



Dynamic Adaptability in Support of Extreme Scale

Enhancements to Linux I/O Scheduling

Seetharami R. Seelam, UTEP

Rodrigo Romero, UTEP

Patricia J. Teller, UTEP


William Bueros, IBM-Austin



Introduction

Dynamic Adaptability in Support of Extreme Scale

- Linux 2.6 provides four I/O schedulers: Anticipatory (AS), deadline, completely fair queuing (CFQ), and noop
- Selection at
 - boot time: one scheduler for all drives
 - runtime: one scheduler per drive
- Default: AS

- **Expected admissible** response time for I/O requests
 - Streaming read in background (forgot to kill it)
 -  Timing Linux source tree read
 - HUGE RESPONSE TIME

Insight by Accident



Motivation-2

Dynamic Adaptability in Support of Extreme Scale

- Questions
 - Is AS the problem?
 - Does AS starve processes?
 - If so, can we extend AS?
 - Do the extensions work?
 - Do other schedulers give better response times?
 - If so, can the best scheduler be selected dynamically and automatically?
 - What metrics can be used to guide selection?



Project Goal

Dynamic Adaptability in Support of Extreme Scale

Enhanced Performance

Generalized  Customized
resource management

Fixed  Dynamically Adaptable
OS/runtime services



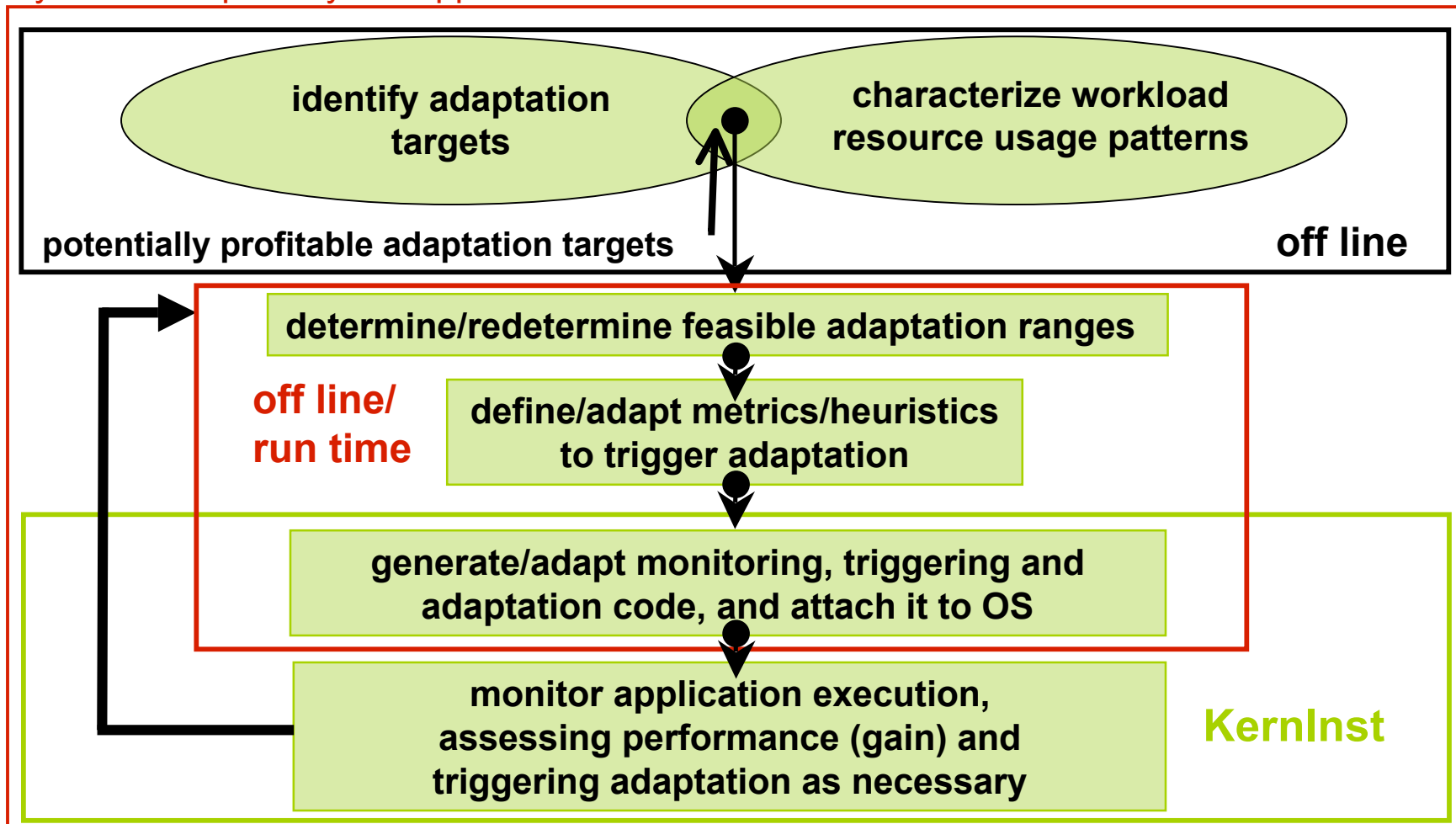
Project Challenges

Dynamic Adaptability in Support of Extreme Scale

Determining

- What to adapt
- When to adapt
- How to adapt
- How to measure effects of adaptation

Dynamic Adaptability in Support of Extreme Scale





Outline

Dynamic Adaptability in Support of Extreme Scale

- I/O Schedulers in Linux
- Problems with Anticipatory Scheduler
- Cooperative Anticipatory Scheduler
- Performance Evaluation
- I/O Characterization for Dynamic & Automatic Scheduler Selection
- Questions for me and for you



Introduction

Dynamic Adaptability in Support of Extreme Scale

- Linux provides four I/O schedulers:
 - anticipatory scheduler (AS)
 - deadline
 - completely fair queuing (CFQ)
 - noop



Deadline Scheduler

Dynamic Adaptability in Support of Extreme Scale

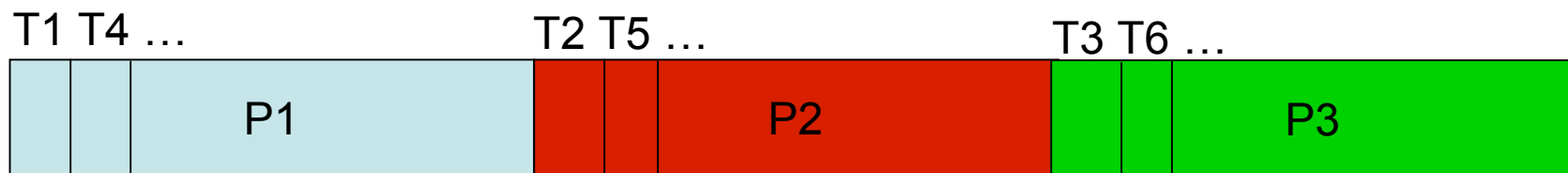
- Work conserving
- Idea:
 - Requests are queued: sorted by block number and fifo
 - At request completion:
 - schedule expired requests from fifo queue
 - schedule requests from sorted queue
 - In between schedule some write requests



Deadline Scheduler Deceptive Idleness

Dynamic Adaptability in Support of Extreme Scale

- Work-conserving nature forces head to move to next selected block
- Deceptive idleness reduces throughput
- Example: multiple synchronous requests generated by different processes to disjoint disk blocks





Linux Anticipatory Scheduler (LAS)

Dynamic Adaptability in Support of Extreme Scale

- Non work conserving
- Goal: seek reduction
- Idea:
 - Per-process anticipation: wait for requests to nearby blocks; periodically evaluate anticipation period
 - Keep head idle during anticipation
 - Balance seek time and anticipation time
- Anticipation improves performance only if it is correct and anticipation time $<$ seek time



When Not to Anticipate

Dynamic Adaptability in Support of Extreme Scale

- Anticipated processes keep dying
 - What if the requests are to nearby blocks from a group of processes?
- Process just started I/O
- Process requests large seeks



LAS Problems

Dynamic Adaptability in Support of Extreme Scale

- Inadmissible turnaround time
 - Two processes: one a good candidate for anticipation, the other beats anticipation
 - Example: Stream read and chunk read, each chunk by a different process
- Poor throughput: deceptive idleness due to anticipation failure
 - Both processes beat anticipation
 - Example: two chunk reads



Cooperative Anticipatory Scheduler (CAS)

Dynamic Adaptability in Support of Extreme Scale

- Detect cooperative processes and anticipate accordingly
- Idea:
 - Per-process anticipation
 - Process group anticipation: if a process just starting I/O belongs to a group, start anticipation
 - Processes requesting nearby blocks belong to a group: one or more can be dead -- still the group exists



CAS

Solution to AS Problems

Dynamic Adaptability in Support of Extreme Scale

- Admissible turnaround time
 - Stream read and chunk read, combination beats anticipation; chunk reads are identified as **a group**
- Poor throughput: deceptive idleness due to anticipation failure
 - Both processes beat anticipation; two chunk reads – **two groups !!!!**
- Does CAS really work?
- Results on an array of application profiles with different I/O characteristics – web server, mail server, file server, meta data operations



Experimental Evaluation

Dynamic Adaptability in Support of Extreme Scale

- Does CAS really work?
- Results on few microbenchmarks
 - Streaming writes and chunk reads
 - Streaming reads and chunk reads
 - Chunk reads
- Results on a set of application profiles with different I/O characteristics – web server, mail server, file server, meta data operations



Experimental Evaluation Platform

Dynamic Adaptability in Support of Extreme Scale

- Dual processor Pentium 4 Xeon – single processor is used
- 1GB memory and 1MB L2 cache
- 2.6.9 Linux Kernel
- 7,200 RPM Maxtor 20 GB IDE disk – separate from OS drive
- Ext3 file system; similar results for xfs file system



Experimental Evaluation Workload

Dynamic Adaptability in Support of Extreme Scale

- Microbenchmarks that defeat anticipation
- Flexible File System Benchmark (FFSB) workload generator
 - Profiles simulating web server, mail server, file server and meta data operations
 - Each profile creates 100,000 files; each file ranges in size from 4 KB to 64KB
 - Four concurrent threads makes 80,000 operations
 - All operations are random
 - Capture time for 80,000 operations



Experimental Evaluation Metrics

Dynamic Adaptability in Support of Extreme Scale

- **Execution Time:** User perspective
- **Throughput:** System architect perspective



Experimental Evaluation Streaming Reads & Writes

Dynamic Adaptability in Support of Extreme Scale

- Mixed workload: “important” reads, “not so important” writes
- Is LAS better than deadline?
- Deliberately delay asynchronous writes

Program 1:

```
while true
do
    dd if=/dev/zero of=file \
        count=2048 bs=1M
done
```

Program 2:

```
time cat 200mb-file > /dev/null
```

Scheduler	Execution Time (sec.)	Throughput (MB/s)
Deadline	129	25
LAS	10	33
CAS	9	33

Table 1: Performance of Programs 1 and 2 under the Deadline Scheduler, LAS, and CAS

- LAS and CAS provide better response times
- Deadline alternates serving reads and writes (several times) hence seeks; eliminated in LAS and CAS
- Thus better MB/s



Experimental Evaluation Streaming & Chunk Reads

Dynamic Adaptability in Support of Extreme Scale

- A: all requests from single process
- B: every file read by different process
- Anticipation works well for A, but what happens with B?

Scheduler	Execution Time (sec.)	Throughput (MB/s)
Deadline	297	9
LAS	4767	35
CAS	255	34

Table 2: Performance of Program A and B under the Deadline Scheduler, LAS, and CAS

Program A:

```
while true
do
    cat big-file > /dev/null
done
```

Program B:

```
time find . -type f -exec \
    cat '{}' ';' > /dev/null
```

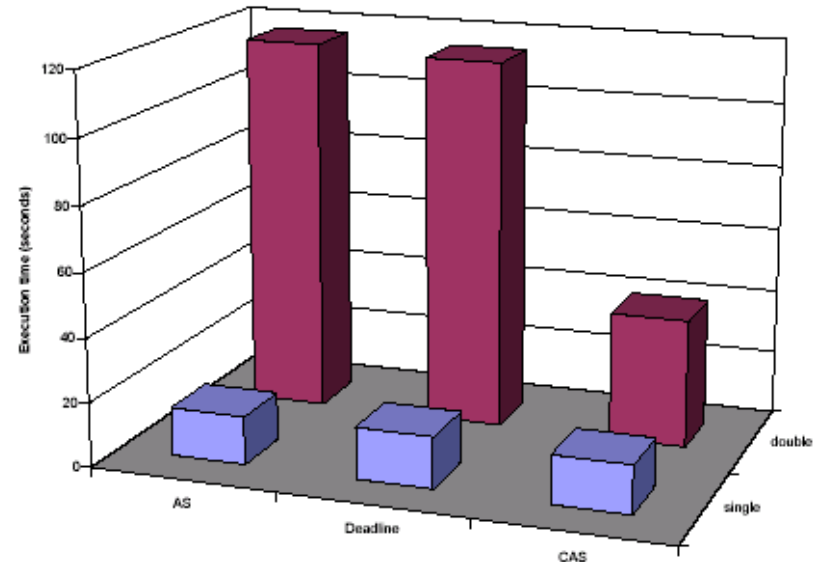
- B: inadmissible time using LAS
- Deadline has too many seeks
- CAS provides anticipation on a per-group basis; thus seeks reduced and throughput improved



Experimental Evaluation Multiple Chunk Reads

Dynamic Adaptability in Support of Extreme Scale

- Illustrates reduced disk throughput problem
- Two instances of chunk reads to disjoint disk blocks
- Anticipation fails for both
- Results for reading Linux source tree



Scheduler	Throughput (MB/s)	
	1 Instance	2 Instances
Deadline	14.5	4.0
LAS	15.5	4.0
CAS	15.5	11.6

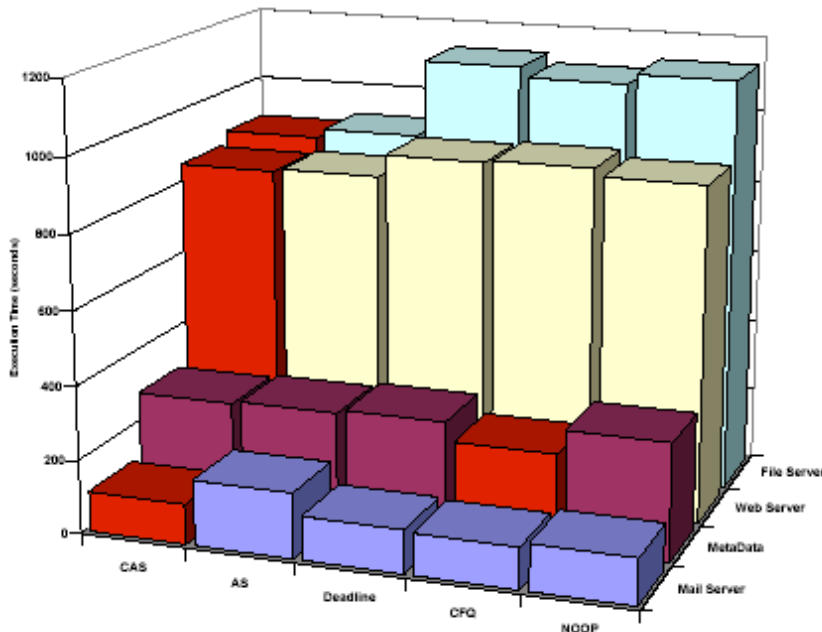
- Deadline and LAS have horrible throughput
- There is some seeking, but CAS does not seek as much as others



Experimental Evaluation Web Server Profile

Dynamic Adaptability in Support of Extreme Scale

- Read requests to randomly selected files
- Simulates a web server



Scheduler	Web Server	Mail Server	File Server	Meta Data
Deadline	924	118	1127	305
LAS	863	177	916	295
CAS	855	109	890	288
CFQ	931	112	1099	253
noop	910	125	1127	319

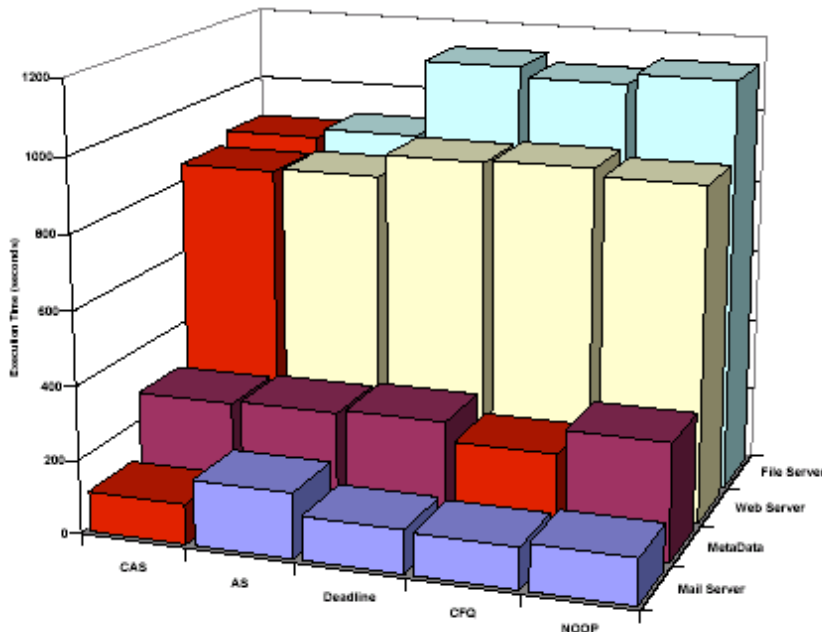
- There is very little anticipation – may be on 8 KB - 64 KB files
- LAS has execution time comparable to CAS
- Deadline, CFQ, and noop trail CAS by 8%, 8.9%, and 6.5% respectively
- Deviation less than 4%



Experimental Evaluation Mail Server Profile

Dynamic Adaptability in Support of Extreme Scale

- 40% reads, 40% file creates and 20% file delete operations
- Operations are on random files



Scheduler	Web Server	Mail Server	File Server	Meta Data
Deadline	924	118	1127	305
LAS	863	177	916	295
CAS	855	109	890	288
CFQ	931	112	1099	253
noop	910	125	1127	319

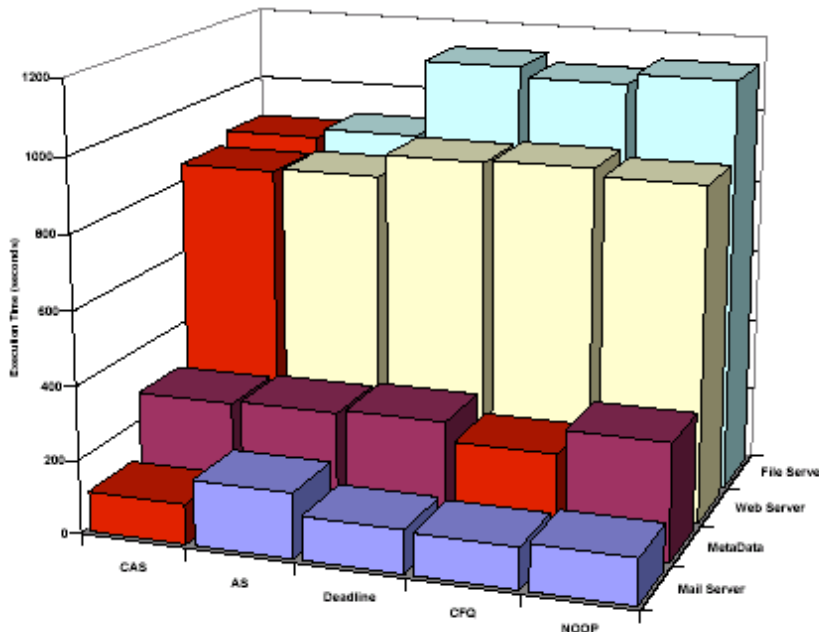
- Deviation is less than 3.5% except for LAS which has 11%
- CAS has best execution time
- LAS has worst performance
- LAS, deadline, CFQ, and noop trail CAS by 62%, 8%, 3%, and 14%, respectively



Experimental Evaluation File Server Profile

Dynamic Adaptability in Support of Extreme Scale

- 80% reads, 20% writes
- Operations are on random files



Scheduler	Web Server	Mail Server	File Server	Meta Data
Deadline	924	118	1127	305
LAS	863	177	916	295
CAS	855	109	890	288
CFQ	931	112	1099	253
noop	910	125	1127	319

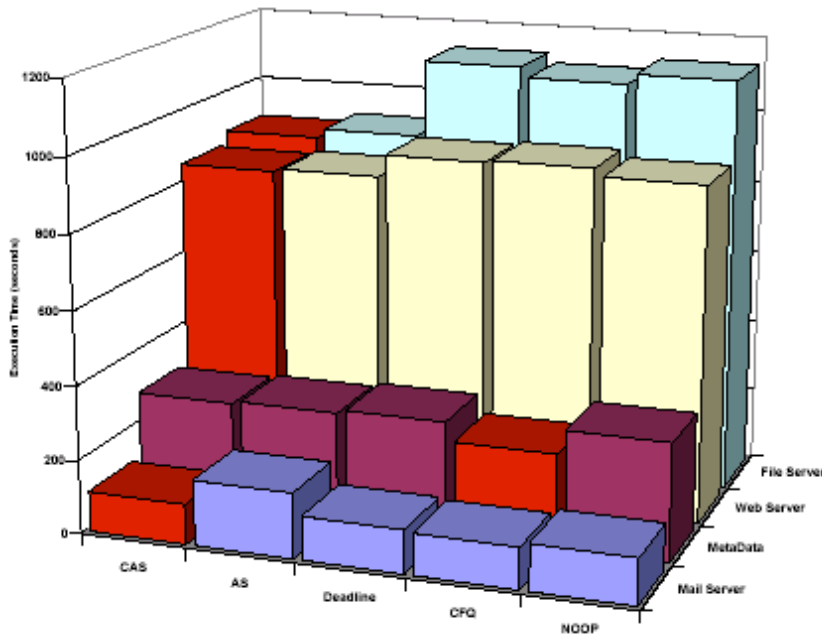
- Deviation is less than 4.5%
- CAS has best execution time
- LAS is very close – less than 3%
- Others trail CAS by at least 23%



Experimental Evaluation Meta Data Profile

Dynamic Adaptability in Support of Extreme Scale

- 40% create, 40% write - append, and 20% file delete operations



Scheduler	Web Server	Mail Server	File Server	Meta Data
Deadline	924	118	1127	305
LAS	863	177	916	295
CAS	855	109	890	288
CFQ	931	112	1099	253
noop	910	125	1127	319

- Maximum deviation is 7.7%
- CFQ has best execution time
- CAS, LAS, deadline, and noop trail CFQ by as much as 26%
- Similar results for xlf file system



Summary so far ...

Dynamic Adaptability in Support of Extreme Scale

- Identified an important performance problem with LAS and offered a solution
- Introduced the concept of cooperative processes and making scheduling decisions based on groups of processes
- Compared performance on a set of microbenchmarks and applications



Motivation

Dynamic Adaptability in Support of Extreme Scale

- Questions

- Is AS the problem? ✓
- Does AS starve processes? ✓
- If so, can we extend AS? ✓
 - Do the extensions work?
- Do other schedulers give better response times? ✓
- If so, can the best scheduler be selected dynamically and automatically?
 - What metrics can be used to guide selection?



Further Interesting Work

Dynamic Adaptability in Support of Extreme Scale

- **Dynamic I/O scheduler selection**
- Dynamic parameter tuning to maximize performance
- Inclusion of learning algorithms
- Perhaps, genetic and neural network combinations



I/O Scheduler Selection

Dynamic Adaptability in Support of Extreme Scale

- Vendors moved from AS to CFQ as the default scheduler
- Steven Pratt's [LINUX2004] paper: scheduler selection is a complicated issue at best
 - Summary: Scheduler selection is a function of
 - Workload, e.g., sequential, random, etc.,
 - File system, e.g., xfs, ext3, raiserfs, etc.,
 - Storage system, e.g., single drive, raid, etc.,
- Selection is difficult for:
 - Workloads with orthogonal requirements
 - Mixed workloads, e.g., file server and a web server on the same system or applications with multiple I/O behaviors



I/O Scheduler Selection First Steps

Dynamic Adaptability in Support of Extreme Scale

- Scheduler selection based on execution characteristics
- Scheduler selection guided by *a priori* measurements
 - Benchmark for *a priori* measurements -- cover entire range of metric

$f(\textit{metric}) \longrightarrow \textit{scheduler}$

- Recompute f transparent to system software and hardware



I/O Scheduler Selection Preliminary Approach-1

Dynamic Adaptability in Support of Extreme Scale

- Goal: maximize disk throughput, t
 - Metric: request size, r
 - Benchmark: For each scheduler, s , generate random reads/writes across range of r

$$t = f_s(r)$$

- Scheduler selected, $SS_r = \max_s f_s(r)$
- Average request size generated by workload indexes into SS



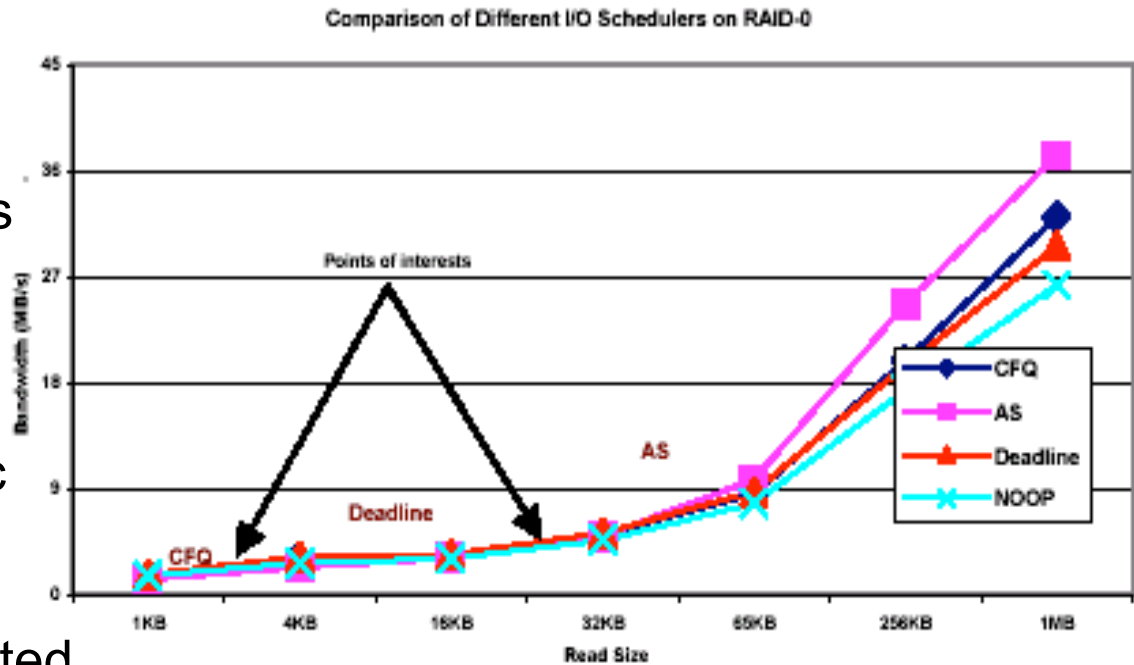
I/O Scheduler Selection Preliminary Approach-2

Dynamic Adaptability in Support of Extreme Scale

- 2.6.11 kernel
- Ext3 file system
- Raid-0 with 4 drives

• Assumption:
Throughput monotonic
with request size

- CAS is not integrated
- In general, AS is best
- For small random choice varies between CFQ and deadline
- Favors applications with large read/write sizes





I/O Scheduler Selection Questions

Dynamic Adaptability in Support of Extreme Scale

- Is *SS* the best scheduler for current workload?
- Why random reads/writes?
 - Do they cover all possible cases?
- Is request size a good metric?
 - What other metrics should be considered?
- Is throughput the only goal of interest?
- How does ratio of reads/writes factor into scheduler selection?



Acknowledgements

Dynamic Adaptability in Support of Extreme Scale

- We thank:
 - Nick Piggin: answering questions and sharing ideas
 - Steven Pratt, IBM LTC: valuable discussions
 - Sonny Rao, IBM LTC: help with FF SB
 - Jay Suresh, UTEP: help with experimentation
 - DOE (Grant # DE-FG02-04ER25622), IBM, and UTEP: financial support
 - Linux Symposium committee for organizing this great event
 - You: for your interest



Legal Stuff

Dynamic Adaptability in Support of Extreme Scale

This work represented the view of the authors, and does not necessarily represent the view of University of Texas-EI Paso or IBM.

IBM is a trademark of International Business Machines

Pentium is a trademark of Intel corporation

Other company, product, and service names may be trademarks or service marks of others

All benchmarking was conducted for research purpose only, under laboratory conditions

Results will not be realized in all computing environments



References

Dynamic Adaptability in Support of Extreme Scale

- Robert, C., Private Communications, 2005.
- Borril, J., J. Carter, L. Oliker, D. Skinner, and R. Biswas, “Integrated Performance Monitoring of a Cosmology Application on Leading HEC Platforms,” *Proceedings of the 2005 International Conference on Parallel Processing (ICPP-05)*, June 2005.
- Pratt, S., and D. Heger, IBM-Austin, “Workload Dependent Performance Evaluation of Linux 2.6 I/O Schedulers,” *Linux Symposium, 2*, July 2004, pp. 425-448.
- <http://www.sourceforge.net/projects/ffsb>



I/O Scheduler Selection Questions

Dynamic Adaptability in Support of Extreme Scale

- Is *SS* the best scheduler for current workload?
- Why random reads/writes?
 - Do they cover all possible cases?
- Is request size a good metric?
 - What other metrics should be considered?
- Is throughput the only goal of interest?
- How does ratio of reads/writes factor into scheduler selection?