Introduction
CS 239
Experimental Methodologies for System Software
Peter Reiher
April 3, 2007

Purpose of Class

- To teach graduate students working in systems software how to design, run, and interpret experiments
- To give such students some experience in experimentation

Introduction

- Basic course information
- Grading
  - Projects
  - Homework
- Textbook and web page
- Office hours
- Outline of class

Basic Course Information

- Pre-requisites - CS111
- Professor - Peter Reiher
- Email address-
  - reiher@cs.ucla.edu

What Will This Course Teach You?

- Proper methods to design and perform experiments on system software
- Proper methods to analyze and present data gathered from such experiments
- Proper methods to critique experiments and data produced by others

What Won't This Course Teach You?

- Basic systems software principles
- Systems software modeling
- Queueing theory
- Simulation techniques for systems software
- Background in statistics

Who Should Take This Course

Well, everyone, but especially -

- software developers
- software researchers
- software purchasers
- software evaluators

Grading

- Project - 60%
- Evaluation of other students' projects - 10%
- Homework - 30%
- Course is offered for variable units, but sign up for 4 units

Project Information

- Design and perform evaluation of a real software system
- Present plans in class
- Present results in class
- Final written report
- Evaluate others' projects
  - In a written report

Suitable Subjects for Projects

- Operating systems
- OS components (file systems, I/O subsystems, process handling, etc.)
- Compilers
- Databases
- Real time applications
- Large application packages
- Distributed systems
- Networks/networking systems

Project Formats

- Group projects
  - Size of group dependent on number of students in class
  - Groups chosen by students
- Project topic chosen by the group
- All group members must participate in all group activities

Written Materials for Project

- Project proposal (1-2 pages)
  - Due April 26
- Project design (5-8 pages)
- Final report (15+ pages)
  - Due June 14th, 5 PM

In-Class Presentations

- Detailed presentation of project designs (May 3)
- Presentation of results (June 5 & 7)
- Unless teams are large, all group members are expected to help present
- Length of presentation will depend on number of groups (but at least 1/2 hour for final presentation)

Grading of Projects

Several criteria will be used:

- Proper design of the experiment
- Care and thoroughness of its execution
- Completeness of analysis
- Quality of data presentation
- Insight gained from experiment

Evaluation of Other Groups' Projects

- Submitted by each student individually
- 1 page critique of each group's proposed experiment
  - Due May 8, 5 PM
- 1 page critique of each group's results
  - Due June 13, 5 PM
- Graded on basis of insight into strengths and flaws of each project
- Email submission fine

Homework

- 5 homework sets worth 6% each
- Assigned Wednesday each of 2nd-6th weeks
- Due Wednesday of the following week
  - Via email or hard copy
- Homeworks must be done individually, not by group

Textbook

The Art of Computer Systems Performance Analysis

Raj Jain

- Readings assigned weekly
- Students expected to find and read required materials to perform projects
- First week's assignment - Chapters 1-3

Class Web Page

http://www.lasr.cs.ucla.edu/classes/239_1.spring07

- Slides from lectures posted at least two hours before class (in handout format)
- List of upcoming important dates/deadlines
- Homework assignments
- Other material relevant to the class

Office Hours

- In BH3532F
- TT 2-3
- Instructor also available by appointment

Class Outline

- Introduction (1 class)
- Review of probability & statistics (3 classes)
- System measurement techniques and experimental design (7 classes)
- Presentation of project designs (1 class)
- Testbeds (1 class)
- Graphical techniques (1 class)
- Analyzing example systems (2 classes)
- Critiquing performance evaluations (1 class)
- Presentation of project results (2 classes)

Overview of Analysis of Software Systems

- Introduction
- Common mistakes made in system software analysis
  - And how to avoid them
- Selection of techniques and metrics

Introduction

- Why do we care about performance evaluation?
- Why is it hard?
- What tools can we use to understand system performance?

Why Do We Care?

- Performance is almost always a key issue in software
  - Especially in system software
- Everyone wants the best possible performance
- Cost of achieving performance is also key
- Reporting performance is necessary in many publication venues
  - Academic and industry

Importance of Performance in Research

- In almost all CS research, performance is a key
- **A solution that doesn't perform well isn't a solution at all**
- Successful research must prove its performance characteristics to a skeptical community

State of Performance Evaluation in the Field

- Generally regarded as poor
- Many systems have little performance data presented
- Many systems are measured by improper criteria
- Many experiments are poorly designed
- Many results are badly or incorrectly presented

With the Result?

- You can't always trust what you read in a research paper
- Authors may have accidentally or intentionally misled you
  - Overstating performance
  - Hiding problems
  - Not answering the important questions

Where Does This Problem Come From?

- Mostly from ignorance of proper methods of measuring and presenting performance
- Abetted by reader's ignorance of what questions they should be asking

But the Field Is Improving

- People are taking performance measurement more seriously
- Quality of published experiments is increasing
- So yours had better be of high quality, too
- And publishing is tough
  - Be at the top of the heap of papers

Sample Performance Measurement Problems

- To be used as running examples throughout the class
- Illustrate a wide variety of the problems and issues of system measurement
- Using real systems, real problems, and (some) real numbers

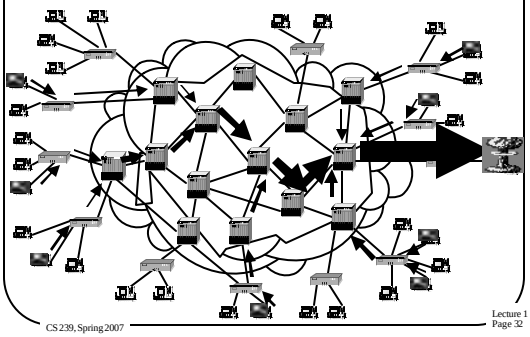
Some Sample Systems

- DefCOM – a system for defending against distributed denial of service attacks
- Conquest – a file system designed to improve performance
- Time Warp – a parallel simulation platform intended to run simulations fast
- Ficus – a replicated file system to offer higher file availability in mobile environments

DefCOM

- A defensive system to counter distributed denial of service (DDoS) attacks
- Especially attacks based on high volumes of garbage traffic
  - Originating from many sources

The DDoS Problem



Why Distributed Attacks?

- Targets are often highly provisioned servers
- A single machine usually cannot overwhelm such a server
- So harness multiple machines to do so
- Also makes defenses harder

How to Defend?

- A vital characteristic:
  - Don't just stop a flood
  - ENSURE SERVICE TO LEGITIMATE CLIENTS!!!
- If you only deliver a manageable amount of garbage, you haven't solved the problem

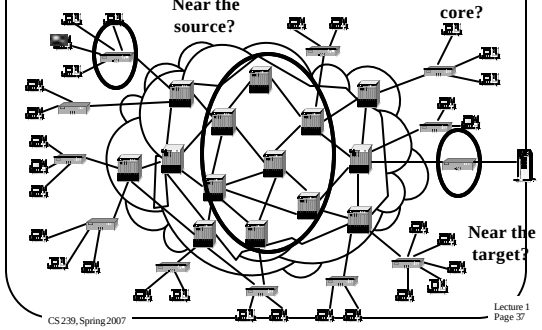
Complicating Factors

- High availability of compromised machines
  - At least tens of thousands of zombie machines out there
- Internet is designed to deliver traffic
  - Regardless of its value
- IP spoofing allows easy hiding
- Distributed nature makes legal approaches hard
- Attacker can choose all aspects of his attack packets
  - Can be a lot like good ones

DefCOM Defense Approach

- Addresses the core problem:
  - Too much traffic coming in, so get rid of some of it
  - A common idea in DDoS defense
- Vital to separate the sheep from the goats
- Unless you have good discrimination techniques, not much help

Where Do You Filter?



Filtering Near the Target

+ Easier to detect attack
+ Sees everything
+ Obvious deployment incentive
? May be hard to prevent collateral damage
? May be hard to handle attack volume

Filtering Near the Sources

+ Easier to prevent collateral damage
+ Easier to handle attack volume
? May be hard to detect attack
? Only works where deployed
? Deployment incentives?

Filtering in the Internet

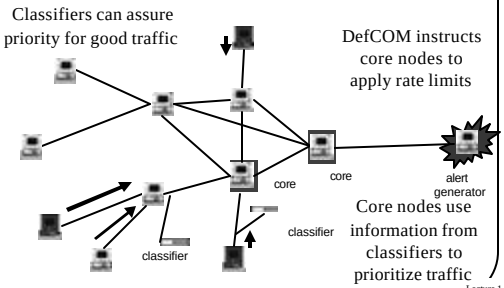
+ Spreads attack volume over many machines
+ Sees everything
  - With sufficient deployment
  - Which can be quite reasonable

? May be hard to prevent collateral damage
? May be hard to detect attack
? Low per-packet processing budget
? Deployment incentive?

What If All Parties Cooperated?

- Could we leverage strengths of all locations?
- While minimizing their weaknesses?
- That's the DefCOM approach
- A prototype system built at U Delaware and UCLA

DefCOM



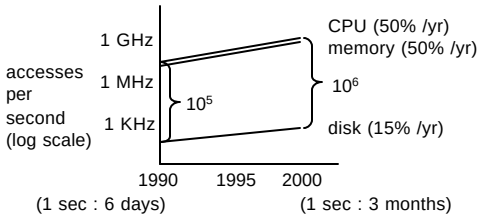
Performance Questions for DefCOM

- How well does DefCOM defend against attacks?
- Does DefCOM damage performance of normal traffic?
- Can all DefCOM components run fast enough for realistic cases?
- How much does partial deployment pattern matter?

Conquest

- A file system meant to improve performance
- Key observation is that disks suck
  - Always have, but sucking harder every year
- Vast amounts of OS effort spent in hiding how badly disks suck

The Key Hardware Trend



A Solution: Use Persistent RAM

- RAM that saves its state when power goes off
- Speed similar to regular RAM
- Battery-backed DRAM available today
- Flash RAM also common
  - But some bad characteristics
- Other forms of persistent RAM under development

Basic Idea of Conquest

- Use a few gigabytes of persistent RAM
- Store many files permanently in persistent RAM
  - Also store all metadata there
- Use disk only for big files
  - Mostly accessed sequentially
  - Which is OK for disks
- Prototype built here at UCLA

Performance Questions for Conquest

- How much faster than pure disk?
- Can it perform better than just persistent caching?
- What about performance of big files?

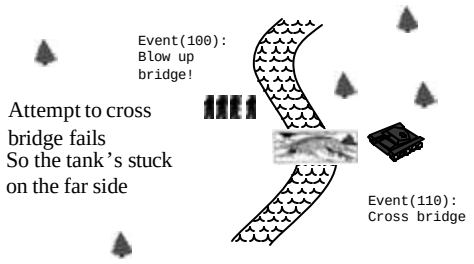
Time Warp

- Engine for running discrete event simulations in parallel
- Using "interesting" synchronization mechanisms
- Goal is essentially to run things faster
  - Than sequentially
  - Than competing parallel methods

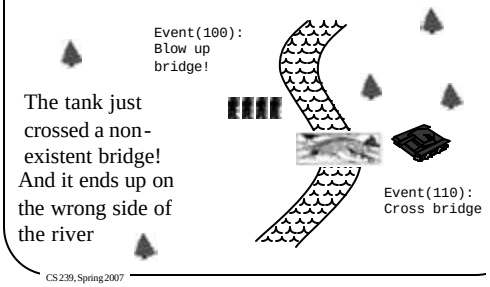
Discrete Event Simulations

- Simulate a system by simulating individual events that comprise it
- Events scheduled/communicate via messages
- Parallelize by running multiple events simultaneously
- Key constraint is must get same results as if all events run sequentially
- Issues of proper event ordering vital

Illustrating the Problem



Illustrating the Problem, Con't



Basic Idea Behind Time Warp

- Be optimistic
- Run as many events in parallel as you can
  - Which could mean you run some out of order
- Detect out-of-order events, roll them back, and rerun them properly
  - Also rolling back all their side effects
  - Like scheduling other events
- Prototype built at JPL
  - Based on idea from UCLA professor

Performance Questions for Time Warp

- Can it speed up simulations?
- How much benefit do you get from adding more hardware?
- Which internal optimizations are worthwhile?
- Can it run simulations faster than conservative methods?
- How do optimistic artifacts (like rollbacks) affect performance?

Ficus

- A replicated file system
- Keeps multiple copies of files on different machines
  - Including possibly disconnected portable machines
  - Uses optimistic synchronization
- Benefits are availability and local performance
- Issues are overall performance and effects of conflicts

Basic Idea Behind Ficus

- Store replicas where users might need them
- Allow updates at any replica
- Propagate updates to other replicas
- Can lead to consistency problems
  - Detect them
  - Correct them (automatically, when possible)
- Prototype built at UCLA

Performance Issues for Ficus

- Is it really cheaper than remote access?
- What are the performance costs of maintaining multiple replicas?
- What are the costs of updates?
- How often do conflicting updates occur?
- How often is stale data read from a replica that hasn't gotten the latest update?