

# Some thoughts on Parameters, Workloads, Metrics & Graphing

Shivkumar Kalyanaraman  
Rensselaer Polytechnic Institute  
shivkuma@ecse.rpi.edu

<http://www.ecse.rpi.edu/Homepages/shivkuma>

GOOGLE: "Shiv RPI"



- ❑ Black-box modeling: parameters/factors, metrics
- ❑ Iterative design: importance of hypothesis (or expectation) before running an experiment or looking at a graph
- ❑ Summary results vs Graphing vs Tracing
- ❑ Jain book: Early Chapters

# Recall: System-Under-Test (SUT) Model

*Parameters*



*Factors*



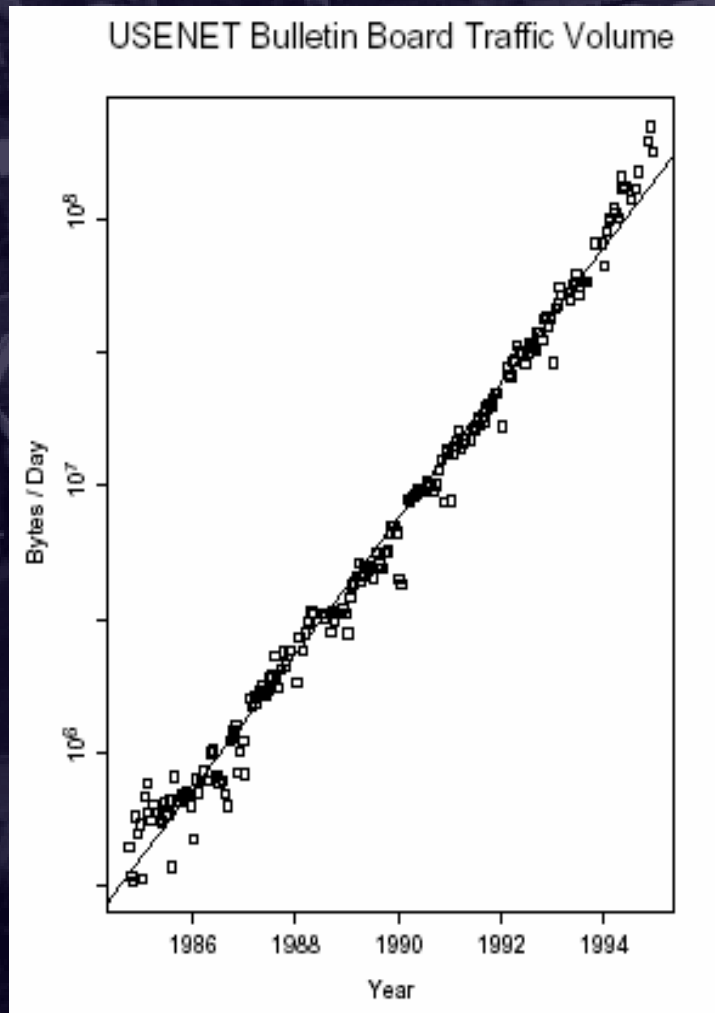
**System-  
Under-  
Test**

*Metrics*



Subject system to a set of tests (workloads/conditions)

## Example SUT



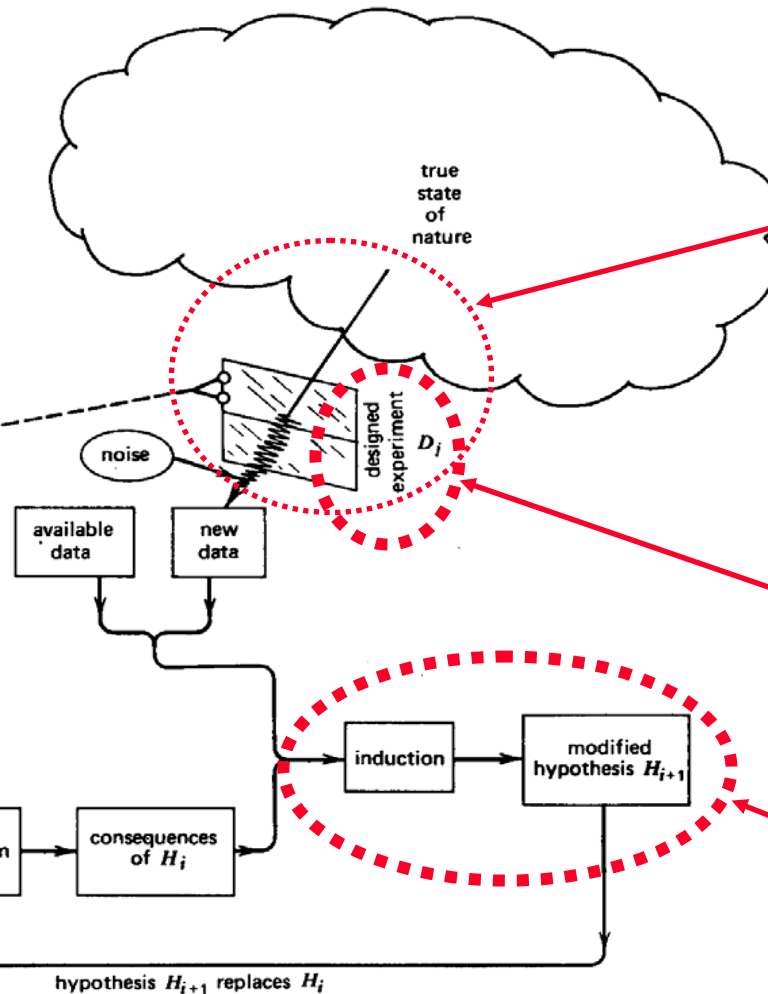
- Performance analysis “statements”:
- Growth of 80%/year
- Sustained for at least ten years ...
- ... before the Web even existed.
- ⇒ Internet is always changing. You do not have a lot of time to understand it.

**SUT: USENET BBoard**  
**Factor: Year (1970-2000)**  
**Metric: Bytes/Day**

**Model: Linear regression**

# Iterative Design Process

Model,  
Hypothesis,  
Predictions



Measure,  
simulate,  
experiment

How to make  
this empirical  
design process  
**EFFICIENT**??

How to *avoid*  
*pitfalls* in  
inference!

FIGURE 1.3. Data generation and data analysis in scientific investigation.

# Important: Have a *hypothesis*!

- ❑ A.k.a: “expectation”
- ❑ Else, nothing to test for!
- ❑ Hypothesis defines uncertainty (i.e. defines the problem)
  - ❑ Only way to navigate uncertainty: define what exactly is uncertain and what is certain (or less uncertain!)
  - ❑ Hypothesis is an educated “guess” about the residual uncertainty
- ❑ Performance analysis is like a detective story:
  - ❑ Goal: validate or invalidate the hypothesis either through illustration or rigorous quantitative results of experiments
  - ❑ You first need a “theory” or “hypothesis” (based upon hunches etc) which you are trying to validate
- ❑ In other words,
  - ❑ Expect something from each simulation result: graph or metric.
    - ❑ Eg: If design is bug-free, I should expect .... to happen
  - ❑ Any deviation from such expectations (or hypotheses) gives valuable clues.
  - ❑ True understanding => fitting a chain of causation from parameters to the results obtained.
  - ❑ Don't move on till you really understand!

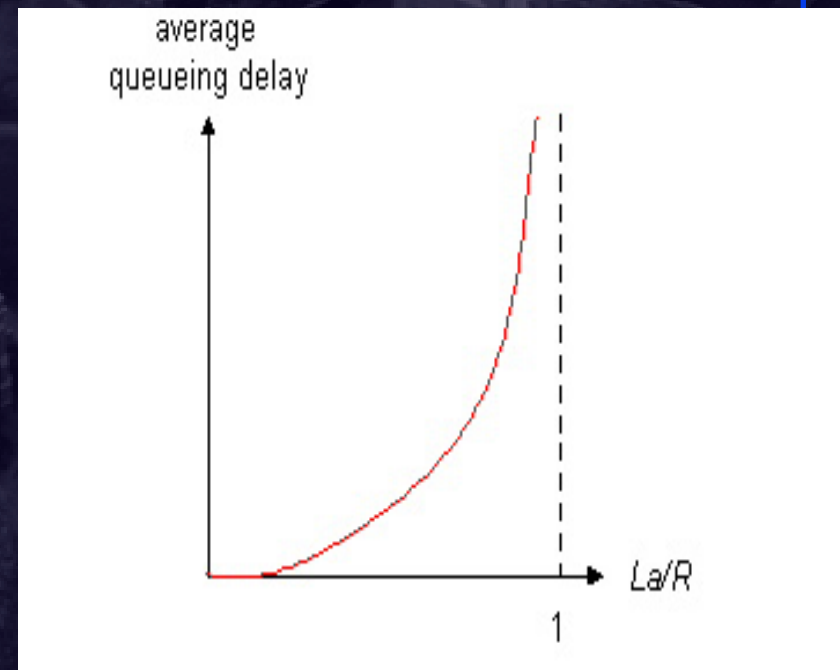
Shivkumar Kalyanaraman

# What's a performance *tradeoff* ?

- A situation where you cannot get something for nothing!
- Also known as a zero-sum game.

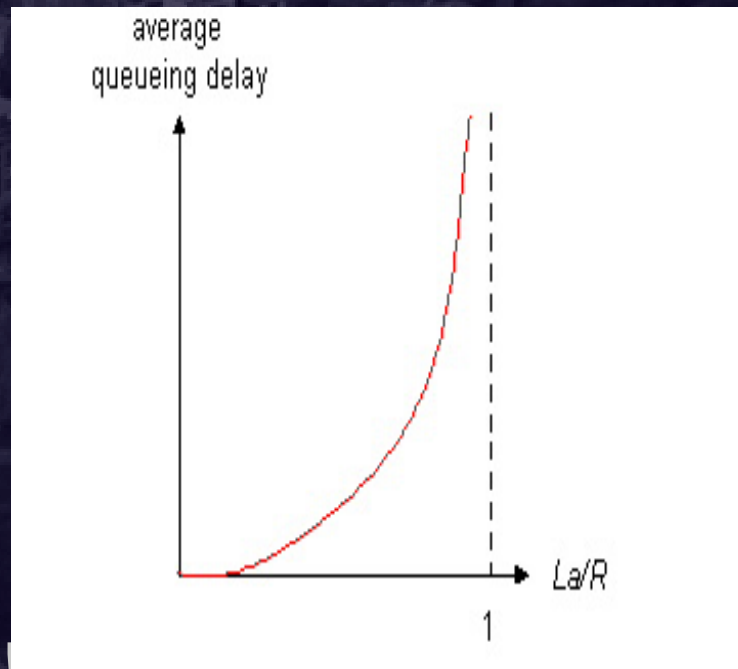
- ❑  $R$ =link bandwidth (bps)
- ❑  $L$ =packet length (bits)
- ❑  $a$ =average packet arrival rate

Traffic intensity =  $La/R$



# What's a performance *tradeoff* ?

- $\rho \sim 0$ : average queuing delay small
- $\rho \rightarrow 1$ : delays become large
- $\rho > 1$ : average delay infinite (*service degrades unboundedly  $\Rightarrow$  instability*)!



**Summary:** Multiplexing using bus topologies has both **direct** resource costs and **intangible** costs like potential instability, buffer/queuing delay.

# Amdahl's Law

- ❑ [http://en.wikipedia.org/wiki/Amdahl's\\_law](http://en.wikipedia.org/wiki/Amdahl's_law)
- ❑ Guides the iterative design process.
- ❑ If design implies a set of tradeoffs, the question is how to redesign components so that the system cost-performance (in terms of expensive resources) is improved.
- ❑ Amdahl's law talks about the maximum expected improvement to an overall system when only a part of the system is improved.
  - ❑ Statement of “diminishing returns” when the focus is on improving only a single component

- ❑ System Speedup = 
$$\frac{1}{(1 - P) + \frac{P}{S}}$$

# Amdahl's Law (contd)

- ❑ If a part of a system accounts for 12% of performance ( $P = 0.12$ ) and
- ❑ You speed it up 100-fold ( $S = 100$ )
- ❑ The actual system speedup is only: 13.6% !!!!

$$\frac{1}{1 - 0.12} = 1.136$$

- ❑ **Lesson #1:** Optimize the common cases (accounting for a large fraction of system performance)
- ❑ **Lesson #2:** Bottlenecks shift! If you optimize one component, another will become the new bottleneck!

# A good test case: “Workload”

- ❑ Actually tests the system: a.k.a “workload”
- ❑ Either “stress” test or
  - ❑ “Representative” test
  - ❑ Note: the book talks about representativeness as primary.
- ❑ Goal: to expose clearly:
  - ❑ the potential bugs (impln or design bugs),
  - ❑ and tradeoffs the system makes
- ❑ Test cases should be used:
  - ❑ To validate expectations (or hypotheses):
    - ❑ pay PARANOID levels of attention to unexpected results.
    - ❑ All results are suspect unless thoroughly cross-validated!
  - ❑ To explore areas of uncertain performance (either to get a better “feel” for performance or quantify it)

# Benchmark: Set of Workloads

- ❑ A suite of workloads that together:
  - ❑ captures a set of desired “stress” points, or
  - ❑ Is representative as a group
- ❑ Key:
  - ❑ complementary workloads;
  - ❑ Opportunities for cross-validation to catch bugs
- ❑ Eg: SPECweb benchmarks
- ❑ Pitfalls: gaming of benchmarks
  - ❑ Try to put special hacks that work well especially for the benchmarks
  - ❑ Don't kid yourself that a scheme that works on a poorly designed benchmark actually works well more broadly
- ❑ Flip side: ALL performance is relative to some set of input workloads (your goal is to either optimize the common case or be robust for a wide variety of workloads)

# Parameters vs Factors

- ❑ Both parameters and factors are inputs to the SUT that have an influence on results.
- ❑ Factor = parameter that is actually varied in the simulation(s)
- ❑ Non-Factor Parameters: set to a fixed value
- ❑ Eg: A single bottleneck topology
  - ❑ This topological structure may not be varied for many simulations and is hence a fixed parameter (not a factor)
  - ❑ However, the number of flows, or traffic composition or protocols used may be varied (factors)
  - ❑ A multi-bottleneck topology used => topology is now a “categorical” factor
- ❑ Inputs that don't matter (i.e. output metrics are insensitive to changes in that input value) => non-parameters (can be thrown out)

# Metrics

- ❑ Metrics are measures of system performance
  - ❑ Any system has tradeoffs: don't kid yourself!
- ❑ The set of metrics should together:
  - ❑ capture the hypothesized tradeoffs.
  - ❑ Or, probe into potential “buggy” points or variables (trace or animations)
  - ❑ Or, illustrate the performance and dynamics (graphs)
    - ❑ Facilitate visualization (eg: animation) and exploratory analysis
    - ❑ Goal: to determine WHAT to hypothesize
  - ❑ Facilitate cross-validation questions/hypotheses
  - ❑ Inspire confidence that the system “works” if the cross-validation questions are satisfactorily answered.
- ❑ Metrics can be either summary results, graphs or traces/animations

Shivkumar Kalyanaraman

# Metrics: Summary Results

- ❑ Metric is a single number, or a small set of numbers
  - ❑ A.k.a “point” or “range” estimates.
- ❑ Gives a quick snapshot of performance: eg: mean/median/variance etc
- ❑ Big bugs will show up as unexpected performance, needing deeper analysis
- ❑ But, bugs may also stay hidden.
- ❑ Need detailed graphs and/or traces to validate the reasons underlying the summary results

# Metrics: Graphing

- ❑ Summary results only give summary (eg: mean, CoV).
  - ❑ It does not capture the details of time-dynamics
- ❑ Time Graphs (Metric vs Time) capture dynamics
- ❑ Distribution graphs: capture “distribution”
  - ❑ PDF, distribution of throughputs (histogram)
- ❑ Important to put together the COMPOSITE PICTURE and CROSS-VALIDATE your results and check it against expectations for each case.
  - ❑ Eg: Queue length & Utilization graphs each tell only half the story in bottleneck links
- ❑ Always check the implications & correctness of summary results with detailed graphs that tell the WHOLE story

# Tracing

- ❑ Even more detailed analysis of scheme details
- ❑ Think of scheme sub-components as system-under-test
- ❑ Think of sub-components as “parameters” or “factors”
- ❑ SANITY CHECK for performance (as measured by summary results and graphs)
- ❑ Could use graphing, animation of such detailed variables.
- ❑ Typically done for a short period of interest (where funny things seem to be going on)
- ❑ Multi-resolution tracing:
  - ❑ Start with summary results
  - ❑ Then go to few key graphs
  - ❑ Then go to detailed traces
- ❑ Unexpected => pinpoint & iterate at appropriate level

# Profiling

- ❑ Data to support a form of cross-validation
- ❑ See whether performance “adds up” and/or
- ❑ What parts of the system are actually “exercised”
  - ❑ Did the test actually “stress” the system?
  - ❑ Is there a mystery of missing performance?
- ❑ Eg: Overheads in a new version of TCP over 802.11 network
  - ❑ MAC level overheads
  - ❑ Wasted error recovery packets (FEC or retransmission)
  - ❑ Packet-header overheads
  - ❑ Idle times due to timeouts
- ❑ Identify relative contributions of inefficiency categories.
  - ❑ Apply Amdahl’s law to determine “dominant” categories to optimize
  - ❑ Don’t waste time on optimizing the rest (will not contribute much in terms of system-level performance boost)!

# Iterative Design Changes vs Hacks

- ❑ Hack:
  - ❑ quick fix to a problem;
  - ❑ usually inelegant (quick & dirty) => evokes scorn from seasoned designers 😊
  - ❑ May depend upon “magic numbers”
  - ❑ Brittle: may not work if the workload or conditions change slightly
- ❑ Sometimes hacks are necessary to avoid getting stuck.
  - ❑ Have to keep track of all the hacks and understand their side-effects!
  - ❑ Depending upon hacks long term is poor practice...

# Iterative Design Changes vs Hacks

- ❑ Careful understanding & acknowledgement of tradeoffs: what is costly vs what is cheap?
- ❑ Good designs tradeoff cheap resources and optimize on expensive resources
  - ❑ (or realize properties that were impossible earlier)
- ❑ If the tradeoff is between expensive resources, it is cleanly parameterized that allows for control of the tradeoff.
  - ❑ Parameter sensitivity is well understood.
  - ❑ They make assumptions about the evolution of technology (eg: Moore's law, Metcalfe's law etc) & are consistent with it.
- ❑ Interaction between modules well specified and understood
- ❑ Designs are longer-lived: like interfaces, the goal is “stability” or “invariance”

Shivkumar Kalyanaraman

# SUMMARY: Course Highlights

- 3. **Systematic Tracing, Graphing, Profiling:**
  - Define *parameters* (input) and *metrics* (output)
  - **Parameter** criteria: all params that have performance impact (or a subset relevant to the performance “view”)
  - **Metric** criteria: must capture the relevant tradeoffs
    - Time series graphs vs point estimates
    - Examples of good, poor graphs;
  - **Workloads**: must stress test the system, capture relevant aspects of reality (in stages)
    - Issues with randomness: confidence intervals etc
  - **Profiling**: accounting for performance: contributions of components. Does it add up? Apply amdahl’s law to decide where to make changes
  - **Tracing**: at different degrees of resolution (low pass, high pass): helps in design debugging

An aerial photograph of the Rensselaer Polytechnic Institute campus, showing various academic buildings, a large central building with a dome, and surrounding greenery. The image is overlaid with a semi-transparent blue filter. The word "Amen!" is written in a bold, yellow, sans-serif font in the center of the image.

**Amen!**