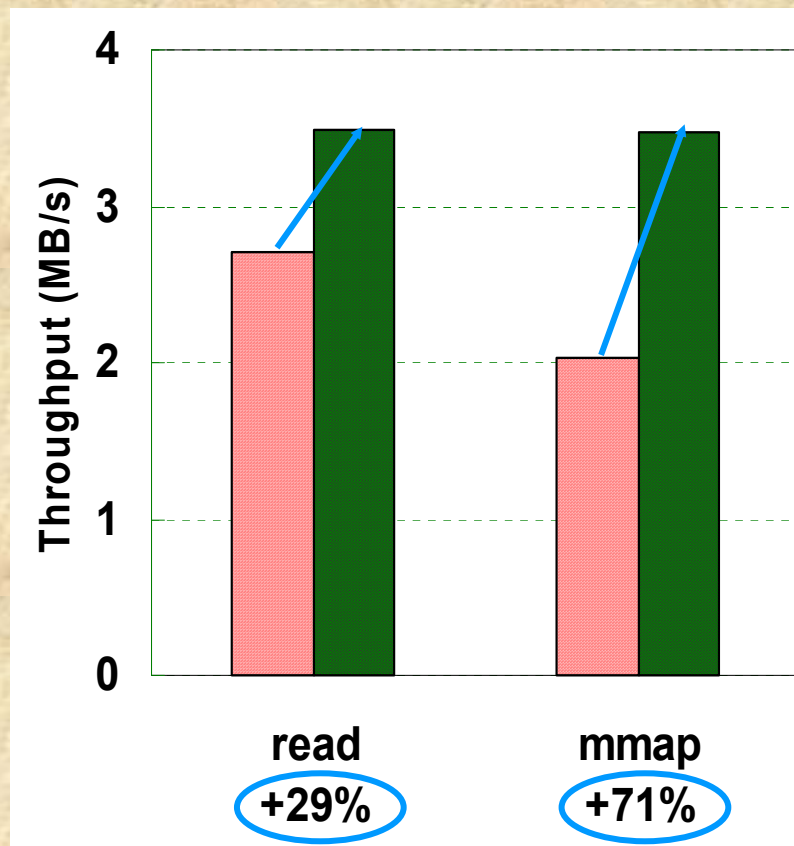
Lower is better



Overall 8% performance improvement

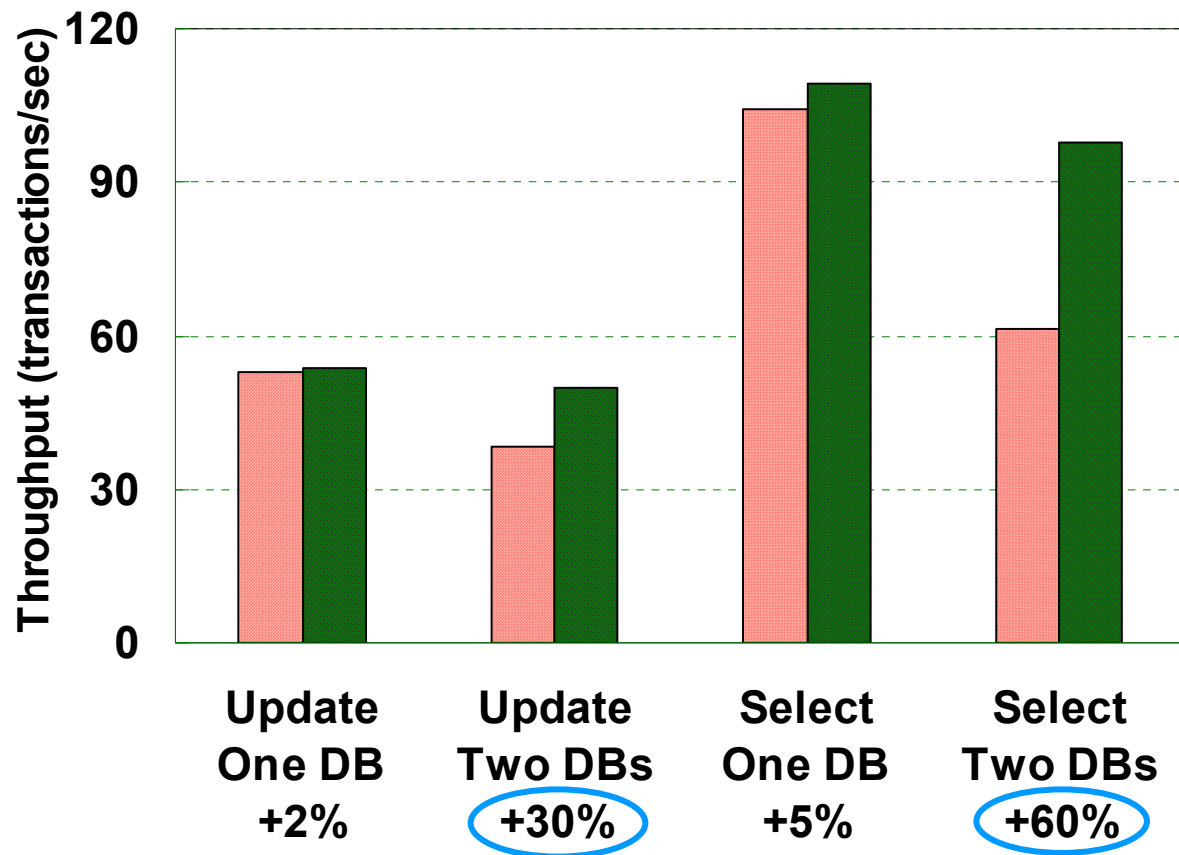
# Apache web server

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



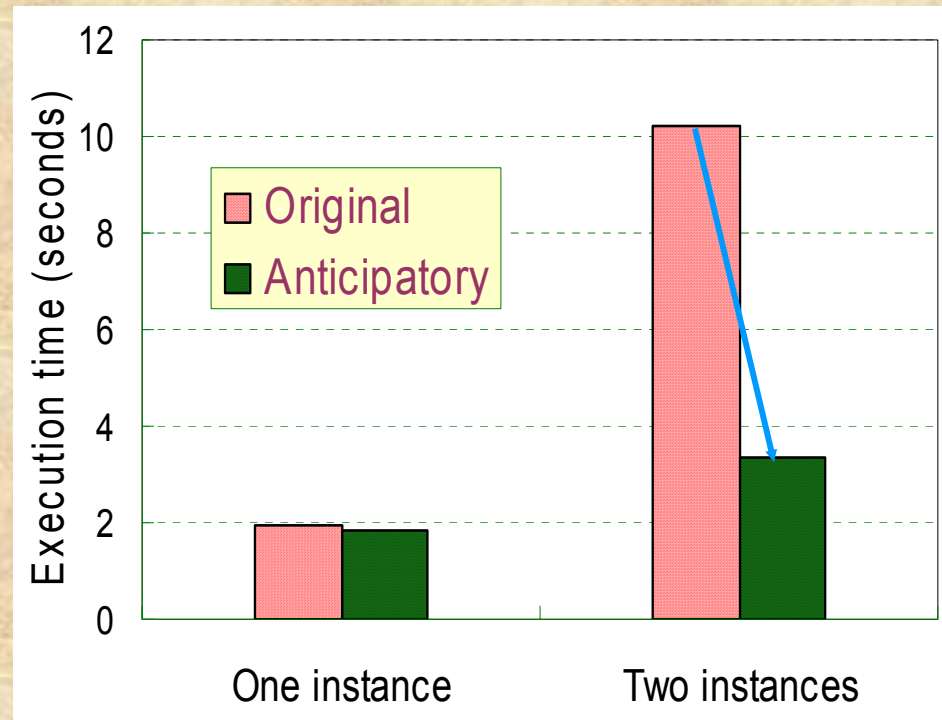
no prefetch

# Database benchmark



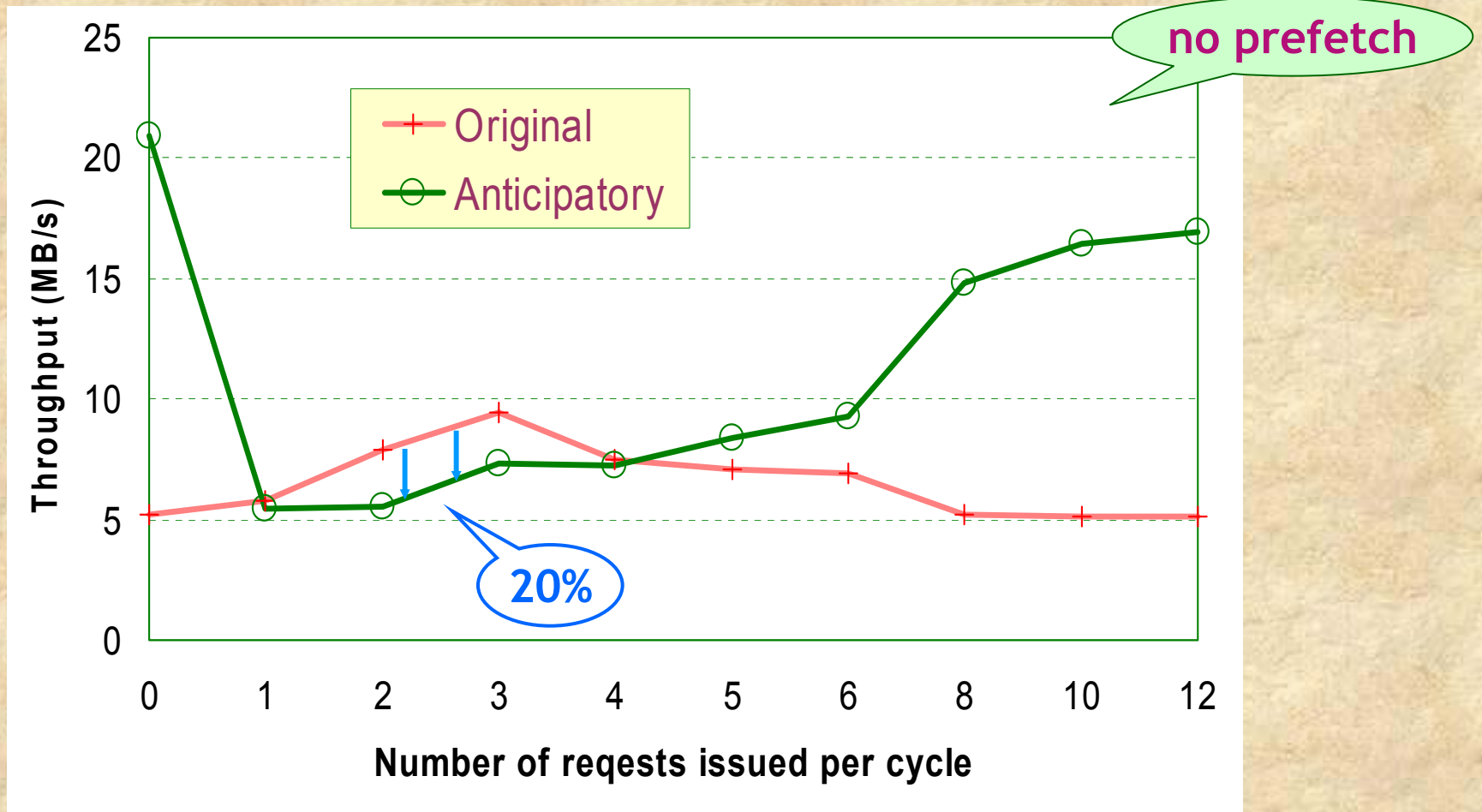
- MySQL DB
- Two clients
- One or two databases on same disk

# GnuLD

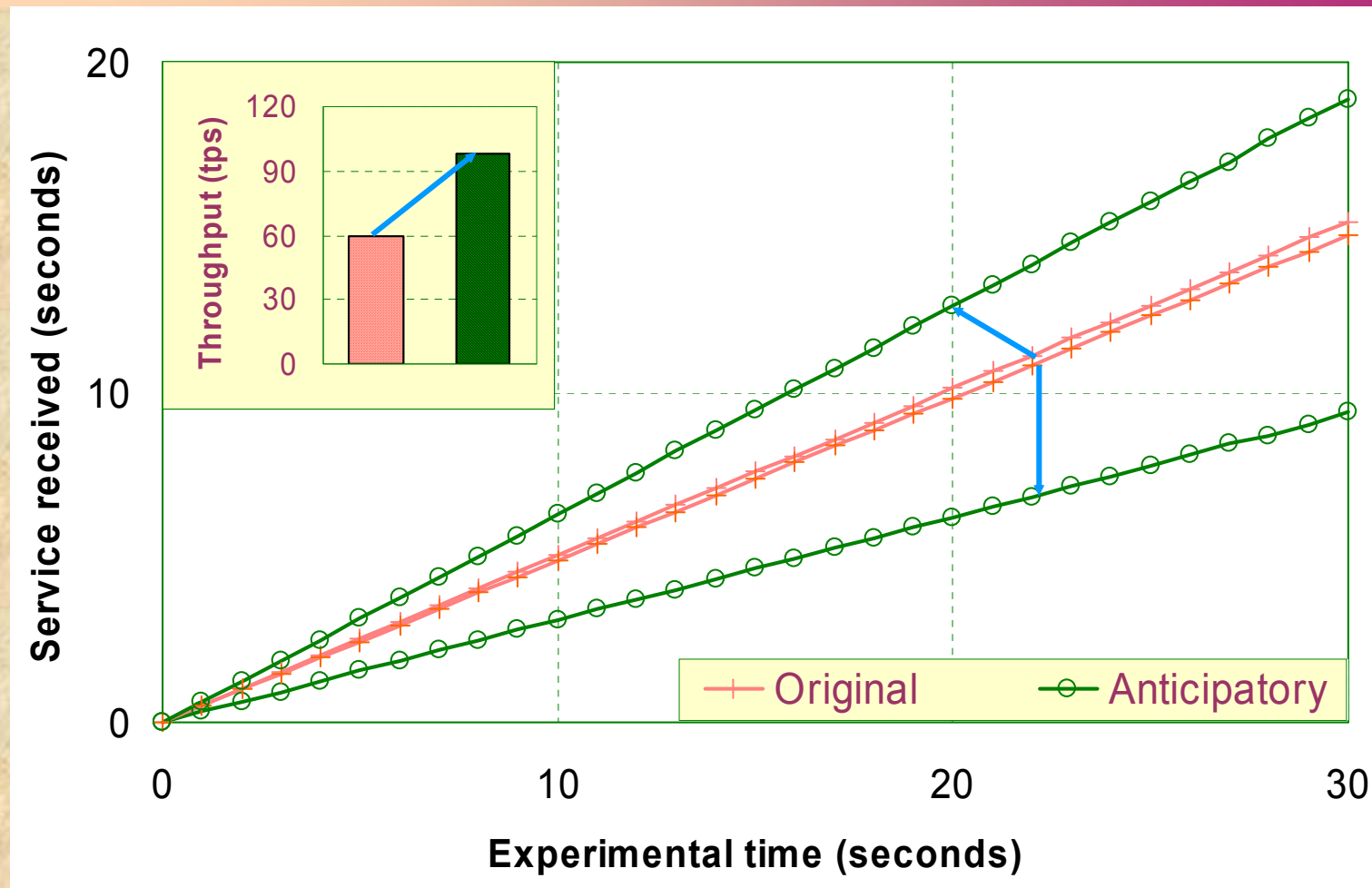


**Concurrent: 68% execution time reduction**

# 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/>