Perspective: Performance and Scalability

CSCE 742 - Lecture 17 - 11/08/2018

Three Pillars of Architectural Design

- **Stakeholders** are the people impacted by the architecture, who have differing expectations and needs.
- **Viewpoints** are used to structure architecture definition by focusing on aspects of the system being designed.
- **Perspectives** focus on how a particular quality attribute impacts each viewpoint of the architecture.

Architectural Perspective

- **Architectural Perspectives** are used to discuss how particular quality attributes affect each view of the overall architecture.
- A perspective is a collection of activities, tactics, and guidelines used to ensure that a system exhibits the chosen set of quality attributes.
    - Systemizes what a good architect does: studies properties, assesses architectural choices, selects and applies tactics to ensure quality.

Performance

- The ability of the software to meet timing requirements.
- Can we characterize the pattern of events arriving and the pattern of responses?
    - Requests served per minute.
    - Variation in output time.
- Driving factor in architecture.
    - Often at expense of other quality attributes.
    - **All** systems have performance requirements.

Scalability

- The ability to “grow” the system to process an increasing number of requests.
  - While still meeting performance requirements.
- Horizontal scalability (“scaling out”)
  - Adding more resources to logical units.
    - Adding a cluster of servers.
    - “Elasticity” - can customers to add or remove VMs from a pool?
- Vertical scalability (“scaling up”)
  - Adding more resources to a physical unit.
    - Adding memory to a single computer.

Scalability

- How can we effectively utilize the additional resources?
- Effective scalability requires that:
  - Additional resources result in a performance improvement.
  - Did not require undue effort to add.
  - Did not disrupt operations.
- Can be thought of as a form of modifiability.
  - The system must be designed to scale (i.e., designed for concurrency).

Response Time

- Time it takes to complete an interaction.
  - I.e., between clicking a button and seeing the response on-screen.
- Responsiveness measures how quickly the system responds to routine tasks.
  - Key consideration: user productivity.
  - How responsive is the user's device? The system?
- Turnaround time measure time to complete larger tasks.
  - Can task be completed in available time?
  - Impact on system while running?
  - Can partial results be produced?

Response Time Requirements

- Responsiveness
  - Under a load of 350 update transactions per minute, 90% of “open account” requests should return a reply to the calling program within 10 seconds.
- Turnaround Time
  - Assuming a daily throughput of 850,000 requests, the process should take no longer than 4 hours, including writing results to a database. No other significant activity will take place during this period.
  - It must be possible to resynchronize with all monitoring stations and reset database to reflect the current state within 5 minutes. No status updates will be processed during the resynchronization period.

Throughput

- The workload a system can handle in a defined time period.
    - Shorter the processing time, higher the throughput.
    - However, as load increases (and throughput rises), response time for individual transactions tends to increase.
        - With 10 concurrent users, request takes 2s.
        - With 100 users, request takes 4s.
    - Possible to end up in situation where throughput goals conflict with response time goals.
        - With 10 users, each can perform 20 request per minute (throughput: 200/m).
        - With 100 users, each can perform 12 per minute (throughput: 1200/m - but at cost to response time).

Impact on Views

- Context
  - Highlights performance requirements for - and problems with - external interfaces.
  - Constraints can be identified and analyzed early.
- Functional
  - May identify changes in functional structure needed to achieve performance requirements.
    - (i.e., combine elements to reduce overhead)
- Information
  - Identifies shared resources, which place constraints on performance and scalability.
  - Identifies where information could be duplicated.

Impact on Views

- Concurrency
  - Identifies problems such as excessive contention.
  - Adding concurrency often improves performance.
- Development
  - Identifies guidelines for achieving performance goals during development (patterns).
- Deployment
  - Often guides creation of performance models.
  - System calibration impacts performance.
- Operational
  - Highlights need for performance monitoring and management.

Performance and Scalability Scenarios

Performance Quality Scenarios

- Measure system performance (as opposed to user performance).
- Begins with an event arriving at the system.
    - Responding requires resources (including time) to be consumed.
- Arrival pattern for events can be:
    - Periodic (at regular time intervals)
    - Stochastic (events arrive according to a distribution)
    - Sporadic (unknown timing, but known properties)
        - "No more than 600 per minute"
        - "At least 200 ms between arrival of two events"

Performance Quality Scenarios

- Response measurements include:
  - **Latency:** The time between the arrival of the stimulus and the system's response to it.
  - **Deadlines in processing:** Points where processing must have reached a particular stage.
  - **Throughput:** Usually number of transactions the system can process in a unit of time.
  - **Response Jitter:** The allowable variation in latency.
  - **Number of events not processed** because the system was too busy to respond.

Performance Quality Scenarios

- For real-time systems (i.e., embedded devices), measurements are absolute.
  - Look at worst-case scenario.
- For non-real-time systems, measurements should be probabilistic.
  - 95% of the time, the response should be N.
  - 99% of the time, the response should be M.

Specifying Requirements

- Response time targets require context of a defined load.
  - One transaction in 3s is easy if that is the only request. Can you still hit 3s if there are 500 transactions per second?
  - Requirements must specify context and a clearly-defined response time goal.
  - Also define when a transaction starts and ends.
- Why probabilistic definitions are important.
  - Not all requests take the same amount of time, even with constant load.

Generic Performance Scenario

- **Overview:** Description of the scenario.
- **System/environment state:** The system can be in various operational modes, such as normal, emergency, peak load, or overload.
- **External Stimulus:** Stimuli arrive from external or internal sources. The stimuli are event arrivals. The arrival pattern can be periodic, stochastic, or sporadic, characterized by numeric parameters.
- **Required system behavior:** The system must process the arriving events. This may cause a change in the system environment (e.g., from normal to overload mode).
- **Response measure:** The response measures are the time it takes to process the arriving events (latency or a deadline), the variation in this time (jitter), the number of events that can be processed within a particular time interval (throughput), or a characterization of the events that cannot be processed (miss rate).

Example Performance Scenario

- **Overview:** Check system responsiveness for adding items to shopping cart under normal operating conditions.
- **System/environment state:** Normal load is defined as deployment environment with no failures and less than 20 customer requests per second. System is communicating over good internet connection to acceptable client (see glossary for expected internet / client specifications).
- **External Stimulus:** Customer adds product to shopping cart.
- **Required system behavior:** Web page refreshes. Icon on right side of web page displays last item added to cart. If item is out of stock, cart icon has exclamation point overlay on top of cart icon.
- **Response measure:** In 95% of requests, web page is loaded and displayed to user within 1 second. In 99.9% of requests, web page is loaded and displayed to user within 5 seconds.

Scalability Scenarios

- Ability to address more requests is often part of **performance** scenarios.
- Scenarios assessing scalability directly (*the ability to adjust available resources to the system*) deal with the impact of adding or removing resources.
- Response measures reflect:
  - Changes to performance.
  - Changes to availability.
  - Load assigned to existing and new resources.

Example Scalability Scenario

- **Overview:** Addition of new hardware improves credit card transaction speed.
- **System/environment state:** Before addition of new hardware, 95% of credit card transactions were completed within 10 seconds, 99.9% within 15s. Additional server has doubled threads available for processing requests. System is under normal load, with normal connectivity (see glossary for expected internet / client specifications).
- **External Stimulus:** Customer completes a purchase.
- **Required system behavior:** Order confirmation is displayed, with a list of items purchased, expected arrival date, and total cost of items.
- **Response measure:** In 95% of requests, web page is loaded and displayed to user within 5 second. In 99.9% of requests, web page is loaded and displayed to user within 7.5 seconds.

Performance and Scalability Tactics

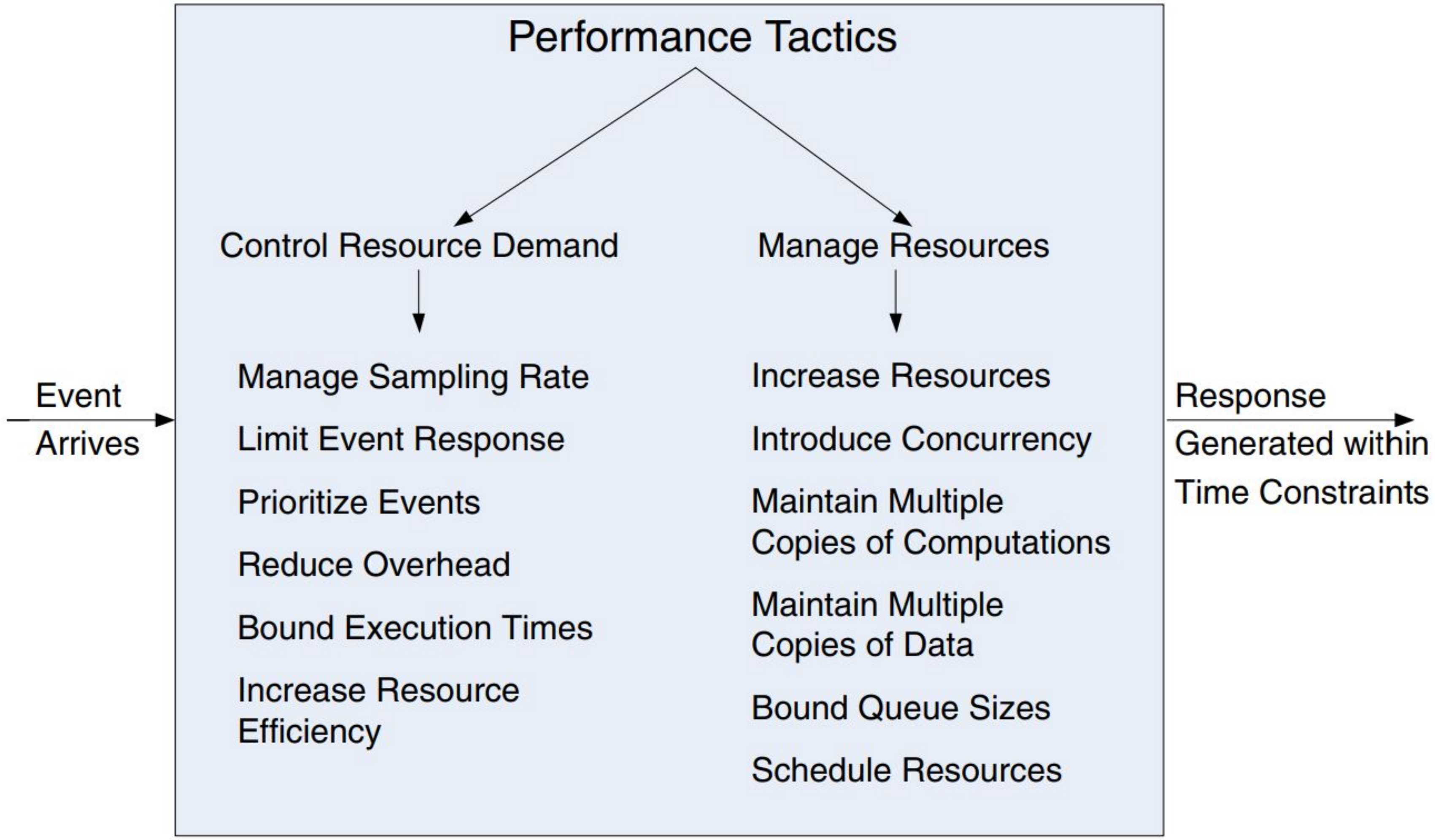
Processing Time

- When the system is working to respond.
  - Processing consumes resources, which takes time.
  - Events are handled by element execution, whose time expanded is a resource.
  - Hardware resources: CPU, data stores, network bandwidth, memory.
  - Software resources: elements of the system.
- Each step of processing consumes time and resources, contributes to processing time.
- Saturated resources show increased performance degradation.

Blocked Time

- Computation can be blocked because of resource contention and unavailability or dependency on unfinished computations.
    - Contention: Many resources can only be used by one client at once. Other clients must wait.
    - Availability: Resource could be offline, an element could have failed, etc.
        - Must identify places where unavailability can contribute to latency.
    - Dependency: May need to synchronize with another computation.
        - Causes major performance issues over networks

Performance Tactics



Control Resource Demand

- Increase performance by managing demand for resources.
- Manage Sampling Rate
  - Reduce the frequency at which data is captured.
  - Decreases demand, but causes loss in fidelity.
  - Trade-off: lower fidelity, but consistent stream of data versus potential packet loss due to processing.
- Limit Event Response
  - If too many events arrive to be processed, queue them until they can be dealt with.
  - Can choose to process events up to a set maximum rate - ensuring predictable processing.

Control Resource Demand

- Limit Event Response
  - Can choose to process events up to a set maximum rate - ensuring predictable processing.
    - Can be triggered at set queue sizes or processor utilization percentage.
  - If unacceptable to lose any events, need to ensure the queues are large enough.
  - If you chose to drop events, need to choose a policy for handling them.
    - Logging dropped events or ignoring them?
    - Notifications to users, admins, systems?

Control Resource Demand

- Prioritize Events
    - If not all events are equally important, can prioritize by importance and service most important first.
    - If resources are limited, ignore low-priority events.
    - Ignoring events improves performance.
        - Building management: if there is a fire alarm, ignore non-critical warnings.
- Bound Execution Times
    - Place limits on how much execution time is used to respond to events.
    - For iterative algorithms, run fewer iterations.
    - Assess impact on result accuracy.

Control Resource Demand

- Reduce Overhead
  - Multiple elements often coordinate to perform tasks.
    - Intermediary elements pass data and coordinate control for parts of the system.
  - Element communication improves modifiability, but the overhead decreases performance.
    - Especially over a network.
  - Removing element communication improves latency
    - Combine elements, host on the same CPU.
  - Latency can also be improved by co-locating elements and data on the same CPU.
  - Periodically “clean” inefficient resources.
  - Execute multiple copies of processes and split work.

Manage Resources

- Resources management can be carefully controlled, even if demand is not reduced.
    - Trade one resource for another depending on time and resource quantity (caching vs regenerating)
- Increase Resources
    - Faster/more CPUs, more memory, faster network.
    - Costs $$$, but easiest and fastest way to improve performance.
        - Sometimes even the cheapest way to improve.

Manage Resources

- Introduce Concurrency
    - Blocked time can be reduced by executing requests in parallel.
    - Process different event streams on different threads.
        - Add threads for new activities.
    - Can schedule actions to maximize fairness (all requests get equal time) or throughput (shortest time to finish first).
- Maintain Clones of Elements
    - Reduce contention by cloning servers to replicate computations.
    - Assign work to available servers.

Manage Resources

- Maintain Copies of Data
  - Cache data on storage with different access speeds.
    - In-memory, stored to disk.
    - Locally or on a server.
  - Maintaining copies reduces contention from simultaneous access.
  - Need to keep all copies of data consistent.
  - Need to choose what to cache.
    - Recently requested items.
    - Predict based on user behavior and pre-cache.

Manage Resources

- Consolidate Related Workload
  - Many operations require contextual information.
  - Managing this information adds overhead when operation is small or context is expensive to load.
  - Consolidate related tasks into batches and process groups of related requests at once.
    - Single initialization, single tear-down.
  - Can also try to reuse operations or results between related jobs.
    - Can be considered as part of the architecture.

Manage Resources

- Bound Queue Sizes
  - Control number of queued arrivals, reducing resources needed to process the arrivals.
  - Need to decide what to do if queue fills.
    - Is it alright not to respond to lost events?
- Schedule Resources
  - If contention exists, resource access must be scheduled.
  - Processors, buffers, networks all operate on schedules.
  - Understand how to use each resource and choose an appropriate scheduling strategy.

Scheduling Policies

- All scheduling policies assign priorities.
  - Simple: FIFO (first-in, first-out)
  - Can be tied to deadline of a request or importance.
  - Competing criteria: optimal resource use, importance, minimization of used resources, minimization of latency, maximizing throughput, preventing resource starvation.
- Events are dispatched to resources.
  - Resource must be available.
  - Can require preempting the current user.
    - Can occur anytime, at specific points, or only if the process is allowed to preempted.

Scheduling Policies

- First-In, First-Out (FIFO)
    - All requests are treated equally.
    - All requests are satisfied in the order they arrive.
    - Easy to implement and manage.
    - One request can be stuck behind a slow request for a long time.
        - If all are **truly** equal, this is fine.
        - If some have higher priority, this is a problem.

Scheduling Policies

- Fixed-priority Scheduling
  - Assigned each request a priority and assigns resources in priority order.
    - Ensures better service for high priority requests.
    - Low-priority requests may never be completed.
  - **Semantic Importance:** Each stream assigned a static priority based on task characteristics.
  - **Deadline Monotonic:** Higher priority assigned to streams with shorter real-time deadlines.
  - **Rate Monotonic:** Higher priority assigned to streams that take less time to complete.

Scheduling Policies

- Requests can be assigned priorities dynamically as well.
  - **Round-Robin:**
    - Order all requests, and assign resources to the next request in that order.
  - **Earliest-Deadline-First:**
    - Assigns priorities based on the pending request with earliest deadline.
  - **Least-Slack-First:**
    - Assigns highest priority to job with the least “slack time” (difference between remaining execution time and time to job’s deadline)

Optimize Repeated Processing

- Systems spend 80% of time running 20% of code, so focus on optimizing that 20%.
  - Total Cost = Invocation Cost * Invocation Frequency
- Rank operations by their cost, and focus on the operations on top of that list.
- Watch for cases where optimizing one operation harms performance of another.
  - Message bus could create table of optimal routes.
  - Would speed route selection (frequent activity)
  - Trade-off: If a node or link is added, the table must be recalculated (expensive operation)

Performance Tactics in Action

- Road traffic engineers use the same tactics to improve highway systems.
  - Manage event rate (lights to control highway entrance, cars queue for their turn)
  - Prioritize events (emergency vehicles can change lights, some highways have HOV lanes)
  - Maintain copies (add traffic lanes to roads)
- Users can also employ tactics.
  - Increase resources (buy a faster car)
  - Increase efficiency (find a faster or shorter route)
  - Reduce overhead (driver closer to other cars, carpool)

Design Guidelines

Allocation of Responsibilities

- Determine system responsibilities that require heavy loading, critical response requirements, heavy use, or large impact.
  - For those, identify processing requirements of each.
  - Determine where bottlenecks exist.
  - Look for responsibilities that result from a thread of control crossing process or CPU boundaries.
  - Look for responsibilities that manage threads (allocation/deallocation, maintaining thread pools)
  - Look for responsibilities for scheduling shared resources and managing queues/buffers/caches.
    - For each, ensure requires response can be met.

Coordination Model

- Determine elements that must coordinate.
- Choose communication mechanisms that:
  - Support introduced concurrency, event prioritization, or scheduling.
  - Ensure that required performance response is met.
  - Can capture periodic, stochastic, or sporadic event arrivals, as needed.
  - Have appropriate properties:
    - Stateful vs stateless
    - Synchronous vs asynchronous.
    - Guaranteed delivery
    - Throughput or latency levels

Data Model

- Determine portions of the data model that will be heavily loaded, have time-critical response needs, are heavily used, or have wide impact.
- For each, determine:
    - Whether maintaining copies of data would help.
    - Whether partitioning data would help.
    - Whether reducing processing requirements for creation, initialization, persistence, manipulation, translation, or destruction is possible.
        - Or whether adding resources would help.

Mapping among Elements

- Where heavy network loading occurs, can co-locating elements reduce loading?
- Ensure that elements with heavy computation requirements are assigned to CPUs with most capacity.
- Consider introducing concurrency.
    - Split job across copies of an element.
- Determine whether choice of threads of control introduces bottlenecks.

Resource Management

- Which resources impact performance?
- Ensure they will be monitored and managed under normal and overloaded system operation.
    - System elements that manage time and other performance-critical resources.
    - Process/thread models.
    - Prioritization of resources and access to resources.
    - Scheduling and locking strategies.
    - Deploying additional resources on demand to meet increased loads.

Key Points

- Performance is about management of resources in the face of demand to achieve acceptable timing behavior.
  - Performance can be measured in terms of throughput and latency.
- Performance can be improved by reducing demand or by managing resources.
  - Reducing demand will have the side effect of reducing fidelity or missing some requests.
  - Managing resources can be done through scheduling, replication, or just increasing resources.

Key Points

- The ability to “grow” the system to process an increasing number of requests.
  - While still meeting performance requirements.
- How can we effectively utilize the additional resources?
- Effective scalability requires that:
  - Additional resources result in a performance improvement.
  - Did not require undue effort to add.
  - Did not disrupt operations.

Next Time

- Perspective: Security
  - Sources: Rozanski & Woods, Ch. 25
  - Bass, Clements, & Kazman, Ch. 9

- Homework:
  - Project, Part 3 - Due on Nov 18th
  - Assignment 3 - Due December 9th