Performance Evaluation

CS 240: *Computing Systems and Concurrency*
Lecture 22

Marco Canini

Context and today's outline

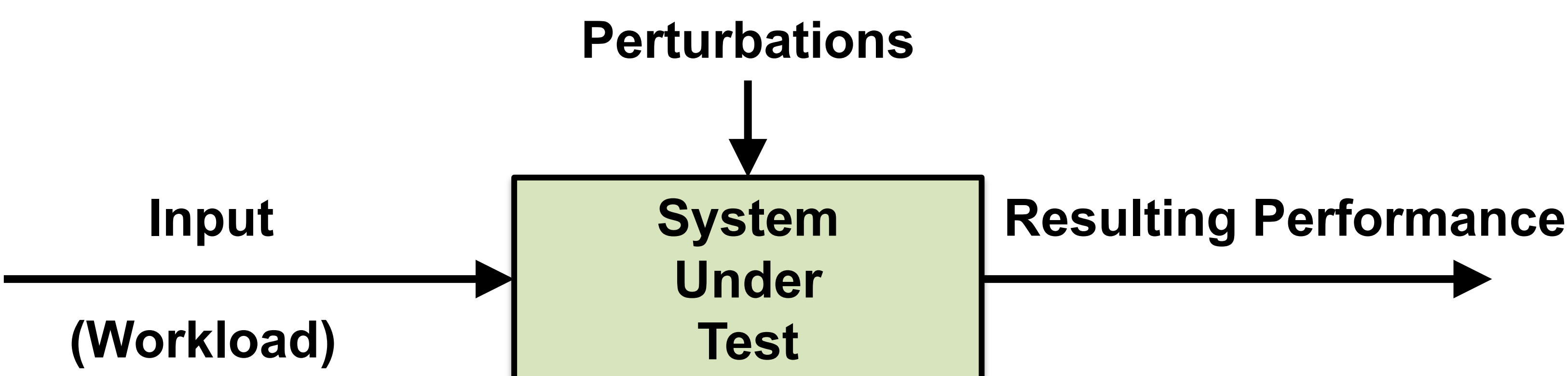
- We cared a lot about:
  - Are the results correct?

- But in practice we also need to consider quantitatively:
  - Are the results obtained in a reasonable time?
  - Is a system faster than another one?

- **Today—** How to analyze the performance of a system?

What’s systems performance?

- The study of an entire system, including all physical components and the full software stack
- Include anything that can affect performance
  - Anything in the data path, software or hardware
  - For distributed systems, this means multiple servers

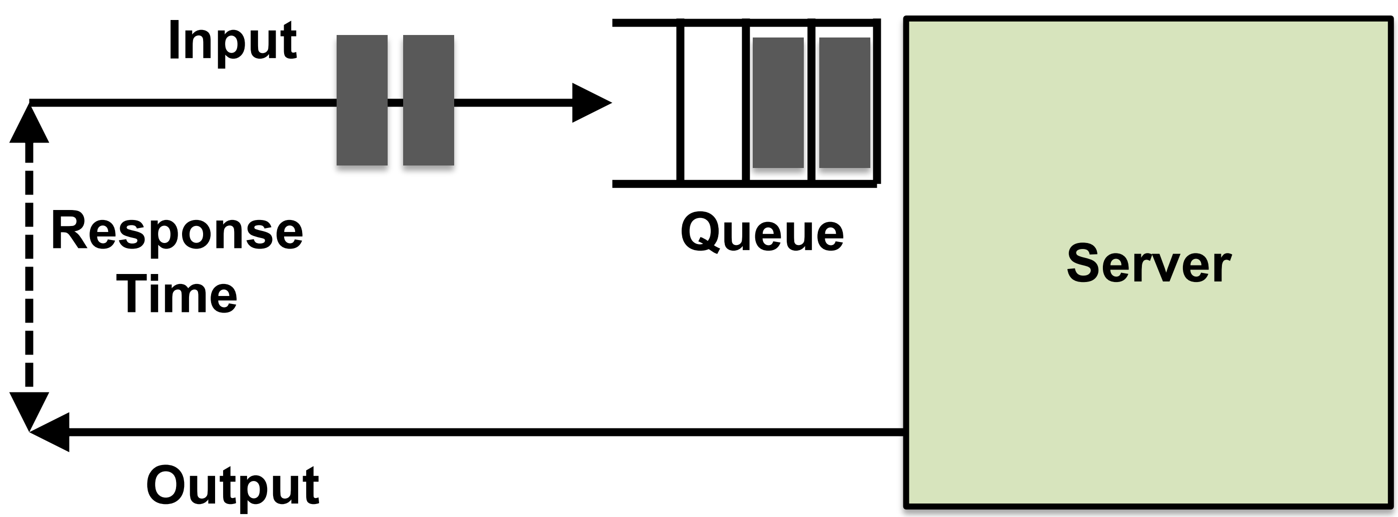


Some terms

- **Workload**
  – The input to the system or load applied
- **Utilization**
  – A measure of how busy a resource is
  – The capacity consumed (for a capacity-based resource)
- **Saturation**
  – The degree to which a resource has queued work it cannot service
- **Bottleneck**
  – A resource that limits the system performance

More terms

- **Response time**
  - The time for an operation to complete
  - Includes any time spent waiting (**queuing time**) and time spent being serviced (**service time**), and time to transfer the result

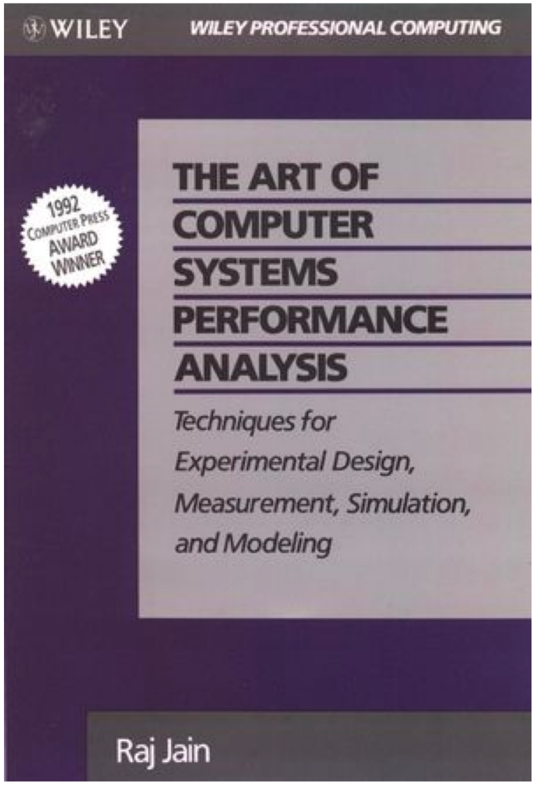


Who in interested?

- Many roles:
  - Sys admins / capacity planners
  - Support staff
  - Application developers
  - DB / Web admins
  - Researchers
  - Performance engineers (primary activity)

Performance evaluation is an art

- Like a work of art, a successful evaluation cannot be produced mechanically
- Every evaluation requires an intimate knowledge of the system and a careful selection of methodology, workloads and tools
- **Performance is challenging**



Performance is subjective

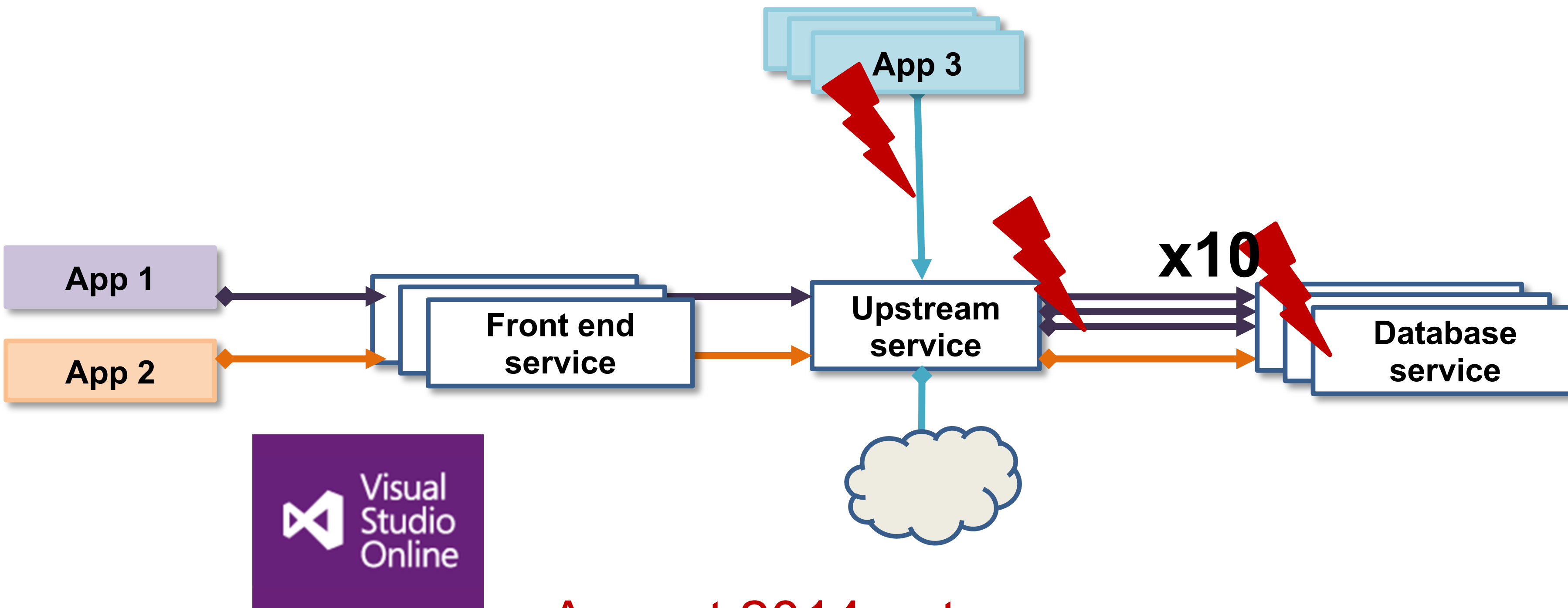
- Is there an issue to begin with? If so, when is it considered fixed?
- Consider:
  - The average disk I/O response time is 1 ms
- Is this **good** or **bad**?
- Response time is one of the best metrics to quantify performance; the difficulty is **interpreting** its information
- Performance objectives and goals need to be clear
  - Orient expectations as well as choice of techniques, tools, metrics and workloads

Systems are complex

- Many components and sources of root causes
- Issues may arise from complex interactions between subsystems that operate well in isolation
  - Cascading failures: when one failed component causes performance issues in others
- Bottlenecks may be complex and related in unexpected ways
  - Fixing one may simply move the bottleneck elsewhere
- Issue may be caused by characteristics of workload that are hard to reproduce in isolation

- Solving complex issues often require a holistic approach
  - The whole system needs to be investigated

Example of cascading failure



August 2014 outage

- One request type was accessing a single slow database and exhausted an upstream service's thread pool
- This starved other unrelated requests… causing application unavailability

Measurement is crucial

- You can’t optimize what you don’t know
- Must quantify the magnitude of issues
- Measuring an existing system helps to see its performance and perhaps the room for possible improvements

- **Need to define metrics**

- Know your tools!
- Be systematic!
- Don’t reinvent the wheel!

Latency

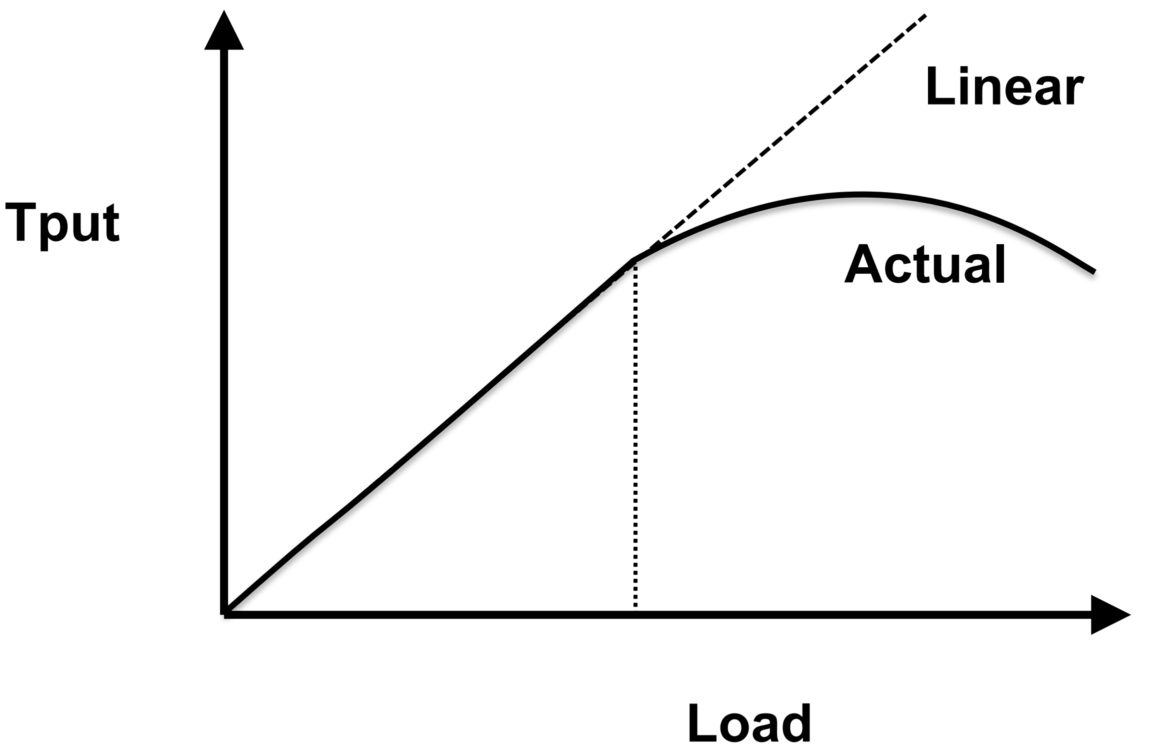
- The time spent waiting
  - E.g., setup a network connection
- **Or (broadly)**
- The time for any operation to complete
  - E.g., data transfer over the network, an RPC, a DB query, a file system write

- Can allow to estimate maximum speedup
  - E.g., assume the network had infinite capacity and transfer were instantaneous, how fast would the system go?

Throughput

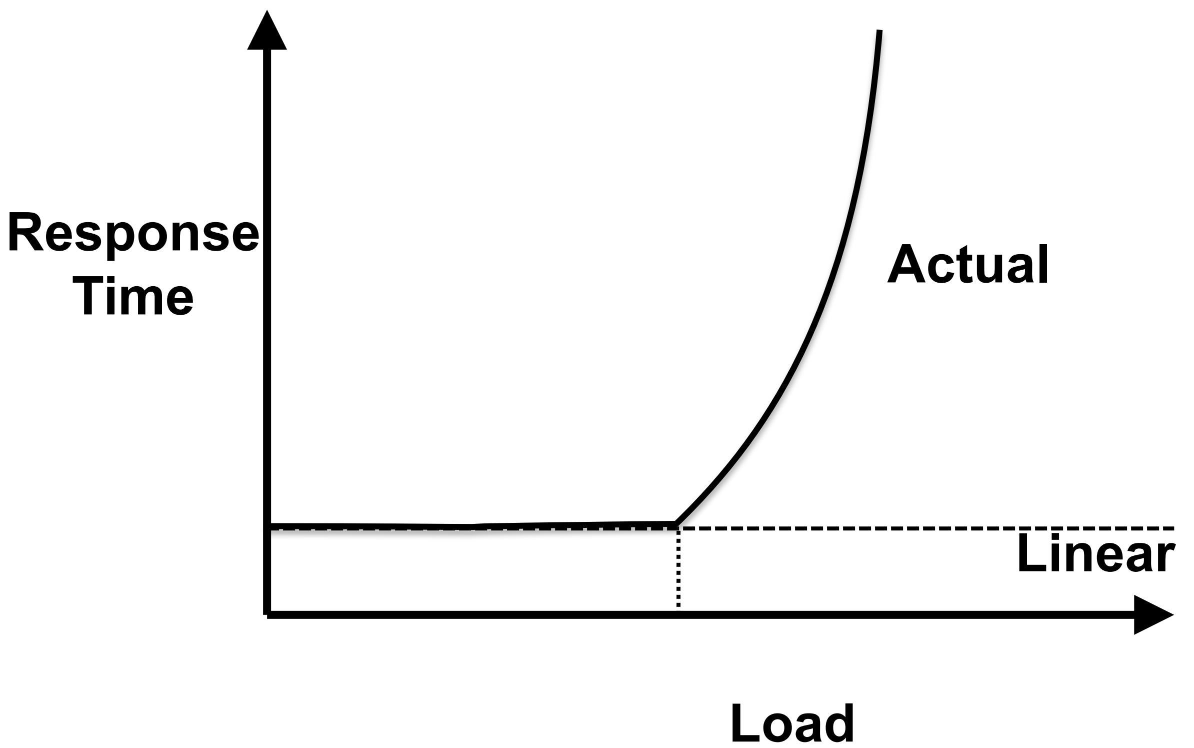
- The rate of work performed
- In communication:
  - Data rate: bytes per second, bits per second
  - (Goodput useful throughput: rate for the payload only)
- Systems:
  - Operation rate: ops per second, txns, per second
- IOPS
  - Input/output operations per second
  - E.g., reads and writes to disk per second

Scalability



(or a resource utilization as it approaches 100%)

Performance degradation



Problem

- System X has a performance problem
- What would you do to identify a cause?
- Scientific Method
  1. Question
  2. Hypothesis
  3. Prediction
  4. Test (Observational / Experimental)
  5. Analysis

Five ways not to fool yourself or: designing experiments for understanding performance

Tim Harris

https://timharris.uk/misc/five-ways.pdf

Measure as you go

- Develop good test harness for running experiments **early**
- Have scripts for plotting results
- Automate as much as possible
  - Ideally it is a single click process!
- Divide experimental data from plot data

Gain confidence (and understanding)

- Plot what you measure
- Be careful about trade-offs
- Beware of averages
- Check experiments are reproducible

Include lightweight sanity checks

- It’s easy for things to go wrong… and without noticing…
- Make sure you catch problems
- Have sufficiently cheap checks to leave on in all runs
- Have sanity checks at the end of a run
- And don’t output results if any problem occurs

Understand simple cases first

- Start with simple settings and check the system behaves as expected
- Be in control of sources of uncertainty to the largest extent possible
  - And use checks to detect if that assumption does not hold
- Simplify workloads and make sure experiments are long enough
- Use these as a performance regression test for the future

Look beyond timing

- End to end improvements are great but are they happening because of your optimization?
- Try to link differences in workloads with performance
- Look further into differences in resource utilization and statistics from performance counters

Toward production setting

- Do observations made in simple controller settings hold in more complex environments?
- If that is not true, try to decouple a number of aspects of this problem
- Change one factor at a time
- Try to understand the differences

Document results

- You will forget!
  - What did that experiment produce?
  - Where did I see that result?
- Pick a good convention to save data
- Use non destructive approaches
- Write summary of observations and possible explanations
  - Recall: our objective is better understanding
- Pick a good tool for experimenting, documenting and sharing
  - Try Jupyter

Next lecture topic:

Data-intensive computing I: graph processing, distributed ML