Workloads for Experiments
CS 239
Experimental Methodologies for System Software
Peter Reiher
April 19, 2007

Introduction

- Introduction to workloads
- Workload selection
- Types of workloads
- Characterizing a workload

Introduction to Workloads

- What is a workload?
- Real workloads
- Synthetic workloads

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 observed during normal operations
- *Synthetic workload*: workload created for controlled testing

Workloads and Systems Experiments

- Systems do something
- Point of experiments is to find out how well
- The workload is what the system does
- To determine true systems performance, must apply good workload

Desirable Properties in Test Workloads

- Realistic
- Representative of whole range of real workloads
- Controllable
- Reproducible
- Tractable

Some Problems in Experimental Workloads

- What is the real workload you'd like to match?
- How can you accurately mirror that workload?
- How do you handle wide ranges of workload variations?
- How do you handle predicted workloads?
- How do you scale workloads to experimental conditions?

Why Not Use the Real Workload?

- Why not just test with reality?
- Not always possible
- Generally not reproducible
- Definitely not controllable
- Sometimes legal issues
- Still, occasionally possible
  - Usually after other methods show general feasibility and properties

Real Workloads

- Advantage is they represent reality
- Disadvantage is they're uncontrolled
  - Can't be repeated
  - Can't be described simply
  - Difficult to analyze
- Nevertheless, often useful for "final analysis" papers
  - E.g., "We ran Ficus and it works well"

Synthetic Workloads

- Advantages:
  - Controllable
  - Repeatable
  - Sometimes standardizable
  - Portable to other systems
  - Easily modified
- Disadvantage: can never be sure real world will match them

Workload Selection

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

Services Exercised

- What services does system actually use?
  - Faster CPU won't speed up big "cp"
  - Network performance useless for matrix work
- What metrics measure these services?
  - MIPS for CPU speed
  - Bandwidth for network, I/O
  - TPS for transaction processing

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
  - I.e., don't use CPU workload to evaluate I/O-bound application

Component Testing

- Sometimes only individual components are compared
  - Would a new CPU speed up our system?
  - Would IPV6 affect Web server performance?
- But component may not be directly related to performance
  - Analysis of Variation (ANOVA) test can help here

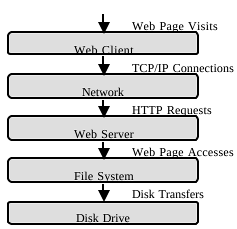
Service Testing

- May be possible to isolate interfaces to just one component
  - E.g., instruction mix for CPU
- Consider *services* provided and used by that component
- System often has *layers* of services
  - Can cut at any point and insert workload

Characterizing a Service

- Identify *service* provided by major subsystem
- List *factors* affecting performance
- List *metrics* that quantify demands and performance
- Identify *workload* provided to that service

Example: Web Server



Web Client Analysis

- Services: visit page, follow hyperlink, display information
- Factors: page size, number of links, fonts required, embedded graphics, sound
- Metrics: response time (both definitions)
- Workload: a list of pages to be visited and links to be followed

Network Analysis

- Services: connect to server, transmit request, transfer data
- Factors: bandwidth, latency, protocol used
- Metrics: connection setup time, response latency, achieved bandwidth
- Workload: a series of connections to one or more servers, with data transfer

Web Server Analysis

- Services: accept and validate connection, fetch HTTP data
- Factors: Network performance, CPU speed, system load, disk subsystem performance
- Metrics: response time, connections served
- Workload: a stream of incoming HTTP connections and requests

File System Analysis

- Services: open file, read file (writing doesn’t matter for Web server)
- Factors: disk drive characteristics, file system software, cache size, partition size
- Metrics: response time, transfer rate
- Workload: a series of file-transfer requests

Disk Drive Analysis

- Services: read sector, write sector
- Factors: seek time, transfer rate
- Metrics: response time
- Workload: a stream of read/write requests

Level of Detail

- Detail trades off accuracy vs. cost
- Highest detail is complete trace
- Lowest is one request, usually most common
- Intermediate approach: weight by frequency
- We will return to this when we discuss *workload characterization*

Representivity

- Obviously, workload should represent desired application
  - Arrival rate of requests
  - Resource demands of each request
  - Resource usage profile of workload over time
- Again, accuracy and cost trade off
- Need to understand whether detail matters

Timeliness

- Use patterns change over time
  - File size grows to match disk size
  - Web pages grow to match network bandwidth
- 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

Types of Workloads

- Microbenchmarks
- Benchmarks
- Traces
- Generators and exercisers
- Live workloads

Microbenchmarks

- A test of the performance of a very low level operation
  - CPU arithmetic operation
  - Sending one message
  - Allocating one buffer

Purpose of Microbenchmark

- Sometimes that's precisely what you want to measure
  - E.g., measuring an improvement in memory allocator
- Sometimes it describes key property of overall system
  - Message send cost is crucial in distributed system

Advantages of Microbenchmarks

+ Generally simple to test
+ Pretty easy to understand
+ Limited amount of work to test
+ Can reveal most important elements of system behavior
+ Sometimes exactly what you are looking for

Disadvantages of Microbenchmarks

- Doesn't show interactions
- Often not relevant to the real issue
- Tend not to be considered in varying circumstances
  - Usually "best case"
- May offer little insight on how to improve system

Using Microbenchmarks

- Usually suitable for simple situations
  - Or when minimum cost is of interest
- Generally don't fully describe real systems
- Microbenchmarks are almost never enough
  - So do them only when they provide insight
  - Be suspicious of whole systems studies that only report microbenchmarks

CS 239, Spring 2007 Lecture 5 Page 31

Benchmarks

- A standardized artificial workload
- Generally designed to test specific type of system
  - File system, database, web server, etc.
- Usually intended for wide use
  - Which allows system comparisons
    - In principle . . .

CS 239, Spring 2007 Lecture 5 Page 32

Where Do Benchmarks Come From?

- Sometimes from standards bodies
  - Or industry consortia
  - Occasionally government fiat
- Sometimes proposed by leading researchers
  - Either picked up by others or not

CS 239, Spring 2007 Lecture 5 Page 33

Some Types of Benchmarks

- File system benchmarks
- Processor performance benchmarks
- Database benchmarks

CS 239, Spring 2007 Lecture 5 Page 34

How Do You Build a Benchmark?

- Pick a representative real-world application
- Pick 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 data
- Fails to consider other applications, data
  - So just how representative was your choice?

CS 239, Spring 2007 Lecture 5 Page 35

Popular Benchmarks

- Sieve
- Whetstone
- Debit/credit
- SPEC
- Modified Andrew Benchmark

CS 239, Spring 2007 Lecture 5 Page 36

Debit/Credit Benchmark

- Developed for transaction processing environments
  - CPU processing usually trivial, but demanding I/O, scheduling
- Models real TPS workloads synthetically
- Modern version is TPC benchmark
  - Comes in several flavors

SPEC Suite

- Result of multi-manufacturer consortium
- Several different benchmarks
  - For CPU, graphics, mail, web servers, etc.
- Addresses flaws in existing benchmarks
- Workloads derived from real applications
- Still supported, with new CPU version released in 2006

Modified Andrew Benchmark

- Used in research to compare file system, operating system designs
- Based on software engineering workload
- Exercises copying, compiling, linking
- Probably ill-designed and badly outdated, but still widely used

Advantages of Benchmarks

+ Standardized
+ Usually widely available
+ Very well understood
+ Provides a defense against "why didn't you test X"?
  ? Even if you didn't bother to think much

Disadvantages of Benchmarks

- Not customizable
  - So often not relevant to your system
- Often outdated
- Tend not to scale well
- Can be hard to interpret what they really mean

Traces

- A seductive idea
- Record live activity in a real system
- Play it back into the system under test
- Since it's real, clearly it should tell you the real performance of your system
  - Right?

Traces in More Detail

- Watch live system
  - Of same type as system under test
  - Under the conditions you consider characteristic and important
- At suitable level of detail, record each event
- "Play back" trace into your test system
- If you measured performance while gathering trace, can instantly compare

Some Issues for Traces

- Level of detail
- Representivity of trace
- Length of trace
- Privacy issues
- Gathering data might perturb behavior
- How to properly rerun it

Advantages of Traces

\+ Based on real world phenomena

\+ Captures many nitty-gritty details of real use

\+ Can be reused on many systems

\+ Standardizable

Disadvantages of Traces

- Often hard to gather
- Replay can easily become invalid
  - If behavior of system changes dynamics
    - E.g., dropping a packet
- Very specific to particular situations
  - And can quickly be outdated
- Anonymization may wash out important details
- Many companies regard their traces as valuable property
- Often hard to scale properly

Some Trace Examples

- NLANR packet header traces
  - Collection of Internet packet header traces
- U. of Oregon Routeviews traces
  - Of BGP routing updates
- File system traces
  - Seer traces gathered at UCLA
- Web traces
  - Many are quite old

Generators and Exercisers

- Create program that generates the workload
- Run it against the system under test
- Measure performance while workload is being generated

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 of Exercisers

+ Easy to develop, port
+ Incorporate measurement
+ Easy to parameterize, adjust
+ Good for repetitive testing of component to see if it fails

Disadvantages of Exercisers

- High cost if external
- Often too small compared to real workloads
  - Thus not representative
  - May use caches "incorrectly"
- Internal exercisers often don't have real CPU activity
  - Affects overlap of CPU and I/O
- Synchronization effects caused by loops

Generators

- A program that generates a workload
- Usually intended to match some specific real-world behavior
- Hook up the generator to the testing framework and turn it on
- Measure the result, and there you are

Issues in Creating Generators

- Level of detail
- Breadth of applicability
- Scalability
- Fidelity
- Reproducibility
- Efficiency

Issues in Using Generators

- Choosing the right one
- Properly setting its parameters
- Interaction with other elements of testing framework
- Scaling

Advantages of Generators

+ Can be quite realistic
+ Can have reproducible results
+ Usually highly parameterizable
+ More scaleable than some alternatives
+ Easy to do stress tests

Disadvantages of Generators

- Hard to build good ones
  - Commercial ones can be expensive
- Might be hard to find right parameters
- Not always well validated against your reality
- Might require extra hardware

An Example Generator - Harpoon

- A network traffic generator
  - At flow level
  - Intended to provide workloads of realistic network traffic
- Uses network traces to determine type of network traffic to mimic
  - Gathered with other tools
- Runs on dedicated machine in test network

How Harpoon Works

- Runs lots of TCP and UDP sessions to match traffic pattern
  - File transfers for TCP
  - Pings for UDP
  - Sets up response software on the client machine
- Can parameterize in many ways

Uses of Harpoon

- To test message handling on a client
- To test network hardware and protocols in between clients and servers
- To test boxes that handle traffic in between

Live Workloads

- Hook system under test up to real world traffic
  - Either in place of existing system
  - Or mirror requests from existing system
- Measure how it performs

Advantages of Live Workloads

+ Very realistic
+ Pretty representative of your real workload
+ Tend to exercise some “uncommon” cases
  + The ones that you didn’t think of, but that actually occur

Disadvantages of Live Workloads

- Not reproducible
- Unless mirrored, you can screw up real work
- If mirrored, you lose fidelity
- Can’t stress test
  - Beyond what really happens

Live Tests and Human Experiments

- Not all live tests involve humans
- But if they do, you typically need to be careful
- Most institutions have rules about human tests
- US government grants do, too
- You could get in a lot of trouble if you’re not careful

Workload Characterization

- How do you characterize your workload?
- Important for:
  – Creating a generator
  – Understanding system behavior
- Basically, you need a model of the workload
- How do you get one?

Workload Components

- A *workload component* is a single “service request”
- Selecting them vital to the model
- Most important is that components be external: at the interface of the SUT
- Components should be homogeneous
- Should characterize activities of interest to the study

Workload Parameters

- *Parameters* are quantities that characterize the workload
- Select parameters that depend only on workload (not on SUT)
- Prefer controllable parameters
- Omit parameters that have no effect on system, even if important in real world

Types of Models

- Simple models
- Models based on distributions
- Markov models
- Code-based models
- Clustering and models

Simple Models

- Use average value of each parameter
  - Not necessarily arithmetic mean
- Good for uniform distributions or gross studies
- Can augment with simple indices of dispersion
  - Using some method to sample parameter values based on those

Models Based on Distributions

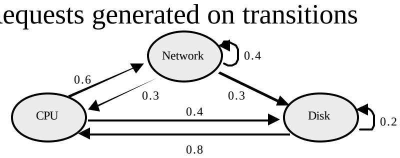
- Determine full distribution of parameter values
  - Perhaps via histograms
  - Jain (Chapter 29) discusses many distributions
- Select test values based on that distribution
- If multiple parameters, need to account for interactions

Markov Models

- Sometimes, distribution of requests isn't enough
  - Sequence affects performance
- Example: Modeling web browsing
  - Users ask for a web page, wait for response
  - Then examine page, then ask for another
  - Single user shouldn't generate second request while waiting for first

Introduction to Markov Models

- Represent model as state diagram
- Transitions between states are probabilistic
- Requests generated on transitions



Creating a Markov Model

- Observe long string of activity
- Use matrix to count pairs of states
- Normalize rows to sum to 1.0

	CPU	Network	Disk
CPU		0.6	0.4
Network	0.3	0.4	0.3
Disk	0.8		0.2

Example Markov Model

- Reference string of opens, reads, closes:
  ORORRCOORCRRRRCC
- Pairwise frequency matrix:

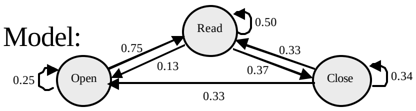
	Open	Read	Close	Sum
Open	1	3		4
Read	1	4	3	8
Close	1	1	1	3

Markov Model for I/O String

- Divide each row by its sum to get transition matrix:

	Open	Read	Close
Open	0.25	0.75	
Read	0.13	0.50	0.37
Close	0.33	0.33	0.34

- Model:



Code-Based Modeling

- Use simplified version of actual code to model system
- Or of well-defined protocol
- E.g., modeling TCP behavior
  - Average time between sends is bogus
  - Pure distribution is bogus
  - Simple Markov model might not be enough
  - But you can build relatively simple generator that "follows the rules"
- How much can you simplify . . .

Clustering

- Often useful to break workload into categories
- "Canonical example" of each category can be used to represent all samples
- If many samples, generating categories is difficult
- Clustering algorithms can solve this problem

Steps in Clustering

- Select sample
- Choose and transform parameters
- Drop outliers
- Scale observations
- Choose distance measure
- Do clustering
- Use results to adjust parameters, repeat
- Choose representative components

Interpreting Clusters

- Art, not science
- Drop small clusters (if little impact on performance)
- Try to find meaningful characterizations
- Choose representative components
  - Number proportional to cluster size or to total resource demands

Drawbacks of Clustering

- Clustering is basically an AI problem
- Humans will often see patterns where computer sees none
- Result is extremely sensitive to:
  - Choice of algorithm
  - Parameters of algorithm
  - Minor variations in points clustered
- Results may not have functional meaning