CS838: Special Topics in OS
Gray-Box Systems

Andrea C. Arpaci-Dusseau
Fall 2002
Department of Computer Sciences,
University of Wisconsin, Madison
www.cs.wisc.edu/graybox

Introductory Lecture

- Overview of two parts of course
  - Motivation for subject material
  - Review reading list
  - Format of class
  - Readings for next lecture

Overview: Two parts of course

- First part of course
  - Building systems of systems
  - Challenge: dealing with inadequate interfaces to existing systems
  - Solution: Gray-Box Systems
    - Useful techniques
      - Microbenchmarking
      - Fingerprinting
      - Reverse engineering
      - Algorithmic mirroring (Simulation)
    - Case studies
      - TCP, implicit coscheduling, MS Manners, Visual Proxies,
      - Semantically smart disks, Differential Power Analysis

- Second part of course
  - Building a new system from scratch
  - Challenge: Design system w/ more complete interface
  - Solution: Unknown
    - Expose more information/control
    - Choose implementation easier to express
    - Allow applications/OS to adapt control to current information
  - Open-ended readings
Part 1

Motivation

- Software systems are increasingly complex
- Early days
  - Build new system from scratch
  - Or, Build from existing simple components
  - Can understand details of all components

Simple System

New System

Motivation

- Current state of software design
  - Impossible to build system from scratch
  - Leverage complex systems as components
  - Impossible to understand details of components

*Systems of Systems*
System vs. Component

• What are the similarities?
  • Both are modular entities
    • Interface
      • Defines outputs as function of previous and current inputs
      • Hides unnecessary details
    • Implementation: Independent of interface

Merriam-Webster definition of "system"
  • "A group of units so combined as to form a whole and to operate in unison"

Andrea’s definition
  • “A component becomes a system when its interesting behavior can no longer be precisely or succinctly defined”

Interface fails to describe useful information
  • Does not explain performance
  • Does not describe internal algorithms or policies
  • Does not expose internal state

Outline

• Systems of Existing Systems
  • Motivation
  • Challenges
  • Examples
  • Gray-Box Systems
  • Techniques and Case Studies
  • Designing a new System

Challenges of Building Systems

• Choose correct components or functions
  • Which implementation is most appropriate?
  • Requirements: Performance, memory, power usage (as a function of usage)
  • Example: Two list implementations - sorted or not
    • Sorted: Fast lookup, slow insert
    • Unsorted: Slow lookup, fast insert
  • Depends upon common operations in workload

• Adapt to behavior of components
  • How to use in way it was optimized for?
  • Example: Implementation uses a cache of N elements
    • Reorder usage such that working set does not exceed N
What is Needed?

- More information about components
  - Cost of operation (performance)
    - Example: Inserts vs. lookups
  - Value of parameter (configuration)
    - Example: Size of cache
  - Existing policy (algorithm)
    - Example: Replacement algorithm for cache
  - Internal state (dynamic)
    - Example: What is in cache?

- More control over components
  - Change policies
    - Example: Change replacement policy from LRU to LFU

Outline

- Systems of Existing Systems
  - Motivation
  - Challenges
  - Examples
  - Gray-Box Systems
  - Techniques and Case Studies
- Designing a new System

Simple Examples

- Focus on systems where OS is one of the building blocks
- Questions
  - What type of information and/or control does each component want from others?
  - What do current interfaces provide?
  - How is the information/control obtained when an interface doesn’t exist?
  - What assumptions are usually made about the other components?

Example #1: Base Case

What information does OS want about workload?
What information could application use about OS?
(cost, parameters, policies, state)
Example #2: Special Case

OS policies (and interfaces) do not meet database needs
What OS policies (control) are inappropriate for databases?

Example #3: Multiple Layers

What does an application need to know about the db?

Example #4: Not just software

What information is useful to know about each?

Example #5: Multiple nodes

What does distributed service want to know?
Example #6: Networks

Outline

- Systems of Existing Systems
  - Motivation
  - Challenges
  - Examples
- Gray-Box Systems
  - Techniques and Case Studies
- Designing a new System

Main Problem

- What should we do when system does not provide interface for desired information and/or control?
- Assume cannot change interface or implementation

Solution: Gray-Box Systems:

- Assume basic behavior of component beyond interface
- Sources of this base knowledge
  - Understanding of common techniques
  - Access to previous versions of code
  - Some high-level documentation
  - Skimming of source code
- Probe component and observe reactions to confirm and learn more
Alternatives

- Black-Box Systems
  - Defined only by interface; assume nothing about internals
  - Problem: Difficult to learn about without making some assumptions
- White-Box Systems
  - Can see (and change?) all details of internals
  - Problem: Not realistic

Realistic Assumptions

- Component implemented by someone else
- Cannot change implementation
- Cannot change or add interfaces
  - Standards must be consistent
- Source code may or may not be available
  - Complex enough that difficult to understand
- Documentation may or may not be available
  - Cannot trust it to be correct or complete

Outline

- Systems of Existing Systems
  - Motivation
  - Examples
  - Gray-Box Systems
  - Techniques and Case Studies
- Designing a new System

Course Goals

- What are useful techniques for gray-box systems?
  - Under what circumstances are each appropriate?
  - What are the strengths and weaknesses of each?
  - What are the costs and overheads?
- What information and/or control can be acquired with these techniques?
  - When must one change an implementation?
  - When must one add an interface?
- How should systems be designed to help?
Useful Techniques #1

- Acquiring additional information not exposed by interface
- Information: Cost of operations and key parameters
  - Solution: Microbenchmarking
    - Perform operations, time results
    - Use time to infer parameters
  - Reading List
    - Cache and TLB: Saavedra
    - Mhz: Staelin and McVoy
    - Disks: 3 papers

Useful Techniques #2

- Information: policies and algorithms
- Solution: Fingerprinting
  - More general (and complex) than microbenchmarks
  - Add new probes or use existing operations
  - Observe "results" of operations
    - Any covert channel: Time, power, cache misses
  - Combine results with knowledge to infer policies

Useful Techniques #3

- Information: Internal state of component
- Solution #1: Observations
  - Observe outputs; combine with general knowledge to correlate with internal state
- Solution #2: Algorithmic mirroring
  - Observe (all) inputs to other component
  - Simulate state of component given detailed knowledge of policies
- Reading List
  - Self-monitoring and self-adapting OS
  - ISTORE
  - Others???
Useful Techniques #4

- Exerting control outside of interface?
- Solution: Control-theory
- Reading List
  - Entire (different) course
  - Two case studies

Case Studies

- Networking and distributed systems
  - TCP and RED
  - Implicit coscheduling
- Application behavior
  - Visual proxies
  - MS Manners
- Control Theory
- Semantically smart disks
- Security
  - Differential power analysis
  - Differential fault analysis

Part 2

- Previous problem
  - How to deal with building blocks and interfaces that are wrong
- New problem
  - Building a new system from scratch
  - How to build a better building block and interface
  - Focus on OS as building block
- Brief overview today
  - More in second half of course

Outline

- Systems of Existing Systems
- Designing a new System
  - Three goals
  - Implication of each
  - Reading list for each
Goal #1 for new OS

- Expose all "useful" information through explicit interfaces (to apps and subsystems)
  - Examples
    - Cost of operations
    - Parameter values
    - Algorithms and policies
    - Internal state
  - Problems?

Goal #2 for new OS

- Allow applications to modify policies (control)
- Extensible operating systems
  - No single policy is best for every workload
    - Microkernels: Remove all policies, export only mechanisms (and/or protection)
    - Download safe code to change policy
  - Problems?
Goal #2 for new OS

- Allow applications to modify policies (control)
- Extensible operating systems
  - No single policy is best for every workload
    - Microkernels: Remove all policies, export only mechanisms (and/or protection)
    - Download safe code to change policy
- Problems?
  - Difficult to remove all policies
  - Must have policy to arbitrate across different processes
  - Higher cost w/ policies in user-space
  - Hard to ensure downloaded code composes

(Possible) Implication #2

- Implement configurable default policies
  - Example: Prefetching with configurable units
- Advantages
  - Arbitration: More flexible
  - Low cost: Policy in-kernel; few interactions
  - Composable: Anyone can specify
- Drawback: Limits extensibility
- Reading List: Extensible systems
  - Synthesis
  - SPIN
  - Exokernel
  - VINO
  - Scout

Goal #3 for new OS

- Combine goals 1 and 2
- Applications control OS as a function of current state in OS (and HW)
- Examples
  - Schedule process when buffer cache (or data or instruction cache) is loaded with “useful” data
  - Prefetch disk blocks when disk head is near blocks and application will access blocks
- Problems
  - How to decide fast? (w/o going back to user)?

(Possible) Implication #3

- Express information in terms of “cost” and “benefit”
  - OS: Charge cost to process for using resources
    - Scheduling and unloading cache of others data
    - Moving disk arm some distance
  - Application: Calculates benefit of performing set of operations
    - Reuse of data set in cache
    - Expected benefit of prefetching
  - OS: Performs operation when benefit > cost
(Possible) Implication #3

- Related Work
  - Many individual case studies of cost-benefit
    - Affinity scheduling
    - cohort scheduling
    - SLEDS
    - anticipatory scheduling
    - compiler directed prefetching
  - Cost models, Economic models
- Goal: General framework???

Summary

- Complex systems of systems
  - Interfaces don’t reveal all useful information
  - Interfaces prevent interesting control
  - Unrealistic to assume can change interface
- Existing systems: Need ways to acquire new information and control
  - Gray-Box systems
  - Techniques: Microbenchmarking, Fingerprinting, Reverse engineering, simulation
- New system: More expressive interfaces; implementations that lead to better interfaces