INTRODUCTION
Motivation
▶ Why model system performance (in runtime)?
  – performance directly correlated to user satisfaction
  – performance is hard to predict, tuning often needed
  – required for (manual and automated) scaling
  – required for bottleneck detection and troubleshooting
▶ System performance has huge impact at (large) scale
  – Amazon: 100 ms extra latency costs 1% in sales
  – Bing: 2s slowdown changed queries / user by -1.8%
  – Google: (August 16 2013) 5 minute down time cost 545k$
  – Blizzard: <insert game name> launch overwhelms servers

What is Performance?
▶ Performance metrics
  – latency (transmission delay)
  – bandwidth (maximal transmission capacity)
  – throughput (average transmission rate)
  – response time (time to see result of action)
▶ Performance metrics typically application-specific
  – transactions per second in a database
  – frame rate or lag in a computer game
  – response time in a web application

But... What is Performance?
▶ Distributed systems: the illusion of using a local system
▶ Anything that breaks the illusion deteriorates performance
▶ Performance is application-specific
  – database: transparent scalable storage of data
  – computer game: immersion
  – web application: real-time response of action
▶ Often expressed in terms of time (e.g., delays)
  – database: consistency / synchronization timeouts
  – computer game: noticeable lag / delays
  – web application: response time
▶ As systems are complex, performance is complex
  – need to design for performance
  – need to tune for performance

Performance Degradation
▶ Performance degradations are typically continuous
  – non-perceivable slowdown
  – perceivable slowdown
  – unresponsive system
▶ Often possible to construct application-specific performance degradation limits, e.g., interactive system slowdown
  – 0 - 10 ms: not noticeable
  – 10 - 100 ms: noticeable but not annoying
  – 100 ms - 1 s: annoying but still usable
  – 1 - 10 s: perceived as asynchronous
  – 10+ s: useless
▶ Goals
  – hide performance degradations
  – degrade gracefully when unavoidable
Characterizing Performance

- What needs to be improved?
  - and how does it need to be improved?
- What do we need to measure?
  - NOTE: not what we can measure (surrogate metrics)
- Under what conditions?
  - realistic system settings
  - (predicted and actual) workload
- What do we have to work with?
  - metrics (measurements of performance facets)
  - models (system structure, configuration and components)
  - workloads (arrival rates, request types, data volumes, ...)
  - tools (simulators, emulators, workload generators, benchmarks, stress tests, profilers, ...)

Introduction

Key Performance Indicators (KPIs)

- Performance metrics relevant to a particular application
- Measures the success of a system in a specific dimension
- Often used to specify an acceptable level of QoS
- Requires knowledge of "what is important" to a system
- Systems composed of subsystems typically have KPIs with dependencies on subsystem KPIs
  - database transaction rate depends on storage throughput
  - web GUI response time depends on database query time
  - computer game lag depends on network latency
- Relating (sub)system KPIs often difficult and important

The Usual Suspects

Bandwidth

- Total system bandwidth (bits / second)
  - a measure of (maximal) throughput
  - amount of data that can be transferred in a specific time
- Local networks
  - system bandwidth = data transfer rate
    (when full transmission capacity can be utilized)
- Wide-area networks
  - system bandwidth not transfer rate
  - route heterogeneity
  - parallel transfers
  - load variations
  - ...

Transmission Time

Throughput

- The rate of successful message delivery over a communication channel
- Maximum Theoretical Throughput
  - channel capacity (ideal circumstances)
- Peak Measured Throughput
  - throughput measured over a short period of time
  - useful for burst transmission systems
- Maximum Sustained Throughput
  - max throughput integrated over a long period of time
  - zero packet loss: minimum load that increases latency
- Average (Sustained) Throughput
  - average throughput integrated over a long period of time
**Architectures**

**Models**

**Performance Metrics**

**Introduction**

**Today**

**Response Time**

- System reaction time
  - the time from issuing a command and seeing a result
  - typically the sum of a number of factors
- Variable and sometimes hard to predict
  - the effect of load variations (overload) on latency, transfer rate, and system bandwidth depends on the network technology
  - user perception of response time tends to focus on outliers ("users feel variance, not means")

**Latency, Throughput, Response Time**

- Throughput and latency are related, but not identical
  - pipelining
  - parallelism
  - noise
- Response time a function of both latency and throughput
  - Need to measure each individually
  - Which is more important?
  - depends on the application

**Amdahl’s Law**

- Estimates the maximum expected improvement to an overall system when only part of the system is improved
- Often used in parallel computing to predict the theoretical maximum speedup from using multiple processors
- Overall performance improvements are proportional to the importance of an improved component / execution path
- Can be used to identify bottlenecks in distributed systems
  - weigh upgrades by improvement, cost, risk and effort

**Amdahl’s Law Example**

- L1 cache reference: 0.5 ns
- Branch misprediction: 5 ns
- L2 cache reference: 7 ns
- Mutex lock / unlock: 25 ns
- Main memory reference: 100 ns
- Compress 1 kB with Snappy (Zippy): 3’000 ns
- Send 1 kB over 1 Gbps: 10’000 ns
- Read 1 MB from RAM (sequentially): 250’000 ns
- Packet round-trip within data center: 500’000 ns
- Read 1 MB from SSD (sequentially): 1’000’000 ns
- HD disk seek: 10’000’000 ns
- Read 1 MB from network (sequentially): 10’000’000 ns
- Read 1 MB from HD (sequentially): 20’000’000 ns
- Packet round-trip CA - Netherlands - CA: 150’000’000 ns

**Latency Numbers (ca 2009)**

- L1 cache reference: 0.5 ns
- Branch misprediction: 5 ns
- L2 cache reference: 7 ns
- Mutex lock / unlock: 25 ns
- Main memory reference: 100 ns
- Compress 1 kB with Snappy (Zippy): 3’000 ns
- Send 1 kB over 1 Gbps: 10’000 ns
- Read 1 MB from RAM (sequentially): 250’000 ns
- Packet round-trip within data center: 500’000 ns
- Read 1 MB from SSD (sequentially): 1’000’000 ns
- HD disk seek: 10’000’000 ns
- Read 1 MB from network (sequentially): 10’000’000 ns
- Read 1 MB from HD (sequentially): 20’000’000 ns
- Packet round-trip CA - Netherlands - CA: 150’000’000 ns
Latency Lessons

- Reads are cheap
- Writes are expensive (±40x cost of read)
- Network bandwidth > disk bandwidth
- Global shared data is expensive
- Lock contention in written objects kills performance
- Architect for scaling writes
- Optimize for low write contention
- Optimize wide (make writes parallel)

Performance Metrics

- Amount of resource capacity / time used to serve requests
- Resource capacity is multidimensional
  - CPU
  - RAM
  - network
  - I/O
  - storage
- Bottlenecks often arise from fragmentation or stranding
  - fragmentation: spare capacity not usable
  - stranding: capacity dimensions ignored (in scheduling)

Noise

- A number of factors impact performance metrics indirectly
  - network contention
  - request heterogeneity
  - OS scheduling overhead
  - cache misses
  - I/O contention
  - ...
- Models need to encompass some notion of noise (and how to deal with it)
  - filtering
  - sampling
  - robust statistics

Jitter

- Packet delay variation
  - delivery delay excluding lost packets
- Variations over time
  - noise
  - timing
  - complex interactions resulting in emergent behavior
- Repeat experiments, and interpret data strictly
  - box-plots instead of single value graphs
  - Is median more descriptive than mean?
  - Is your value an approximation or an (upper / lower) bound?

Resource Operational State

- Cold: just booted
  - caches empty
  - systems not adapted to workloads
- Warm: in continuous operation
  - caches filled, performance tuned
  - generally faster than cold
- Transient state
  - adapting to changed parameters
  - number of users, requests distribution, hardware
- Steady state
  - system configuration converged to optimum
  - generally faster than transient

Optimization of data center workload scheduling

- (Initial) placement of incoming jobs (virtual machines)
- (Continuous) opportunistic (migration based) re-placement

Constraints for

- RAM, CPU (core), and storage utilization
- local storage requirements for I/O intensive applications
- anti-affinity constraints for I/O intensive applications
- restrictions for migrations across network segments
- affinity constraints for CPU chipsets and architectures

Cost / Objective functions for

- load balancing
- service consolidation
- resource fragmentation
- energy efficiency
Today
Introduction
Performance Metrics
Models
Architectures

ARCHITECTURES

Client-Server Architectures

▶ Clients offload tasks to, and use resources of, centralized servers
▶ Typically client-driven
▶ Servers more or less monolithic
▶ Scaled by server clustering and load balancing

ARCHITECTURES

Two-Tier Architectures

▶ Separates server side into two tiers
▶ Interface tier
  – aggregates information and renders client data / interfaces
  – e.g., web servers
▶ Storage tier
  – manages data (including replication, redundancy, etc)
  – e.g., database servers
▶ Scaled via a level of indirection between tiers

ARCHITECTURES

Models

▶ Application models
  – describe system configuration and deployment
▶ User behavior models
  – describe user-system interactions
▶ Workload models
  – describe statistical patterns in workloads
▶ Infrastructure models
  – describe resource configuration, state, load, etc

ARCHITECTURES

Models

▶ Closed systems
  – closed loop between users and systems
  – all requests related to earlier requests
▶ Open systems
  – no loop between users and systems
  – requests independent of each other
▶ Partially open systems
  – requests from both closed loop and open loop users
  – users modeled with (loop) resubmit / leave probabilities

ARCHITECTURES

Data Mining / Big Data

▶ (Semi-)Automated analysis of large quantities of data to identify and extract new patterns of interest
  – cluster analysis (finding patterns)
  – anomaly detection (finding deviations)
  – association rule mining (finding dependencies)
▶ Interdisciplinary overlap
  – artificial intelligence
  – machine learning
  – statistics
  – database systems
▶ Current buzzword: business intelligence
▶ Applied to tune large data centers and systems

ARCHITECTURES

Workload Modeling

▶ Ability to predict workloads greatly useful
  – requires characterization (behavior models) of users
  – gives data to tune systems in many dimensions
▶ Offline analysis
  – analysis of historical data
  – high data volumes, large batch processing
▶ Online analysis
  – analysis of performance metrics in real time
  – requires high quality monitoring
  – adapts world view based on models from offline analysis
▶ “The future is hard to predict, especially the parts that have not happened yet”
Three-Tier Architectures
- Extend two-tier architectures with a logic layer
  - addresses, e.g., parallelization of compute tasks
- Interface tier
  - aggregates information and renders client data / interfaces
  - e.g., web servers
- Logic tier
  - encapsulates logic in extensible components
  - e.g., application servers
- Storage tier
  - manages data (including replication, redundancy, etc)
  - e.g., database servers
- Scaled via levels of indirection between tiers

Four-Tier Architectures
- Federates and deploys services in hierarchical Service-Oriented Architectures
  - addresses, e.g., flexible aggregation of services
- Client tier
  - components responsible for experience delivery
  - e.g., mobile clients, wearables, IoT devices
- Delivery tier
  - optimizes content for devices, caches data, models users
- Aggregation tier
  - aggregates and federates services
  - discovery for underlying services
  - data protocol translations (e.g., SOAP to JSON)
- Service tier
  - on-premise systems (records, services, data)
  - external third-party services
- Scaled via levels of indirection between tiers

Micro-Service Based Architectures
- Four/Many-tier architectures with lightweight services
- Services typically stateless
- Services typically replicated en masse
- Service interactions short-lived (single packet)
- Highly scalable
  - more flexible architectures / system orchestration
- More flexible architectures / system orchestration
- Highly instrumentable
  - better monitoring and analytics