Motivation

- “Figures don’t lie, but liars figure” - Mark Twain
- “There are three kinds of untruths; lies, damn lies, and statistics” - Mark Twain
- “How accurate a number is depends on the cost of acquiring it and how important it is” - Deborah Stone
Motivation

• Why is your software so slow?
  – It took five minutes to show the sentence I just typed!
  – It took five minutes to spell check my thousand page book!
  – What do you mean I need a faster video card?
  – Why does your application make everything else so slow?
Overview

• Why is performance important?
• How can we measure performance?
• Once we measure performance, what can we do about it?
• Limitations of performance measurement
• Conclusions
Why is performance important?

• The most obvious non-functional requirements
• This colors what the end user sees and how he uses the application
  – A feature that is too slow is not usable
  – Slow software never appears “good”
• Slow software loses in a competitive market
Why is performance important?

• Performance needs to be understood if the software is going to scale
  – In your lab, the software may work fine for 10 simultaneous users
  – What happens with 1000 users?
  – If your software is going to scale, you need to know more than “it seems good enough”
  – Even if your software is not planning to scale, you are better off knowing about these limitations
Why is performance important?

- Marketing
  - Performance and scalability claims are good marketing material
  - Benchmarking
- Product Validation
  - Does your system react the same under load?
  - Is a validation under light load good enough?
Why is performance important?

• Performance tuning
  – When you receive a performance complaint, how do you track it down?
  – How do you narrow down what to fix?

• Data collection
  – Is the software too slow or the customer’s computer?
  – How can you guide the customer as to what the system requirements are?
Why is Performance Important

1979 GAO Study ($6.8 million)

1995 GAO Study ($35.7 billion)
Why is Performance Important

- Delivered But Never Successfully Used: 47%
- Paid For But Never Delivered: 29%
- Used But Extensively Reworked Or Abandoned: 20%
- Used After Changes: 3%
- Used As Delivered: 2%

1979 GAO Study ($6.8 million)
1995 GAO Study ($35.7 billion)
How can we measure performance?

• Measurement is hard
  – What kind of accuracy can you expect?
    • What factors contribute to the inaccuracy of measurement?
  – How sure can you be of any result?
  – How should you state your result?
  – How much measurement is sufficient?
  – Liars figure
    • How do you know what you need to know?
How can we measure performance?

• Trying to measure is hard
  – What tools do you use?
  – Are all the important factors measurable?
  – What do you measure?
  – How are your measurements biased?
  – What measurements are dependent on others?
  – How do inaccuracies propagate?
  – What is your experimental design?
Know Your Limitations

• What are sources of uncertainty in hardware and software?
  – Is software deterministic?
    • If you model software to be deterministic, are you measurements relevant?
  – Is your measurement accurate?
    • Accuracy of the tool
  – Is the system static?
    • Changing code
    • Changing hardware
    • Changing Environment
Know Your Limitations

• Write a program to measure seconds
  – How precise is it?

• If a simple set of measurements is not precise, how precise is a complex system?
Taking Measurements

- Individual measurements are unreliable
- Statistics
  - Assuming that measurements are independent
    - Take many measurements
    - Standard deviation
    - Variance
    - Confidence intervals
- Over time, measurement of seconds should approximate a bell curve
Taking Measurements

• Guidelines
  – Keep it simple
  – Single measurements are not reliable
    • Experimental error
    • Precision of measurements
  – Vary one parameter at a time
  – Be aware of dependent parameters
  – Verify your numbers
Measuring Software

• Is it useful to do measurement of software through measurements of runtime performance?
• The best performance improvements are done at design-time, not maintenance time!
  – Understand the system performance before you write the software
Measuring Software

• Four parts
  – Performance Goals
  – Workload
  – Software
  – Execution Environment

• Resource and usage estimates
Measuring Software

(Bell, Boehm, Watson, 1972)
Measuring Software

- Arrival Rate: how many requests that arrive per unit of time
- Throughput: how many requests complete (depart) per unit of time
- Utilization: What percentage of unit time a resource is in use
- Service Requirement: How long it takes a resource to handle an operation
- Residence Time: How long a request is in the system before it departs
- Requests in System: How many requests are in the system
Measuring Software

• How do we apply what we know to measuring software performance?
• Know what you want to measure
  – The cost of activity
• Know the component parts of that value
  – I/O costs
  – Processing costs
  – Network costs
Measuring Software

• I/O costs
  – Disk reads and writes
  – Seek times

• Processing Costs
  – Time spent in the CPU

• Network costs
  – Network latency
Measuring Software

• Start with a system model
  – Classify the connections between parts of the system
    • What are the types of costs associated with the links?
  – Determine the number of operations
    • How many I/O operations?
    • How many CPU operations?
    • How many network operations?
    • Estimate!
Measuring Software

• Model the platform
  – How many I/O units can be processed per second?
  – How many network units?
  – How many processing units?

• What happens when more units of work come in that can be processed?
  – Queuing
Measuring Software

• Operations queue when more operations come in then can be processed
  – But how do we determine how many operations come in?

• Think time
  – The time between operations
  – Models the user
    • Time for the user to read and act
  – Number of operations * Think time is the number of operations per second
Measuring Software

• Building your scenario
  – Number of concurrent users
  – Arrival rates
  – Goal

• Start with the most common scenarios
  – Add in the larger scenarios / critical scenarios later
Pareto Principle / Zipf’s Law

• Less than 20% of library books are checked out >80% of the time
• 80% of the processing in a system is in less than 20% of the code
• Focus on the hot-spots
  – You could design a great algorithm that is O(n), but if it is not a hot spot, did that buy you anything?
Measuring Software

• Example, ATM
  – Initiate the session
  – Get and interpret request
  – Authorize transaction
  – Dispense cash
  – Print Receipt
  – Terminate session

• What’s the goal for the entire transaction?
• What are the likely hotspots?
Measuring Software

1. Initiate
2. Get and Interpret Request
3. Authorize
4. Dispense Cash
5. Print Receipt
6. Terminate Session
What can we do about performance?

• The goal is to reduce the queues
  – Queues directly impact performance, they are a measure of the system being used beyond capacity

• What are the strategies to reduce queuing?
What can we do about performance?

• Increase the capacity of the system
  – Throw more hardware at the problem

• What are the effects?
  – i.e. increasing the I/O capacity means that more I/O operations can be handled

• What is the downside?
  – Increased cost of hardware
  – Increased system requirements
  – Another resource could become the bottleneck
What can we do about performance?

• Decrease the number of operations
  – Optimize the system
  – Reduce unnecessary or redundant operations
    • i.e. information that is used often can be pulled from the database once

• What are the effects?
  – A reduction in the number of operations leading to less needed capacity

• What is the downside?
  – Potential complexity increase
  – Optimized code may be more difficult to understand or debug
What can we do about performance?

• Redesign the system
  – Significantly changing the application so that functionality is combined and resource access is reduced

• What are the effects?
  – A reduction in the number of operations or rebalancing of the operation types

• What is the downside?
  – Changes to the application are expensive, in terms of development time
Conclusions

• Performance is an important non-functional requirement
  – It is the non-functional requirements that most directly impacts the user
    • New major software updates tout performance as their new major feature
  – Bad performance can overshadow the features and functionality of the application
Conclusions

• Measurements are hard
  – How can you do precise measurements of software?
    • Any measured performance needs to be done under experimental conditions
      – Document your experiment
      – Repeat it
      – Statistical information
  – But is this useful information?
    • Non-determinism of software
    • Limitation of your tools
Conclusions

• Doing performance modeling may be more useful
  – Can be done before implementation
  – Can show “hot spots” in the model
  – Doing performance changes before implementation is cheaper!

• Model operations and operational sources
  – I/O operations and disk speed
Conclusions

• Model the external interaction
  – Think time

• Look for queuing
  – Places where more operations are coming in than the platform can handle

• Reduce the queuing
Conclusions

- Other factors to work with
  - Utilization
    - If resources are underutilized, they are being wasted
  - Service Requirement
    - If the service is taking too long to process the request, investing in a faster resource may improve performance
Way Forward

• System Design
  – Taking the requirements and turning them into a system plan