Towards Dynamic Adaptivity of Operating Systems

Pat Teller
Professor, Computer Science
The University of Texas at El Paso
pteller@utep.edu
Outline

- Overview of DAiSES
- Infrastructure
- Research
  - UW
    - Kernel Performance
    - Scalability
  - UTEP
    - Proof of Concept: I/O Scheduling
    - Lessons Learned
    - Candidate Adaptation Targets
      - VMM Parameter Adaptation
      - Adaptive Page Size Allocation
      - SMT/CMP Scheduling
      - Virtualization
- Acknowledgments
- Publications
In a nutshell….

give the workload what it needs in order for the system and the workload to perform best

Collaboration among UTEP, University of Wisconsin-Madison (Bart Miller), and UT-Austin LTC (Bill Buros)
Overview Goals - 2

• Build dynamic adaptivity (of policies and parameters) into the Linux OS
• Deliver maximum attainable performance to diverse applications while meeting system constraints
• Develop general-purpose methodologies for dynamic adaptation of parameters and policies of stateful and stateless resources
• Develop mechanisms to dynamically sense, analyze, and adjust common performance metrics, fluctuating workload situations, and overall system environment conditions
• Demonstrate, via Linux prototypes and experiments, dynamic self-tuning and self-provisioning in HPC environments
Overview
Original Methodology

- Characterize application resource usage patterns
- Identify candidate adaptation targets that show promise in terms of enhancing performance
- Determine feasible adaptation ranges
- Define heuristics to trigger adaptations
- Implement monitoring, triggering, and adaptation code
- Quantify performance gains
Dynamic Adaptivity in Support of Extreme Scale

Overview

Methodology

- Identify adaptation targets
- Characterize workload resource usage patterns
- Define/adapt heuristics to trigger adaptation
- Generate/adapt monitoring, triggering, and adaptation code, and attach it to OS
- Monitor application execution, triggering adaptation as necessary

KernInst
Outline

• Overview of DAiSES
• **Infrastructure**
• Research
  – UW
    • Kernel Performance
    • Scalability
  – UTEP
    • **Proof of Concept: I/O Scheduling**
    • Lessons Learned
    • Candidate Adaptation Targets
      – VMM Parameter Adaptation
      – Adaptive Page Size Allocation
      – SMT/CMP Scheduling
      – Virtualization
• Acknowledgments
• Publications
At UTEP:

• Experimental platforms running experimental versions of Linux 2.6
  – four dual-processor Xeon workstations
  – IBM eServer pSeries 690, 590 and 550 (IBM SUR grant, UT System STARS Award)
  – Itanium2 cluster

• Workloads
  – SPEC OSG and HPG benchmarks
  – I/O: tiobench, FFSB, MADBench, SPECjAppServer2004, SPECmail, IOzone, I/O kernels (Bob Loewe-LLNL and Gary Grider-LANL)
  – NERSC5 (received recently through LBNL)
  – Memory: STREAMS, ASCI Purple Benchmarks (hopefully stress memory)
  – Process scheduling: Hackbench, Interbench, Sweep3D (daemons)

• Tools
  – oprofile
  – blktrace
  – systemd
  – kprobes
At UW:

- **Kerninst port for Power/Linux 2.6 (including hypervisor compatibility):** appears in Kerninst 2.1.2 beta
  - Removes dependence on /dev/kmem (because of too many variations from the various Linux distributors)
  - Lots of bug fixes and performance improvements
  - Demonstrated most recent Kerninst at the Paradyn/Dyninst annual meeting, March 2006

- **Developed a Linux 2.4/2.6 kernel profiler using KerninstAPI**
  - Identifies kernel functions invoked on behalf of a specific process by tracing call path execution starting at the system call interface
  - Extends the CrossWalk tool for tracing application performance problems across the user/system boundary
  - Generates call graph (using “dot”) to observe the control flow
  - Collects execution counts of edges in the call graph in order to help identify hot functions and paths
• Nodes are kernel functions

• Edges are calls (dotted lines are indirect calls detected at runtime)

• Edge labels are caller’s address and number of times called
Outline

• Overview of DAiSES
• Infrastructure
• **Research**
  – **UW**
    • Kernel Performance
    • Scalability
  – **UTEP**
    • **Proof of Concept: I/O Scheduling**
    • Lessons Learned
    • Candidate Adaptation Targets
      – VMM Parameter Adaptation
      – Adaptive Page Size Allocation
      – SMT/CMP Scheduling
      – Virtualization

• **Acknowledgments**
• **Publications**
Exploration of kernel performance of HPC applications:
• Tested UW tools on various applications, e.g.,
  – MILC su3_rmd
    • Found that application spends 8% of total CPU time in the kernel, and
    • `sys_read` and `sys_write` account for the majority of kernel CPU time, with each at around 3%.
  – OM3
    • Determined that application spends 12% of total CPU time in the kernel
    • The `sys_read` function accounted for the highest proportion of calls, corresponding to 6% of kernel CPU time.
  – BLAST (genetic sequence matching)
    • I/O took less than 6% of total running time, depending on the data set.
Dynamic Adaptivity in Support of Extreme Scale

Scalability

- Investigating issues relating to monitoring and control on the type of high-end systems targeted by FASTOS
  - Functions such as start-up, tracing, processing control and status monitoring
  - Evaluating in the context of various schedulers, process controllers (such as MPICH’s MPD), and tools (such as Totalview)
  - New process control, beyond BProc: looking at leveraging the 9P protocol from Plan 9 (with Ron Minnich at LANL) for a truly scalable process control facility
- Exploring the use of Kerninst as the instrumentation engine for Al Malony’s KTAU kernel profiling
Outline

• Overview of DAiSES
• Infrastructure
• Research
  – UW
    • Kernel Performance
    • Scalability
  – UTEP
    • Proof of Concept: I/O Scheduling
    • Lessons Learned
    • Candidate Adaptation Targets
      – VMM Parameter Adaptation
      – Adaptive Page Size Allocation
      – SMT/CMP Scheduling
      – Virtualization
• Acknowledgments
• Publications
Build a framework for dynamic adaptation of the I/O scheduler into the Linux OS
Deliver maximum attainable performance to diverse applications while meeting system constraints
Develop general-purpose methodology for dynamic adaptation of policies of stateless resources
Demonstrate, via Linux prototypes and experiments, dynamic self-provisioning
Dynamic Adaptivity in Support of Extreme Scale Challenge

I/O Scheduling

No silver bullet

Workloads with different I/O needs

Different system configurations

Different I/O schedulers
Solution
I/O Scheduling

• ADAPT!
• As workload characteristics change, switch to appropriate scheduler
• Get best performance for each different type of workload
Solution
I/O Scheduling

• ADAPT!
• As workload characteristics change, switch to appropriate scheduler
• Get best performance for each different type of workload
• Easier said than done!
• Linux 2.6 includes four schedulers (Anticipatory, Deadline, CFQ, and noop) with boot-time and run-time selection – this helped!
Significant Work in I/O Scheduling


• Developed/enhanced four disk scheduling algorithms
  – A new time-based disk scheduling algorithm; first ever algorithm to provide predictability and performance isolation (CFQ-CRR)
  – A new algorithm to exploit device queuing (CFQ-CRR with P)
  – A new algorithm (RDCLOOK) to improve disk utilization of asynchronous write requests – paper accepted to QEST’06, September 2006
  – Extension of Anticipatory Scheduler (Cooperative Anticipatory Scheduler – CAS) to mitigate starvation problem – paper in Linux Symposium, July 2005

• An explicit policy selection methodology
• An implicit policy selection methodology
I/O Scheduling Publications

Explicit Policy Selection: Combined Queuing and Policy

- Only one policy is active at any time; only one data delivery requirement can be satisfied at any time.
- Switching policies requires moving requests across queues or draining them.

System Model and I/O Schedulers

I/O Subsystem

I/O Scheduler

Policy 1

Policy 2

Policy N

Workload(s)

Device Queue

Storage System
Two-Policy Adaptation
explicit policy selection

- When CFQ (default) is active, both fairness and latencies are satisfied.
- When Deadline is active only latencies are satisfied; no guarantees on fairness.
- As the number of queued requests increases, the potential for not satisfying latency requirements may increase – in addition, adaptation takes longer due to draining.
Explicit Policy Selection

Conclusions

• Only one policy is active at any time; only one requirement can be satisfied at any time
• Either copying requests or draining for adaptation impacts performance
• Even with multiple policies only one requirement can be satisfied
  – Identifying conditions for adaptation is not always possible
• Single queue system and multiple policies are the way to go
Implicit Policy Selection: Separate Queuing and Policy

- Provide performance isolation
  - Applications should not be able to monopolize disk system
  - Most algorithms do not provide this

- Provide predictable performance
  - Unpredictability hinders performance guarantees

- Fair scheduling is the key to performance isolation and predictable performance
  - Allow satisfying multiple data delivery requirements
  - Each application could have its own scheduling algorithm
Separate Queuing and Policy

- **Policy 0**
  - must provide fairness, predictable performance, and performance isolation
  - controls allocation of I/O system to applications

- **Policy 1** could be for queue 1 or for queue 1 through K (0<K<=M)

- Examples
  - Policy for device queuing
  - Policy for async requests
Resource Sharing

- Fair queuing and round-robin scheduling is a well known approach for resource sharing
- Allocation metric depends on the shared resource type
- Resource allocation metrics
  - Number of requests
  - Amount of data
  - Resource time
Performance Target: Performance Isolation

- Given the number of I/O-intensive applications, the total disk time allocated to an application is not dependent on the characteristics of the other applications.
- Results in predictable performance.
- Thus, resource time allocation must be the fairness measure.
- Implement in OS or in disk controller.

Figure 1: Application Execution Times when a) both generate 4KB requests and b) when one generates 4KB requests and the other generates 512KB requests.
• Analysis of I/O schedulers w.r.t. sharing notion (number of requests, amount of data transferred, resource time) and fairness

• None of the I/O schedulers result in a fairness that results in performance isolation and performance predictability
Dynamic Adaptivity in Support of Extreme Scale Performance

Unpredictability of CFQ(N)

- A set \( s \) of requests is dispatched from each queue
- It takes longer to service a 512KB request than a 4KB request
- 29% increase in execution time

Figure 1: Application Execution Times when a) both generate 4KB requests and b) when one generates 4KB requests and the other generates 512KB requests.
**CFQ-Compensating Round Robin (CFQ-CRR) - 1**

- Idea: compensated disk-time metric
  - Requests are scheduled one-by-one until the quantum is exhausted
  - When a request is completed, its service time is subtracted from the queue’s quantum
  - Scheduling from a queue stops when the quantum is zero or negative
  - Quantum for next round is shortchanged with the negative quantum from this round
  - Unused quantum in a round is not carried to the next round
CFQ-Compensating Round Robin (CFQ-CRR) - 2

- Provides
  - Performance isolation
  - Predictable performance
  - QoS guarantees with different values of quantum
  - A framework for simultaneously satisfying multiple data delivery requirements
CFQ-CRR(P): Extension for TCQ Drives

- TCQ and NCQ improves disk utilization of workloads with random accesses; benefit for hyper threading processors
- CFQ-CRR cannot take advantage of TCQ drives; it dispatches one request at a time
- CFQ-CRR with P: dispatches multiple requests from each queue to fill device queue
- Continuous filling of device queue results in starvation
- Given: each queue has a time quantum

***System Model and I/O Schedulers***

I/O Subsystem

Policy 0  Policy 1  ...  Policy N

I/O Scheduler

Device queue

Storage System

Workload(s)

Queuing
Dynamic Adaptivity in Support of Extreme Scale

CFQ-CRR(P): Implicit Adaptation

- Given the quantum and the maximum bandwidth obtainable from a storage system
  - Can be measured in less than 1 sec.
  - Must be measured at each mount operation of the disk system; avoid cache effects
- Compute amount of data that can be transferred as the product of the quantum and maximum bandwidth
- Dispatch requests such that total data transferred is greater than or equal to estimated amount
- Compensate extra time in succeeding rounds
CFQ-CRR(P): Experimental Evaluation

Application execution times of the threads that finished first and last among 32 concurrent threads; 1000 random 4KB requests/thread

Maximum and average latency of requests with different schedulers; each thread accesses disjoint areas of the disk
CFQ-CRR(P): Extension for TCQ Drives

- Maintains strict fairness
- Results in better average and maximum latency compared to others
  - Deadline, Noop, and Anticipatory have more than 5% requests exceeding 1sec. latency
- Preserves performance predictability while using TCQ drives
- Performs poorly because it does not exploit TCQ and schedules one request at a time
Outline

- Overview of DAiSES
- Infrastructure
- **Research**
  - UW
    - Kernel Performance
    - Scalability
  - UTEP
    - **Proof of Concept: I/O Scheduling**
    - Lessons Learned
    - **Candidate Adaptation Targets**
      - VMM Parameter Adaptation
      - Adaptive Page Size Allocation
      - SMT/CMP Scheduling
      - Virtualization
- Acknowledgments
- Publications
Lessons Learned - 1

See Operating Systems Review article for more details.

• Identifying promising adaptation targets is a challenging and time-intensive task
  – Gain familiarity with related literature and Linux code
  – Identify applications/workloads that will be affected by targeted adaptation
  – Use static adaptation and a variety of workloads to quantify potential performance gains
  – Perform a feasibility study of the dynamic adaptation

• Complexities associated with parametric and policy adaptations differ significantly
  – Original methodology more directed at parametric adaptation
**Lessons Learned - 2**

See Operating Systems Review article for more details.

- **Adaptations come in two “flavors”**
  - Application performance objectives (relatively easy)
  - System performance objectives (concurrent tuning of multiple applications that share resources is decidedly more difficult)

- **Improved execution-time performance if not the only objective**
  - Necessary system constraints, e.g., fairness and latency

- **Different strategies are needed for resources with state and resources without state**
Candidate Adaptation Targets

- I/O scheduling policy adaptation
- I/O scheduling parameter adaptation
- Parametric adaptation of virtual memory manager
- Multiple page size management
- Network stack
- Scheduling of chip multiprocessors
- File I/O
- Virtualization
Outline

• Overview of DAiSES
• Infrastructure
• Research
  – UW
    • Kernel Performance
    • Scalability
  – UTEP
    • Proof of Concept: I/O Scheduling
    • Lessons Learned
    • Candidate Adaptation Targets
      – VMM Parameter Adaptation
      – Adaptive Page Size Allocation
      – SMT/CMP Scheduling
      – Virtualization

• Acknowledgments
• Publications
VMM Parameter Adaptation

- Builds on work of Gokul Kandiraju (Penn State)
- Master’s thesis (Ricardo Portillo): using static adaptation and different types of applications, understand the effect of changing one or more parameters
- Preliminary results: in the case of SPEC APSI and the SCM parameter (minimum no. of pages freed on a reclamation pass due to failure to allocate memory), 63% improvement in terms of both execution time and number of page faults
- But Stephen Poole says: “What’s a swap?”
Dynamic Adaptivity in Support of Extreme Scale

VMM Parameter Adaptation

63% improvement as compared to default value of SCM

10 runs of apsi for each SCM value

Elapsed Time (seconds)

SCM value

Default Value

Best Value
Adaptive Page Size Allocation

- Builds on work of Juan Navarro (Rice)
- Reduce TLB misses
- Linux Symposium – BoF on supporting multiple page sizes
- Exploring how to experiment with ideas
  - K42?
  - Itanium cluster?
• **Goals**
  – Better processor and system utilization
  – Less interference w.r.t. cache – impacts synchronization (Beckman, et al.)

• **Initial Objectives**
  – Characterize interference of classes of applications running on SMT processors
  – Develop co-scheduling heuristics
  – Use heuristics to tune SMT knobs like hardware thread priorities, SMT On/Off, and SMT Snooze for maximizing system performance (IPC)
  – Explore on-the-fly SMT-knob tuning in kernel space
Virtualization

- IBM eServer pSeries 550
- Impact of scheduling decisions
  - Allocations
  - Capped and uncapped
- Overhead
  - Memory
  - Performance
- SPECjAppServer
- Scientific workloads
- Funded by IBM-Austin
Outline

- Overview of DAiSES
- Infrastructure
- Research
  - UW
    - Kernel Performance
    - Scalability
  - UTEP
    - **Proof of Concept: I/O Scheduling**
    - Lessons Learned
    - Candidate Adaptation Targets
      - VMM Parameter Adaptation
      - Adaptive Page Size Allocation
      - SMT/CMP Scheduling
      - Virtualization
- Acknowledgments
- Publications
Acknowledgments

• DOE (Grant # DE-FG02-04ER25622)
• IBM Corporation for IBM Shared University Research Grant (SUR)
• IBM-Austin LTC, especially Bill Buros
• Linux Developers, especially Jens Axboe
• UTEP
General Challenges

• Overhead to
  – Identify effective tools and learn how to use them
  – Identify target benchmarks/applications and get them running
    • Real applications for proofs of concept

• Experimental platforms
  – File I/O (Beckman, et al.)
  – K42 (Paul Hargove, et al.)