

# Performance Evaluation

Åge Kvalnes

Microsoft

# Performance Evaluation

Computer system designers, administrators, and users all have a goal of providing the highest performance at the lowest cost


A performance evaluation is a quantification of performance and cost

# A motivating example

Windows Virtual Machine pricing in Azure

Instance	vCPU	RAM	TMP storage	Price
D2 v3	2	8GiB	16GiB	\$0.192/hour
D4 v3	4	16.00 GiB	32 GiB	\$0.384/hour
D8 v3	8	32.00 GiB	64 GiB	\$0.768/hour

For there to be a profit margin, cost of service must have been quantified, *thoroughly*



But if I want to use this service for my system, how do I know whether I should select a D2 v3 or a D8 v3?

=> You need to conduct a performance evaluation of your system

# But what is performance

Code example:

Quantifying the performance of a simple program by *instrumenting* the program

# Some learnings from the code example

Execution time can be used as a metric for performance. But, execution time varies with *workload* (# items to sort and their distribution)

Is there a better performance metric? What about execution time/#items (i.e., execution time per item)?

- $2/8 = 0.25$
- $21/128 = 0.16$
- $132499/262144 = 0.51$

Execution time per item is not constant (NLogN algorithm)..and we have an *outlier* for 8 items

## Note:

- **A performance evaluation involves selection of one or more workloads**
- **To provide meaningful results, workloads must be representative of system use in real life**
- **Beware of outliers in the performance data (use statistical techniques to remove)**
- **Background load in the system under test can be a source of performance variance**

# Another code example

Performance data could be obtained through program instrumentation, but this approach was somewhat cumbersome

Code example:

Automating the instrumentation by help from the compiler (profiling)

# Other profiling approaches

Compiler-automated profiling simplifies instrumentation, but adds a lot of overhead. In general, cannot profile without affecting system behavior

*Statistical profiling*: interrupt program regularly and collect information about the currently executing instruction. Correlate with a program symbol table to obtain a performance profile. Overhead can be traded for accuracy

*Hardware-supported profiling*: most modern CPUs have built-in capabilities for counting events such as # instructions executed, # cache misses, +++

- Have a look at the [Intel Architecture Manuals](#) to see capabilities

See [here](#) for a large list of profiling tools with various capabilities

# Quantifying performance, in practice

- A *monitor* tool is typically run on machines hosting a production service
- The monitor observes and collects system performance statistics (counters and logs) provided through OS interfaces and from service instrumentation
  - Look at [Event Tracing for Windows](#) for how to instrument an application to produce trace events and consume trace events efficiently and [Windows Performance Monitor](#) for inspecting traces and counters
  - Important to instrument such that a request can be tracked across system components
    - Create *correlation ID* when request enters system and pass across component calls. All log messages include correlation ID

Example application instrumentation



```
RealTimeMetricUtilities.LogLatencyMetric(  
    this.griffinRequestContext.Diagnostics.RealTimeMetricStore,  
    "ObjectStoreClient",  
    "DeleteItem",  
    RealTimeMetricConstants.YggdrasilRealTimeMetricMetadata,  
    RealTimeMetricConstants.ObjectStoreSetItemNetworkLatencyMetricName,  
    coprocResult.PerformanceRecord.ServicePerformanceList[0].SumWaitTimeUs / 1000);
```



# Windows performance counters

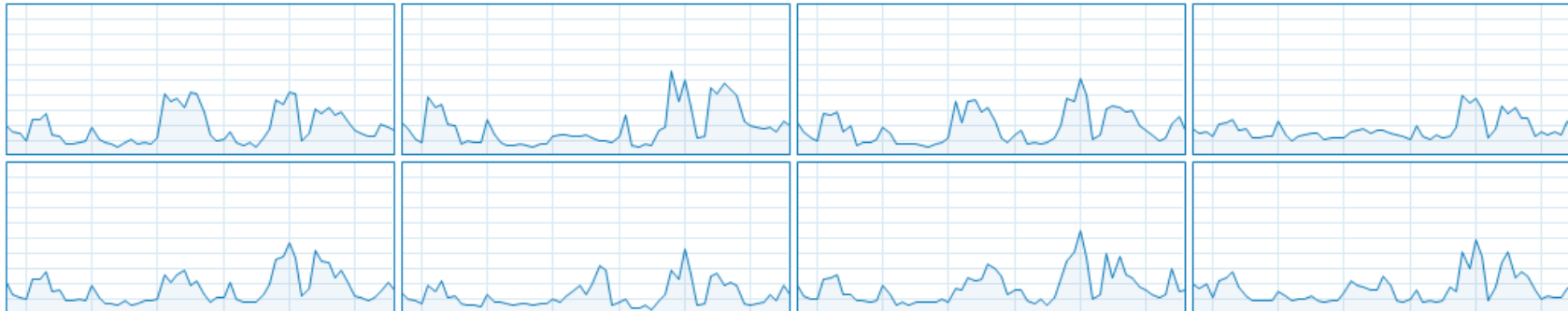
Windows (and Linux) is heavily instrumented to provide key performance data

## CPU

Intel(R) Xeon(R) CPU E5-1620 v2 @ 3.70GHz

% Utilization over 60 seconds

100%



Utilization	Speed	Maximum speed:	3.70 GHz
17%	2.78 GHz	Sockets:	1
Processes	Threads	Cores:	4
274	4231	Logical processors:	8
Handles	169535	Virtualization:	Enabled
Up time	2:13:41:58	L1 cache:	256 KB
		L2 cache:	1.0 MB
		L3 cache:	10.0 MB



All this perfmon data is available programmatically through performance counter APIs

# More motivation for performance evaluation

- Validating that performance requirements are met
  - 99% of all requests should complete in less than 500ms
- Evaluating design alternatives
  - Should we buy those expensive SSDs or can HDDs be used?
- Finding performance bottlenecks
  - Shifting that bottleneck to somewhere else can reduce cost
- Characterizing system load
  - Applications that stress different resources can potentially be packed on the same machine, thereby reducing cost
  - Forecasting future HW resource needs. It can take months from order to delivery. Growth estimation for next 6 months common in industry

# Systematic performance evaluation

## 1. State goals and define the system

- System boundaries affect selection of performance metrics
  - If client requests have to pass through an intermediate network, client-to-service network latency is perhaps not a viable performance metric
  - If the system runs in a virtualized environment with other competing VMs, a more thorough approach to outlier removal is perhaps needed

## 2. List services and outcomes

- Determine what services the system provides, and what requests can be made to each system component.
- Involves learning about system architecture and how things work
- List possible outcomes of use of each service and component requests

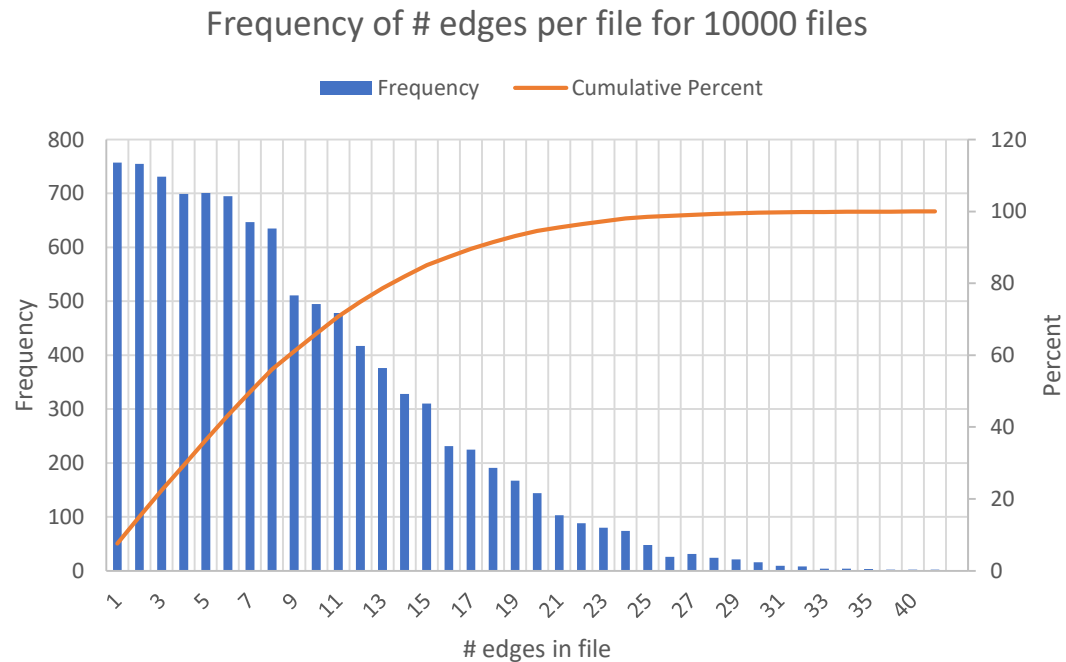
# Systematic performance evaluation

3. Select metrics. They are generally related to speed, reliability, and availability

- If requests succeed
  - Typically response time, throughput, utilization
- If requests fail
  - What is the observed error probability and time between errors?
- If the system fails (or a component)
  - What is the likelihood and duration
- Be prepared to iterate over metric selection based on observations

# Systematic performance evaluation

4. List parameters that affect performance and iterate based on experience
  - System parameters include hardware and software parameters
  - Workload parameters must be carefully selected and be representative of real load



# Systematic performance evaluation

5. Select factors to study. Parameters that will be varied are called factors

- Parameters that are likely to have high impact on performance are factor candidates
- Ideally all parameters should be factors, but this is not viable. Be very scared of that parameter not selected as a factor

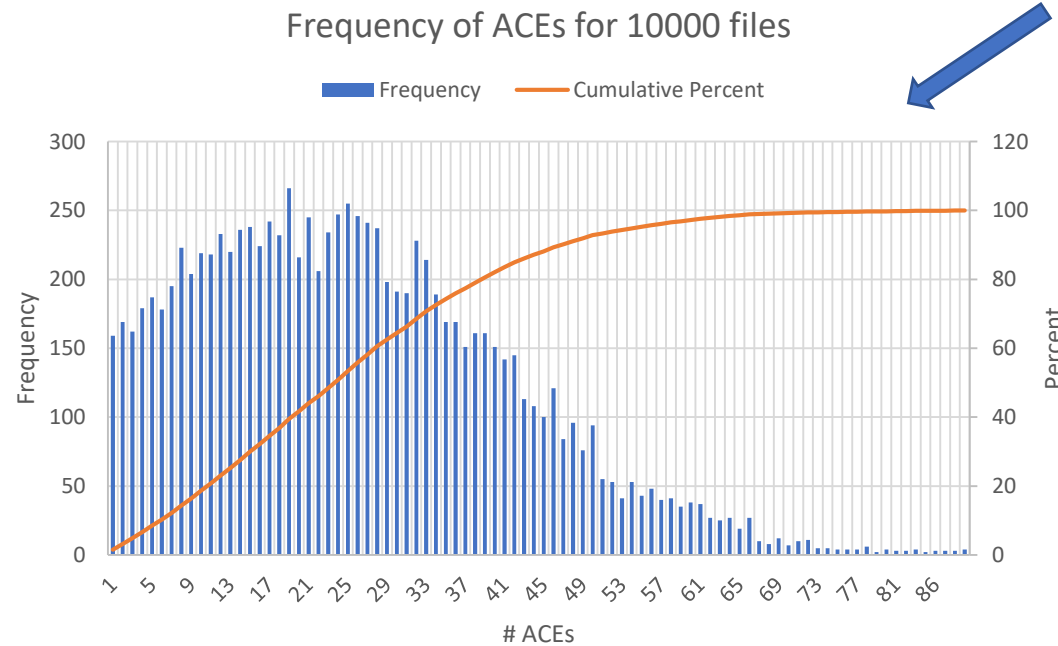
6. Select evaluation technique

- Analytical modeling. Low accuracy, but cheap
- Simulation. Moderate accuracy and moderate cost
- Measuring a real system. You have to build it
- All techniques may have to be used

# Systematic performance evaluation

## 7. Select workload

- For analytical modeling, probability of different requests
- For simulation, one can use a trace from a real system
- For measurement, representative of system usage



Synthetic data, but generated based on trace samples

# Systematic performance evaluation

## 8. Design experiments

- Decide on a sequence of experiments that offer maximum information with minimal effort
- Useful to conduct some initial experiments to determine the sensitivity of factor levels. This can significantly reduce the # experiments needed

## 9. Analyze and interpret data

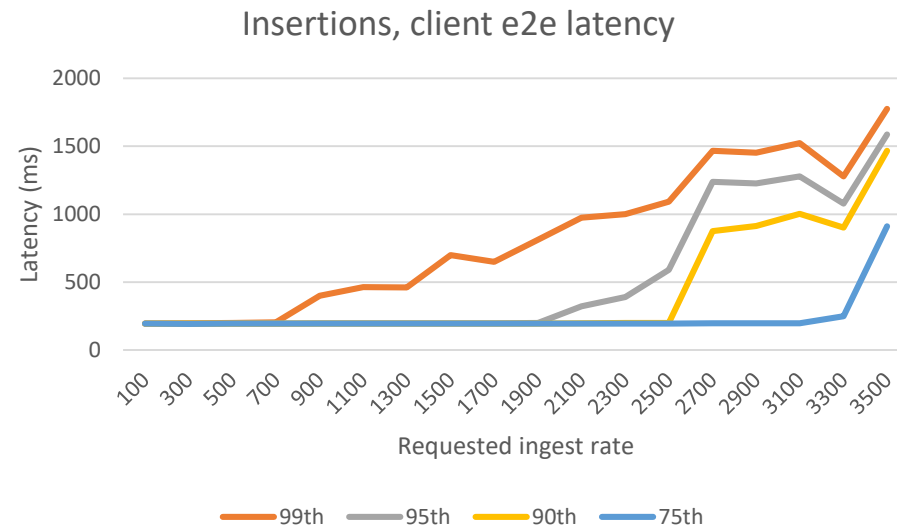
- Can experiments be repeated with the same outcomes? If not, variability must be taken into account
- Do not fall into the using-only-averages trap
  - Standard deviation (what is normal)
  - Percentiles are great (75% of all requests have this performance, 90% this, 95% this, etc.)



# Systematic performance evaluation

## 10. Present results

- Keep it simple and focus on the key outcomes
- Present graphs, not formulas
- Graphs must be self-contained (label each axis, descriptive legends, etc.)
- Hindsight is 20/20. Be prepared to repeat all steps based on the knowledge gained at this point..



# Common pitfalls

Goals	Methodology	Completeness	Analysis	Presentation
None	Unsystematic	Overlooking parameters	None	Ignoring social aspects
Biased	Incorrect metrics	Ignore factors	Erronous	Omitting assumptions
	Wrong workload	Level of detail	Too complex	Omitting limitations
	Wrong technique		Ignoring levels	
	Bad experiments		Ignoring errors	
			Ignoring variability	
			Ignoring outliers	

# Concluding remarks

- Be systematic
- Do not trust results until they have been validated. And even then, expect surprises

Performance evaluation is an art