Understanding Performance in Operating Systems
CS239
Advanced Operating Systems
April 28, 1997

Outline

- Importance of operating systems performance
- Major issues in understanding operating systems performance
- Issues in experiment design

Importance of OS Performance

Performance is almost always a key issue in operating systems
- File system research
- OS tools for multimedia
- Practically any OS area

Since everyone uses the OS (sometimes heavily), everyone is impacted by its performance

A solution that doesn't perform well isn't a solution at all

Importance of Understanding OS Performance

Great, so we work on improving OS performance

How do we tell if we succeeded?

<u>Successful research must prove its performance characteristics to a skeptical community</u>

So What?

Proper performance evaluation is difficult

Knowing what to study is tricky

Performance evaluations take a lot of careful work

Understanding the results is hard

Presenting them effectively is challenging

For Example,

An idea - save power from a portable computer's battery by using its wireless card to execute tasks remotely

Maybe that's a good idea, maybe it isn't

How do we tell?

Performance experiments to validate concept

But What Experiments?

- What tasks should we check?
- What should be the conditions of the portable computer?
- What should be the conditions of the network?
- What should be the conditions of the server?
- How do I tell if my result is statistically valid?

Techniques for Understanding OS Performance

- Analytic modeling
- Simulation
- Measurement

Which technique is right for a given situation?

Analytic Modeling

+ Sometimes relatively quick
+ Within limitations of model, testing alternatives usually easy
– Mathematical tractability may require simplifications
– Not everything models well
– Question of validity of model

Simulation

+ Great flexibility
+ Can capture an arbitrary level of detail
– Often a tremendous amount of work to write and run
– Testing a new alternative often requires repeating a lot of work
– Question of validity of simulation

Experimentation

+ Lesser problems of validity
+ Sometimes easy to get started
– Can be very labor-intensive
– Often hard to perform measurement
– Sometimes hard to separate out effects you want to study
– Sometimes impossible to generate cases you need to study

Elements of Performance Evaluation

- Performance metrics
- Workloads
- Proper measurement technique
- Proper statistical techniques
- Minimization of effort
- Proper data presentation techniques

Performance Metrics

The criteria used to evaluate the performance of a system

E.g., response time, cache hit ratio, bandwidth delivered, etc.

Choosing the proper metrics is key to a real understanding of system performance

Workloads

The requests users make on a system

If you don't evaluate with a proper workload, you aren't measuring what real users will experience

Typical workloads -

- stream of file system requests
- set of jobs performed by users
- list of URLs submitted to a Web server

Proper Performance Measurement Techniques

You need at least two components to measure performance

1. A load generator

   To apply a workload to the system

2. A monitor

   To find out what happened

Proper Statistical Techniques

Computer performance measurements generally not purely deterministic

Most performance evaluations weigh the effects of different alternatives

How to separate meaningless variations from vital data in measurements?

Requires proper statistical techniques

Minimizing Your Work

Unless you design carefully, you'll measure a lot more than you need to

A careful design can save you from doing lots of measurements

Should identify critical factors

And determine the smallest number of experiments that gives a sufficiently accurate answer

Proper Data Presentation Techniques

You've got pertinent, statistically accurate data that describes your system

Now what?

How to present it -

- honestly
- clearly
- convincingly

Why Is Performance Analysis Difficult?

Because it's an art - it's not mechanical

You can't just apply a handful of principles and expect good results

You've got to understand your system

You've got to select your measurement techniques and tools properly

You've got to be careful and honest

Some Common Mistakes in Performance Evaluation

- No goals
- Biased goals
- Unsystematic approach
- Analysis without understanding
- Incorrect performance metrics
- Unrepresentative workload
- Wrong evaluation technique

More Common Performance Evaluation Mistakes

- Overlooking important parameters
- Ignoring significant factors
- Inappropriate experiment design
- No analysis
- Erroneous analysis
- No sensitivity analysis

Yet More Common Mistakes

- Ignoring input errors
- Improper treatment of outliers
- Assuming static systems
- Ignoring variability
- Too complex analysis
- Improper presentation of results
- Ignoring social aspects
- Omitting assumptions/limitations

Choosing Proper Performance Metrics

Three types of common metrics:

- Time (responsiveness)
- Processing rate (productivity)
- Resource consumption (utilization)

Can also measure various error parameters

Response Time

How quickly does system produce results?

Critical for applications such as:

- Time sharing/interactive systems
- Real-time systems
- Parallel computing

Processing Rate

How much work is done per unit time?

Important for:

- Determining feasibility of hardware
- Comparing different configurations
- Multimedia

CS 239, Spring 1997

Lecture 9
Page 25

Resource Consumption

How much does the work cost?

Used in:

- Capacity planning
- Identifying bottlenecks

Also helps to identify the “next” bottleneck

CS 239, Spring 1997

Lecture 9
Page 26

Typical Error Metrics

- Successful service (speed)
- Incorrect service (reliability)
- No service (availability)

CS 239, Spring 1997

Lecture 9
Page 27

Characterizing Metrics

Usually necessary to summarize

Sometimes means are enough

Variability is usually critical

CS 239, Spring 1997

Lecture 9
Page 28

Choosing What to Measure

Pick metrics based on:

- Completeness
- (Non-)redundancy
- Variability

CS 239, Spring 1997

Lecture 9
Page 29

Designing Workloads

- What is a workload?
- Synthetic workloads
- Real-World benchmarks
- Application benchmarks
- “Standard” benchmarks
- Exercisers and drivers

CS 239, Spring 1997

Lecture 9
Page 30

What is a Workload?

A *workload* is anything a computer is asked to do

Test workload: any workload used to analyze performance

Real workload: any workload observed during normal operations

Synthetic workload: any workload created for controlled testing

Real Workloads

+ They represent reality
– Uncontrolled
  • Can't be repeated
  • Can't be described simply
  • Difficult to analyze

Nevertheless, often useful for "final analysis" papers

Synthetic Workloads

+ Controllable
+ Repeatable
+ Portable to other systems
+ Easily modified
– Can never be sure real world will be the same

What Are Synthetic Workloads?

Complete programs designed specifically for measurement

May do real or "fake" work

May be adjustable (parameterized)

Two major classes:
– Benchmarks
– Exercisers

Real-World Benchmarks

• Pick a representative application and sample data
• Run it on system to be tested

Modified Andrew Benchmark, MAB, is a real-world benchmark

+ Easy to do, accurate for that sample application and data
– Doesn't consider other applications and data

Application Benchmarks

Variation on real-world benchmarks

Choose most important subset of functions

Write benchmark to test those functions

+ Tests what computer will be used for
– Need to be sure it captures all important characteristics

“Standard” Benchmarks

Often need to compare general-purpose systems for general-purpose use

Should I buy a Compaq or a Dell PC?

Tougher: Mac or PC?

Need an easy, comprehensive answer

People writing articles often need to compare tens of machines

“Standard” Benchmarks (cont’d)

Often need comparisons over time

How much faster is this year’s Pentium Pro than last year’s Pentium?

Writing new benchmark undesirable

Could be buggy or not representative

Want to compare many people’s results

Exercisers and Drivers

For I/O, network, non-CPU measurements

Generate a workload, feed to internal or external measured system

I/O on local OS

Network

Sometimes uses dedicated system, interface hardware

Advantages and Disadvantages of Exercisers

+ Easy to develop, port
+ Incorporates measurement
+ Easy to parameterize, adjust
– High cost if external
– Often too small compared to real workloads

Workload Selection

- Services Exercised
- Level of Detail
- Representativeness
- Timeliness
- Other Considerations

Services Exercised

What services does system actually use?

Speeding up response to keystrokes won’t help a file server

What metrics measure these services?

Completeness

Computer systems are complex

Effect of interactions hard to predict

So must be sure to test *entire* system

Important to understand balance between components

Level of Detail

Detail trades off accuracy vs. cost

Highest detail is complete trace

Lowest is one request, usually most the common request

Intermediate approach: weight by frequency

Representativeness

Obviously, workload should represent desired application

Again, accuracy and cost trade off

Need to understand whether detail matters

Timeliness

Usage patterns change over time

File size grows to match disk size

If using "old" workloads, must be sure user behavior hasn't changed

Even worse, behavior may change after test, as *result* of installing new system

"Latent demand" phenomenon

Other Considerations

- Loading levels
  - Full capacity
  - Beyond capacity
  - Actual usage
- Repeatability of workload

Monitors

A monitor is a tool used to observe system activity

Proper use of monitors is key to performance analysis

Also useful for other system observation purposes

Event-Driven Vs. Sampling Monitors

- Event-driven monitors notice every time a particular type of event occurs
  - Ideal for rare events
  - Require low per-invocation overheads
- Sampling monitors check the state of the system periodically
  - Good for frequent events
  - Can afford higher overheads

CS 239, Spring 1997 — Lecture 9 Page 49

On-Line Vs. Batch Monitors

- On-line monitors can display their information continuously
  - Or, at least, frequently
- Batch monitors save it for later
  - Usually using separate analysis procedures

CS 239, Spring 1997 — Lecture 9 Page 50

Issues in Monitor Design

- Activation mechanism
- Buffer issues
- Data compression/analysis
- Priority issues
- Abnormal events monitoring

CS 239, Spring 1997 — Lecture 9 Page 51

Activation Mechanism

When do you collect the data?

Several possibilities:

- When an interesting event occurs, trap to data collection routine
- Analyze every step taken by system
- Go to data collection routine when timer expires

CS 239, Spring 1997 — Lecture 9 Page 52

Buffer Issues

Buffer size should be big enough to avoid frequent disk writes

But small enough to make disk writes cheap

Use at least two buffers, typically

One to fill up, one to record

Must think about buffer overflow

CS 239, Spring 1997 — Lecture 9 Page 53

Data Compression or Analysis

Data can be literally compressed

Or can be reduced to a summary form

Both methods save space

But at the cost of extra overhead

Sometimes can use idle time for this

But idle time might be better spent dumping data to disk

CS 239, Spring 1997 — Lecture 9 Page 54

Priority of Monitor

How high a priority should the monitor's operations have?

Again, trading off performance impact against timely and complete data gathering

Not always a simple question

Monitoring Abnormal Events

Often, knowing about failures and errors more important than knowing about normal operation

Sometimes requires special attention

- System may not be operating very well at the time of the failure

Monitoring Distributed Systems

Monitoring a distributed system is not dissimilar to designing a distributed system

Must deal with:

- Distributed state
- Unsynchronized clocks
- Partial failures

Tools For Software Measurement

- Code instrumentation
- Tracing packages
- System-provided metrics and utilities
- Profiling

Code Instrumentation

Adding monitoring code to the system under study

\+ Usually most direct way to gather data

\+ Complete flexibility

\+ Strong control over costs of monitoring

– Requires access to the source

– Requires strong knowledge of code

– Strong potential to affect performance

Typical Types of Instrumentation

- Counters
  - \+ Cheap and fast
  - – But low level of detail
- Logs
  - \+ More detail
  - – But more costly
  - – Require occasional dumping or digesting
- Timers

Tracing Packages

Allow dynamic monitoring of code that doesn't have built-in monitors
Akin to debuggers
+ Allows arbitrary insertion of code
+ No recompilation required
+ Tremendous flexibility
+ No overhead when you're not using it
– Somewhat higher overheads
– Effective use requires access to source

System-Provided Metrics and Utilities

Many operating systems provide users access to some metrics

Most operating systems also keep some form of accounting logs

Lots of information can be gathered this way

Profiling

Many compilers provide easy facilities for profiling code

+ Easy to use

+ Low impact on system

– Requires recompilation

– Provides very limited information

Introduction To Experiment Design

- You know your metrics
- You know your factors
- You've got your instrumentation and test loads
- Now what?

Goals in Experiment Design

- Obtain maximum information with minimum work
  - Typically meaning minimum number of experiments
- More experiments aren't better if <u>you</u> have to perform them
- Well-designed experiments are also easier to analyze

Experimental Replications

A run of the experiment with a particular set of levels and other inputs is a replication

Often, you need to do multiple replications with a single set of levels and other inputs

For statistical validation

Interacting Factors

Some factors have effects completely independent of each other

– Double the factor's level, halve the response, regardless of other factors

But the effects of some factors depends on the values of other factors

– *Interacting* factors

Presence of interacting factors complicates experimental design

Basic Problem in Designing Experiments

Your chosen factors may or may not interact

How can you design an experiment that captures the full range of the levels?

With minimum amount of work

Common Mistakes in Experimentation

- Ignoring experimental error
- Uncontrolled parameters
- Not isolating effects of different factors
- One-factor-at-a-time experiment designs
- Interactions ignored
- Designs require too many experiments