# Operating System Principles: Performance Measurement and Analysis

# CS 111

# Operating Systems

Peter Reiher

# Outline

- Introduction to  performance measurement
- Issues in performance measurement
- A performance measurement example

# Performance Measurement

- Performance is almost always a key issue in software
- Especially in system software like operating systems
- Everyone wants the best possible performance
  - But achieving it is not always easy
  - And sometimes involves trading off other desirable qualities
- How can we know what performance we've achieved?
  - Especially given that we must do some work to learn that

# Performance Analysis Goals

- Quantify the system performance
  - For competitive positioning
  - To assess the efficacy of previous work
  - To identify future opportunities for improvement
- Understand the system performance
  - What factors are limiting our current performance
  - What choices make us subject to these limitations
- Predict system performance

# An Overarching Goal

- This applies to any performance analysis you ever do:
- **We seek wisdom, not numbers!**
- The point is never to produce a spreadsheet full of data
- The point is to understand critical performance issues

# Why Are You Measuring Performance?

- Sometimes to understand your system's behavior
- Sometimes to compare to other systems
- Sometimes to investigate alternatives
  - In how you can configure or manage your system
- Sometimes to determine how your system will (or won't) scale up
- Sometimes to find the cause of performance problems

# Why Is It Hard?

- Components operate in a complex system
  - Many steps/components in every process
  - Ongoing competition for all resources
  - Difficulty of making clear/simple assertions
  - Systems may be too large to replicate in laboratory
  - Or have other non-reproduceable properties
- Lack of clear/rigorous requirements
  - Performance is highly dependent on specifics
    - What we measure, how we measure it
  - Ask the wrong question, get the wrong answer

# Performance Analysis

- Can you characterize latency and throughput?
  – Of the system?
  – Of each major component?
- Can you account for all the end-to-end time?
  – Processing, transmission, queuing delays
- Can you explain how these vary with load?
- Are there any significant unexplained results?
- <u>Can you predict the performance of a system?</u>
  – As a function of its configuration/parameters

# Design For Performance Measurement

- Successful systems will need to have their performance measured
- Becoming a successful system will generally require that you improve its performance
  - Which implies measuring it
- It's best to assume your system will need to be measured
- So put some forethought into making it easy

# How To Design for Performance

- Establish performance requirements early
- Anticipate bottlenecks
  - Frequent operations (interrupts, copies, updates)
  - Limiting resources (network/disk bandwidth)
  - Traffic concentration points (resource locks)
- Design to minimize problems
  - Eliminate, reduce use, add resources
- Include performance measurement in design
  - What will be measured, and how

# Issues in Performance Measurement

- Performance measurement terminology
- Types of performance problems

# Some Important Measurement Terminology

- Metrics
  - Indices of tendency and dispersion
- Factors and levels
- Workloads

# Metrics

- A metric is a measurable quantity
  - Measurable: we can observe it in situations of interest
  - Quantifiable: time/rate, size/capacity, effectiveness/reliability …
- A metric's value should describe an important phenomenon in a system
  - Relevant to the questions we are addressing
- Much of performance evaluation is about properly evaluating metrics

# Common Types of System Metrics

- Duration/ response time
  - How long did the program run?
- Processing rate
  - How many web requests handled per second?
- Resource consumption
  - How much disk is currently used?
- Reliability
  - How many messages were delivered without error?

# Choosing Your Metrics

- Core question in any performance study
- Pick metrics based on:
  - Completeness: will my metrics cover everything I need to know?
  - (Non-)redundancy: does each metric provide information not provided by others?
  - Variability: will this metric to show meaningful variation?
  - Feasibility: can I accurately measure this metric?

# Variability in Metrics

- Performance of a system is often complex
- Perhaps not fully explainable
- One result is variability in many metric readings
  - You measure it twice/thrice/more and get different results every time
- Good performance measurement takes this into account

# An Example

- 11 pings from UCLA to MIT in one night
- Each took a different amount of time (expressed in msec):

149.1 28.1 28.1 28.5 28.6 28.2
28.4 187.8 74.3 46.1 155.8

- How do we understand what this says about how long a packet takes to get from LA to Boston and back?

# Where Does Variation Come From?

- Inconsistent test conditions
  - Varying platforms, operations, injection rates
  - Background activity on test platform
  - Start-up, accumulation, cache effects
- Flawed measurement choices/techniques
  - Measurement artifact, sampling errors
  - Measuring indirect/aggregate effects
- Non-deterministic factors
  - Queuing of processes, network and disk I/O
  - Where (on disk) files are allocated

# Tendency and Dispersion

- Given variability in metric readings, how do we understand what they tell us?
- Tendency
  – What is common or characteristic of all readings?
- Dispersion
  – How much do the various measurements of the metric vary?
- Good performance experiments capture and report both

# Indices of Tendency

- What can we compactly say that sheds light on all of the values observed?
- Some example indices of tendency:
  - Mean ... the average of all samples
  - Median ... the value of the middle sample
  - Mode ... the most commonly occurring value
- Each of these tells us something different, so which we use depends on our goals

# Applied to Our Example Ping Data

- Mean: 71.2
- Median: 28.6
- Mode: 28.1
- Which of these best expresses the delay we saw?
  - Depends on what you care about

149.1 28.1 28.1 28.5 28.6 28.2
28.4 187.8 74.3 46.1 155.8

# Indices of Dispersion

- Compact descriptions of how much variation we observed in our measurements
  - Among the values of particular metrics under supposedly identical conditions
- Some examples:
  - Range – the high and low values observed
  - Standard deviation – statistical measure of common deviations from a mean
  - Coefficient of variance – ratio of standard deviation to mean
- Again, choose the index that describes what's important for the goal under examination

# Applied to Our Ping Data Example

- Range: 28.1,188
- Standard deviation: 62.0
- Coefficient of variation: .87

149.1 28.1 28.1 28.5 28.6 28.2
28.4 187.8 74.3 46.1 155.8

# Capturing Variation

- Generally requires repetition of the same experiment
- Ideally, sufficient repetitions to capture all likely outcomes
  - How do you know how many repetitions that is?
  - You don't
- Design your performance measurements bearing this in mind

# Meaningful Measurements

- Measure under controlled conditions
  - On a specified platform
  - Under a controlled and calibrated load
  - Removing as many extraneous external influences as possible
- Measure the right things
  - Direct measurements of key characteristics
- Ensure quality of results
  - Competing measurements we can cross-compare
  - Measure/correct for artifacts
  - Quantify repeatability/variability of results

# Factors and Levels

- Sometimes we only want to measure one thing
- More commonly, we are interested in several alternatives
  - What if I doubled the memory?
  - What if work came in twice as fast?
  - What if I used a different file system?
- Such controlled variations for comparative purposes are called *factors*

# Factors in Experiments

- Choose factors related to your experiment goals
- If you care about web server scaling, factors probably related to amount of work offered
- If you want to know which file system works best for you, factor is likely to be different file systems
- If you're deciding how to partition a disk, factor is likely to be different partitionings

# Levels

- Factors vary (by definition)
- Levels describe which values you test for each factor
- Levels can thus be numerical
  - Number of web requests applied per second
  - Amount of memory devoted to I/O buffers
- Or they can be categorical
  - Btrfs vs. Ext3 vs. XFS

# Choosing Factors and Levels

- Your experiment should look at all vital factors
- Each factor should be examined at important levels
- But . . .
- The effort involved in the experiment is related to (number of factors) X (number of levels)
- If you’re not careful, this can cause your effort to explode
  - Especially if you repeat runs to capture variation

# Measurement Workloads

- Most measurement programs require the use of a *workload*
- Some kind of work applied to the system you are testing
  - Preferably similar to the work you care about
- Can be of several different forms
  - Simulated workloads
  - Replayed trace
  - Live workload
  - Standard benchmarks

# Simulated Work Loads

- Artificial load generation
  - On-demand generation of a specified load
- Strengths
  - Controllable operation rates, parameters, mixes
  - Scalable to produce arbitrarily large loads
  - Can collect excellent performance data
- Weaknesses
  - Random traffic is not a usage scenario
  - Simulation may not create all realistic situations
  - Wrong parameter choices yield unrealistic loads

# Replayed Workloads

- Captured operations from real systems
- Strengths
  - Represent real usage scenarios
  - Can be analyzed and replayed over and over
- Weakness
  - Often hard to obtain
  - Not necessarily scalable
    - Multiple instances not equivalent to more users
  - Represent a limited set of possible behaviors
  - Limited ability to exercise little-used features
  - They are kept around forever, and become stale

# Testing Under Live Loads

- Instrumented systems serving clients
- Strengths
  - Real combinations of real scenarios
  - Measured against realistic background loads
  - Enables collection of data on real usage
- Weakness
  - Demands good performance and reliability
  - Potentially limited testing opportunities
  - Load cannot be repeated or scaled on demand

# Standard Benchmarks

- Carefully crafted/reviewed simulators
- Strengths
  - Heavily reviewed by developers and customers
  - Believed to be representative of real usage
  - Standardized and widely available
  - Well maintained (bugs, currency, improvements)
  - Allows comparison of competing products
- Weakness
  - Inertia, used where they are not applicable

# Types of Performance Problems

- Non-scalable solutions
  - Cost per operation becomes prohibitive at scale
  - Worse-than-linear overheads and algorithms
  - Queuing delays associated with high utilization
- Bottlenecks
  - One component that limits system throughput
- Accumulated costs
  - Layers of calls, data copies, message exchanges
  - Redundant or unnecessary work

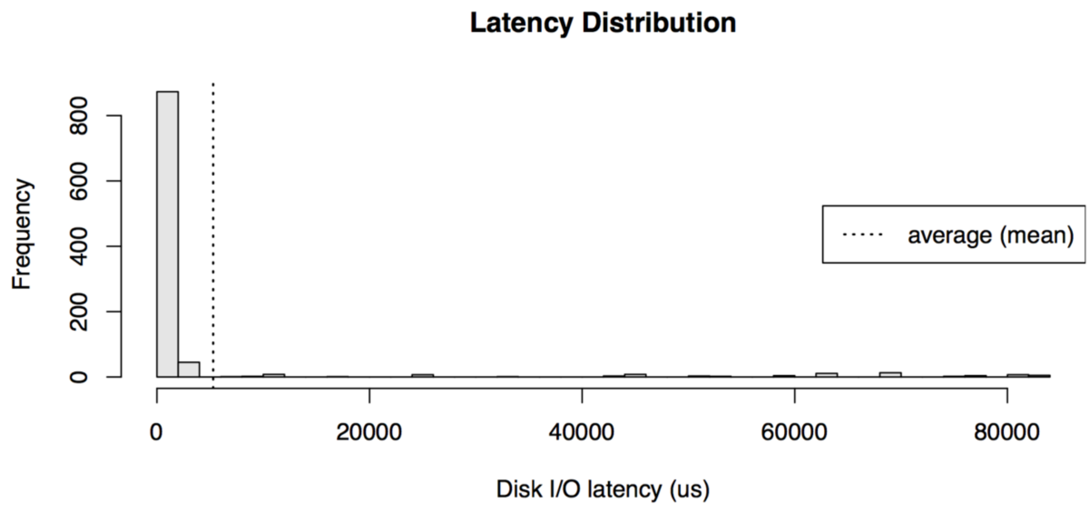
# Dealing With Performance Problems

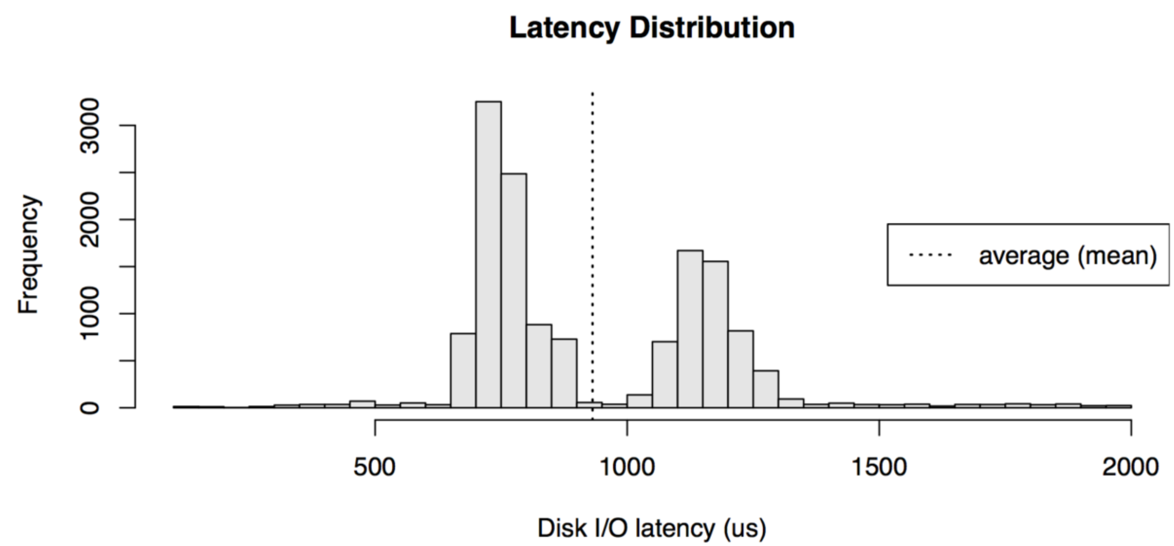
- A lot like finding and fixing a bug
  - Formulate a hypothesis
  - Gather data to verify your hypothesis
  - Be sure you understand underlying problem
  - Review proposed solutions
    - For effectiveness
    - For potential side effects
  - Make simple changes, one at a time
  - Re-measure to confirm effectiveness of each
- Only harder

# Common Measurement Mistakes

- Measuring time but not utilization
  - Everything is fast on a lightly loaded system
- Capturing averages rather than distributions
  - Outliers are usually interesting
- Ignoring start-up, accumulation, cache effects
  - Not measuring what we thought
- Ignoring instrumentation artifacts
  - They may greatly distort both times and loads

# Averages Don't Tell the Story



# Cache, Accumulation Start-up Effects

- Cached results may accelerate some runs
  - Random requests that are unlikely to be in cache
  - Overwhelm cache with new data between tests
  - Disable or bypass cache entirely
- Start-up costs distort total cost of computation
  - Do all start-up ops prior to starting actual test
  - Long test runs to amortize start-up effects
  - Measure and subtract start-up costs
- System performance may degrade with age
  - Reestablish base condition for each test

# Measurement Artifacts

- Costs of instrumentation code
  - Additional calls, instructions, cache misses
  - Additional memory consumption and paging
- Costs of logging results
  - May dwarf the costs of instrumentation
  - Increased disk load/latency may slow everything
- ➢ Minimize frequency and costs of measuring
  - Don’t measure everything always
  - Counters/accumulators instead of individual records
  - In-memory circular buffer, reduce before writing to files
  - Probabilistic methods that don’t execute on each occurrence

# Measurement Tools

- Execution profiling
- Event logs
- End-to-end testing

# Execution Profiling

- Automated measurement tools
  - Compiler options for routine call counting
    - One counter per routine, incremented on entry
  - Statistical execution sampling
    - Timer interrupts execution at regular intervals
    - Increment a counter in table based on PC value
    - May have configurable time/space granularity
  - Tools to extract data and prepare reports
    - Number of calls, time per call, percentage of time
- Very useful in identifying the bottlenecks

# Time Stamped Event Logs

- Application instrumentation technique
- Create a log buffer and routine
  - Call log routine for all interesting events
  - Routine stores time and event in a buffer
    - Requires a cheap, very high resolution timer
- Extract buffer, archive, mine the data
  - Time required for particular operations
  - Frequency of operations
  - Combinations of operations
  - Also useful for post-mortem analysis

# Time Stamping

**Dump of simple trace log**

```
datetime       event          sub-type
----------  ------------      ------------      -------------
05/11/06 09:02:31.207408  packet_rcv     0x20749329
05/11/06 09:02:31.209301  packet_route 0x20749329
05/11/06 09:02:31.305208  wakeup         0x4D8C2042
05/11/06 09:02:31.401106  read_packet  0x033C2DA0
05/11/06 09:02:31.401223  read_packet  0x033C2DA0
05/11/06 09:02:31.402110  sleep          0x4D8C2042
05/11/06 09:02:31.614209  interrupt      0x00000003
05/11/06 09:02:31.614209  dispatch       0x1B0324C0
05/11/06 09:02:31.614210  intr_return    0x00000003
05/11/06 09:02:31.652303  check_queue 0x2D3F2040
05/11/06 09:02:31.652306  packet_rcv     0x20749329
```

# End-to-End Testing

- Client-side throughput/latency measurements
  - Elapsed time for X operations of type Y
  - Instrumented clients to collect detailed timings
- Strengths
  - Easy tests to run, easy data to analyze
  - Results reflect client experienced performance
- Weaknesses
  - No information about why it took that long
  - No information about resources consumed

# A Performance Measurement Example

- The Conquest file system
  - A research system built by one of my students
- Using persistent RAM to store many files
  - Which allowed him to get rid of a lot of OS code related to disk drives
- Stored some files on disk
  - Which we won't worry about here
- Expectation was better performance than disk-based file systems

# How Did We Measure Conquest?

- What were the metrics?
- What were the factors?
- What was the workload?
- What were the results?

# Choosing the Metrics

- Core claim was better speed
- So metrics should be speed-related
- Speeding up overall file system operations was the goal
  – Not speeding up an isolated operation
- So we needed metrics capturing that
- We used several “operations per second” metrics
  – Reads, writes, creates, also bandwidth

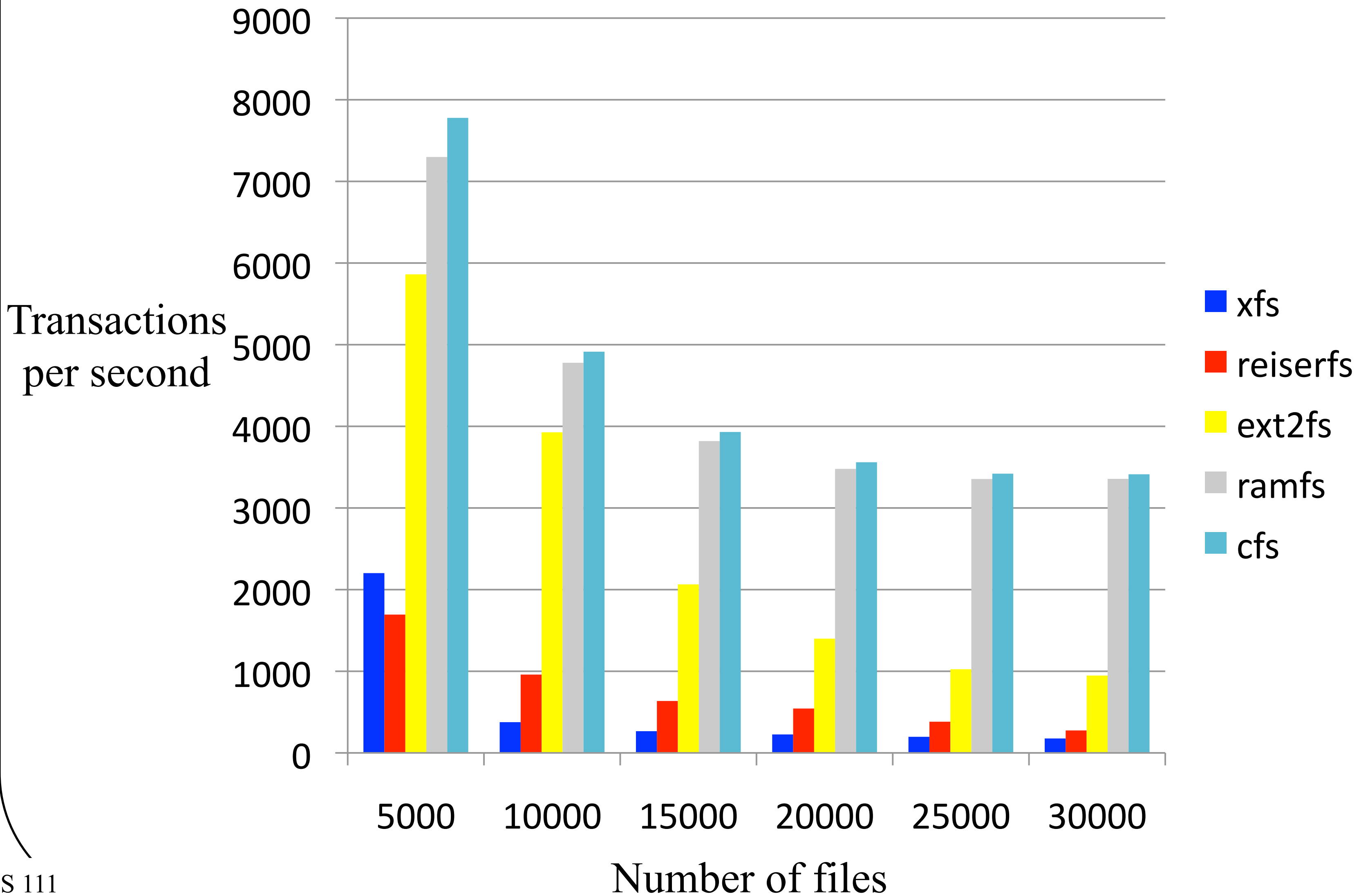
# Choosing the Factors

- We were claiming better performance than other file systems
- So one factor was which file system we tested
- We also wanted to show scaling effects
  - It didn’t perform well just for tiny systems
- So another factor chosen was number of files in the file system

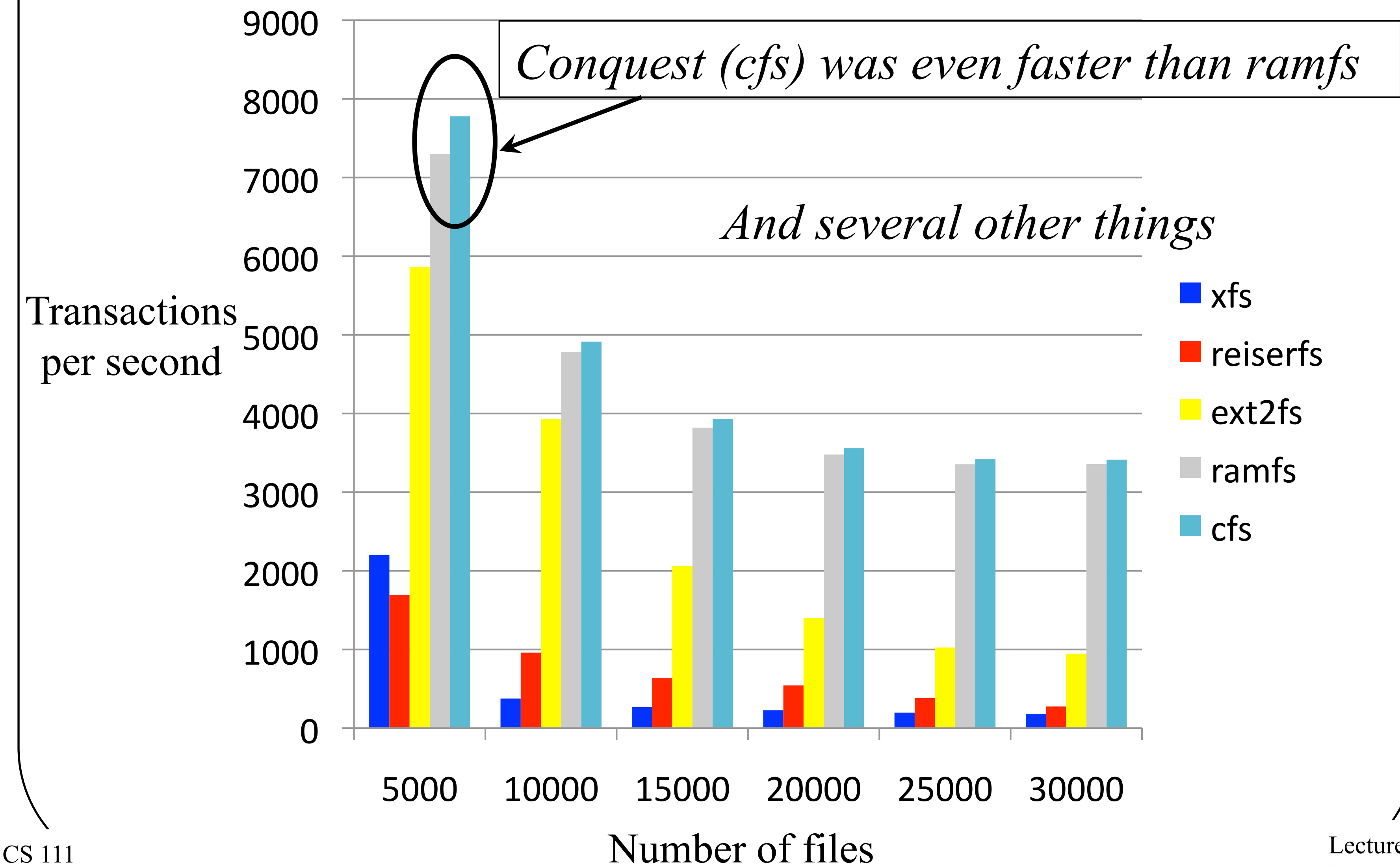
# Choosing the Workload

- File systems are traditionally tested against standard benchmarks
- We tested against several of those
- One benchmark we used is called Postmark
- Postmark performs various "transactions" related to file operations
- The metric we'll show is Postmark transactions per second

# One Set of Results



# Which Showed What?



# A Couple of Words on Presentation

- Always consider these questions:

1. To whom am I speaking?
   - What they know and not know?
   - What are they prepared to absorb, and what not?
2. Why are they listening to me?
   - How might this help them achieve their goals?
   - How might this address their concerns?
3. What do I want them to leave with?
   - What conclusions do I want them to draw?
   - What actions do I want them to take?

# Performance Presentation

- Highlight the key results
  - Answers to the basic questions
  - Identified problems, risks and opportunities
- Why should they believe these results
  - Methodology employed, relation to other results
  - Back-up details
- Not just numbers, but explanations
  - How do we now better understand the system
  - How does this affect our plans and intentions