CS 147:
Computer Systems Performance Analysis
Test Loads

Overview

Overview

Designing Test Loads
Load Types
Applying Loads

Common Benchmarking Mistakes

Test Load Design

- Most experiments require applying test loads to system
- General characteristics of test loads already discussed
- How do we design test loads?

Test Load Design
- Most experiments require applying test loads to system
- General characteristics of test loads already discussed
- How do we design test loads?

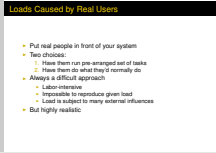
Types of Test Loads

- Real users
- Traces
- Load-generation programs

Types of Test Loads
- Real users
- Traces
- Load-generation programs

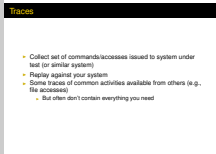
Loads Caused by Real Users

- Put real people in front of your system
- Two choices:
  1. Have them run pre-arranged set of tasks
  2. Have them do what they'd normally do
- Always a difficult approach
  - Labor-intensive
  - Impossible to reproduce given load
  - Load is subject to many external influences
- But highly realistic



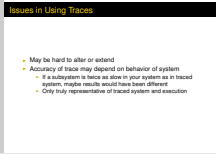
Traces

- Collect set of commands/accesses issued to system under test (or similar system)
- Replay against your system
- Some traces of common activities available from others (e.g., file accesses)
  - But often don't contain everything you need



Issues in Using Traces

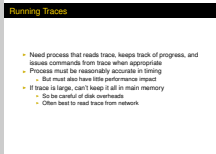
- May be hard to alter or extend
- Accuracy of trace may depend on behavior of system
  - If a subsystem is twice as slow in your system as in traced system, maybe results would have been different
  - Only truly representative of traced system and execution



E.g., processes might run at different rates depending on I/O vs. CPU mix

Running Traces

- Need process that reads trace, keeps track of progress, and issues commands from trace when appropriate
- Process must be reasonably accurate in timing
  - But must also have little performance impact
- If trace is large, can't keep it all in main memory
  - So be careful of disk overheads
  - Often best to read trace from network



Load-Generation Programs

- Create model for load you want to apply
- Write program implementing that model
- Program issues commands & requests synthesized from model
  - E.g., if model says open file, program builds appropriate `open()` command

Load-Generation Programs
- Create model for load you want to apply
- Write program implementing that model
- Program issues commands & requests synthesized from model
  - E.g., if model says open file, program builds appropriate `open()` command

Building the Model

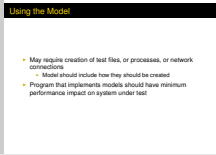
- Tradeoff between ease of creation and use of model vs. its accuracy
- Base model on everything you can find out about the real system behavior
  - Which may include examining traces
- Consider whether model can be memoryless, or requires keeping track of what's already happened (Markov)

Building the Model

- Tradeoff between ease of creation and use of model vs. its accuracy
- Base model on everything you can find out about the real system behavior
  - Which may include examining traces
- Consider whether model can be memoryless, or requires keeping track of what's already happened (Markov)

Using the Model

- May require creation of test files, or processes, or network connections
  - Model should include how they should be created
- Program that implements models should have minimum performance impact on system under test



Applying Test Loads

- Most experiments will need multiple repetitions
  - Details covered later in course
- Results most accurate if each repetition runs in identical conditions

⇒ Test software should work hard to duplicate conditions on each run

  - Requires thorough understanding of system

Applying Test Loads
- Most experiments will need multiple repetitions
  - Details covered later in course
- Results most accurate if each repetition runs in identical conditions

⇒ Test software should work hard to duplicate conditions on each run
  - Requires thorough understanding of system

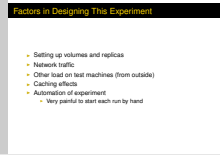
Example of Applying Test Loads

- Using Ficus experiments discussed earlier, want performance impact of update propagation for multiple replicas
- Test load is set of benchmarks involving file access & other activities
- Must apply test load for varying numbers of replicas

Example of Applying Test Loads
- Using Ficus experiments discussed earlier, want performance impact of update propagation for multiple replicas
- Test load is set of benchmarks involving file access & other activities
- Must apply test load for varying numbers of replicas

Factors in Designing This Experiment

- Setting up volumes and replicas
- Network traffic
- Other load on test machines (from outside)
- Caching effects
- Automation of experiment
  - Very painful to start each run by hand



Experiment Setup

- Need volumes to read and write, and replicas of each volume on various machines
- Must be certain that setup completes **before** we start running experiment

Experiment Setup
- Need volumes to read and write, and replicas of each volume on various machines
- Must be certain that setup completes **before** we start running experiment

Network Traffic Issues

- If experiment is distributed (like ours), how is it affected by other traffic on network?
- Is traffic seen on network used in test similar to traffic expected on network you would actually use?
- If not, do you need to run on isolated network? And/or generate appropriate background network load?

Network Traffic Issues
- If experiment is distributed (like ours), how is it affected by other traffic on network?
- Is traffic seen on network used in test similar to traffic expected on network you would actually use?
- If not, do you need to run on isolated network? And/or generate appropriate background network load?

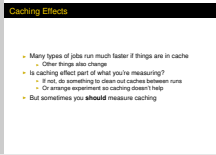
Controlling Other Load

- Generally, want to have as much control as possible over other processes running on test machines
- Ideally, use dedicated machines
- But also be careful about background and periodic jobs
  - In Unix context, check carefully on `cron` and network-related daemons
  - Tough question: use realistic environment or kill all interfering processes?

Controlling Other Load
- Generally, want to have as much control as possible over other processes running on test machines
- Ideally, use dedicated machines
- But also be careful about background and periodic jobs
  - In Unix context, check carefully on cron and network-related daemons
  - Tough question: use realistic environment or kill all interfering processes?

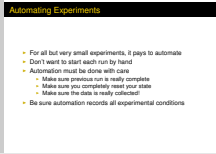
Caching Effects

- Many types of jobs run much faster if things are in cache
  - Other things also change
- Is caching effect part of what you're measuring?
  - If not, do something to clean out caches between runs
  - Or arrange experiment so caching doesn't help
- But sometimes you **should** measure caching



Automating Experiments

- For all but very small experiments, it pays to automate
- Don't want to start each run by hand
- Automation must be done with care
  - Make sure previous run is really complete
  - Make sure you completely reset your state
  - Make sure the data is really collected!
- Be sure automation records all experimental conditions



Common Mistakes in Benchmarking

- Many people have made these
- You will make some of them, too
- But watch for them, so you don't make too many

Common Mistakes in Benchmarking
- Many people have made these
- You will make some of them, too
- But watch for them, so you don't make too many

Only Testing Average Behavior

- Test workload should usually include divergence from average workload
  - Few workloads always remain at their average
  - Behavior at extreme points is often very different
- Particularly bad if **only** average behavior is used

Only Testing Average Behavior
- Test workload should usually include divergence from average workload
  - Few workloads always remain at their average
  - Behavior at extreme points is often very different
- Particularly bad if **only** average behavior is used

Ignoring Skewness of Device Demands

- More generally, not including skewness of any component
  - E.g., distribution of file accesses among set of users
- Leads to unrealistic conclusions about how system behaves

Ignoring Skewness of Device Demands
- More generally, not including skewness of any component
  - E.g., distribution of file accesses among set of users
- Leads to unrealistic conclusions about how system behaves

Loading Levels Controlled Inappropriately

- Not all methods of controlling load are equivalent
- Choose methods that capture effect you are testing for
- Prefer methods allowing more flexibility in control over those allowing less

Loading Levels Controlled Inappropriately
- Not all methods of controlling load are equivalent
- Choose methods that capture effect you are testing for
- Prefer methods allowing more flexibility in control over those allowing less

Caching Effects Ignored

- Caching occurs many places in modern systems
- Performance on given request usually very different depending on cache hit or miss
- Must understand how cache works
- Must design experiment to use it realistically
- Always document whether cache was warm or cold
  - And how warming/cooling was done

Caching Effects Ignored
- Caching occurs many places in modern systems
- Performance on given request usually very different depending on cache hit or miss
- Must understand how cache works
- Must design experiment to use it realistically
- Always document whether cache was warm or cold
  - And how warming/cooling was done

Inappropriate Buffer Sizes

- Slight changes in buffer sizes can greatly affect performance in many systems
- Make sure you match reality

Inappropriate Buffer Sizes
- Slight changes in buffer sizes can greatly affect performance in many systems
- Make sure you match reality

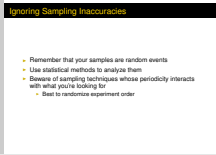
Inappropriate Workload Sizes

- Many test workloads are unrealistically small
- System capacity is ever-growing
- Be sure you actually stress the system

Inappropriate Workload Sizes
- Many test workloads are unrealistically small
- System capacity is ever-growing
- Be sure you actually stress the system

Ignoring Sampling Inaccuracies

- Remember that your samples are random events
- Use statistical methods to analyze them
- Beware of sampling techniques whose periodicity interacts with what you're looking for
  - Best to randomize experiment order



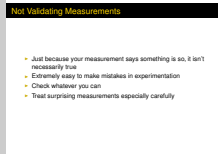
Ignoring Monitoring Overhead

- Primarily important in design phase
  - If possible, must minimize overhead to point where it is not relevant
- But also important to consider it in analysis

Ignoring Monitoring Overhead
- Primarily important in design phase
  - If possible, must minimize overhead to point where it is not relevant
- But also important to consider it in analysis

Not Validating Measurements

- Just because your measurement says something is so, it isn't necessarily true
- Extremely easy to make mistakes in experimentation
- Check whatever you can
- Treat surprising measurements especially carefully



Not Ensuring Constant Initial Conditions

- Repeated runs are only comparable if initial conditions are the same
- Not always easy to undo everything previous run did
  - E.g., same state of disk fragmentation as before
- But do your best
  - And understand where you don't have control in important cases

Not Ensuring Constant Initial Conditions
- Repeated runs are only comparable if initial conditions are the same
- Not always easy to undo everything previous run did
  - E.g., same state of disk fragmentation as before
- But do your best
  - And understand where you don't have control in important cases

Not Measuring Transient Performance

- Many systems behave differently at steady state than at startup (or shutdown)
- That's not always everything we care about
- Understand whether you should care
- If you should, measure transients too
- Not all transients are due to startup/shutdown; be sure you consider those ones too

Not Measuring Transient Performance
- Many systems behave differently at steady state than at startup (or shutdown)
- That's not always everything we care about
- Understand whether you should care
- If you should, measure transients too
- Not all transients are due to startup/shutdown; be sure you consider those ones too

Performance Comparison Using Device Utilizations

- Sometimes this is right thing to do
  - But only if device utilization is metric of interest
- Remember that faster processors will have lower utilization on same load
  - And that's not a bad thing

Performance Comparison Using Device Utilizations
- Sometimes this is right thing to do
  - But only if device utilization is metric of interest
- Remember that faster processors will have lower utilization on same load
  - And that's not a bad thing

Lots of Data, Little Analysis

- **The data isn't the product!**
- **The analysis is!**
- So design experiment to leave time for sufficient analysis
- If things go wrong, alter experiments to still leave analysis time

Lots of Data, Little Analysis
- The data isn't the product!
- The analysis is!
- So design experiment to leave time for sufficient analysis
- If things go wrong, alter experiments to still leave analysis time