

A Foundation for QoS-Aware Computing in Open Systems

Tarek Abdelzaher
Department of Computer Science
University of Virginia

A Little About Me...

- Ph.D. in QoS Adaptation in Real-Time Systems, Department of Computer Science, University of Michigan, 1999.
- 1999-today: Assistant Professor, Department of Computer Science, University of Virginia.
- Research Interests: Real-time Scheduling, QoS, Web Architecture, Application of Automatic Feedback Control to Computing Systems, Sensor Networks

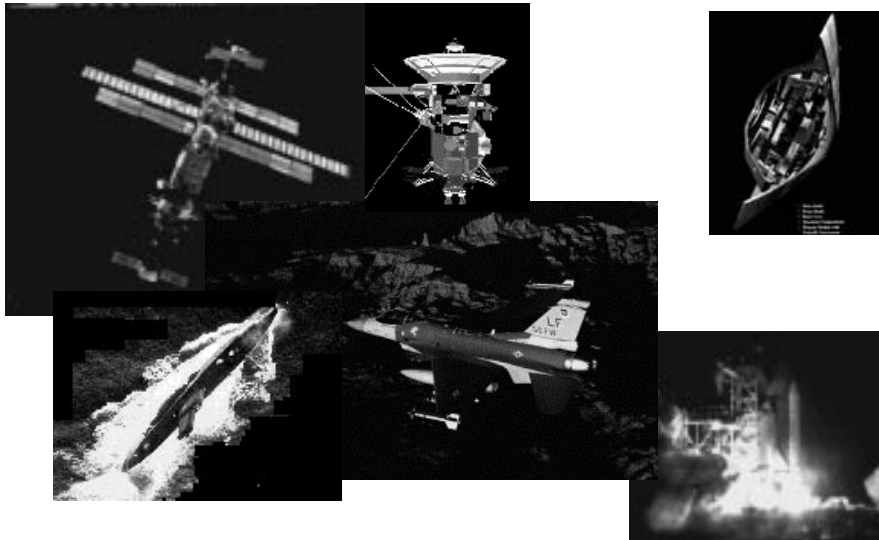
University of Virginia, Real-time Systems Laboratory

Outline of Presentation

- Part I: A New Paradigm for Real-Time Computing?
- Part II: Advances in Aperiodic Schedulability Bounds
 - A bound for aperiodic tasks
 - Extensions
- Part III: Software Performance Control
 - Concepts
 - Case Studies
 - Web Server Control
 - Adaptive Web Cache Control
 - ControlWare
- Part IV: Future Directions in Real-Time Computing

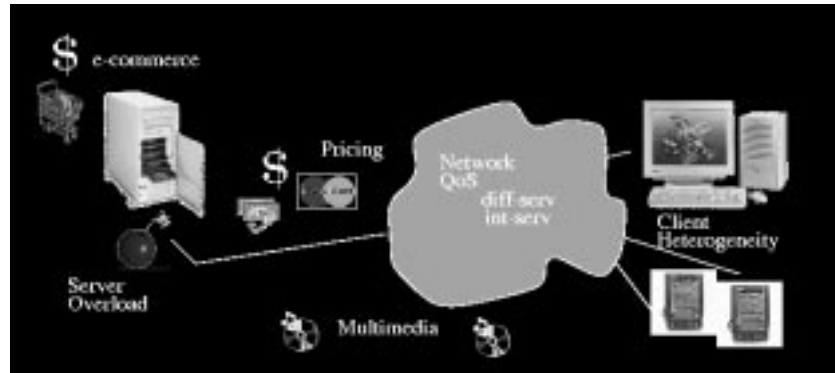
University of Virginia, Real-time Systems Laboratory

What is Real-Time Computing?



Timeliness, Predictability, Reliability, Embedded Computing

Recent Real-Time Applications

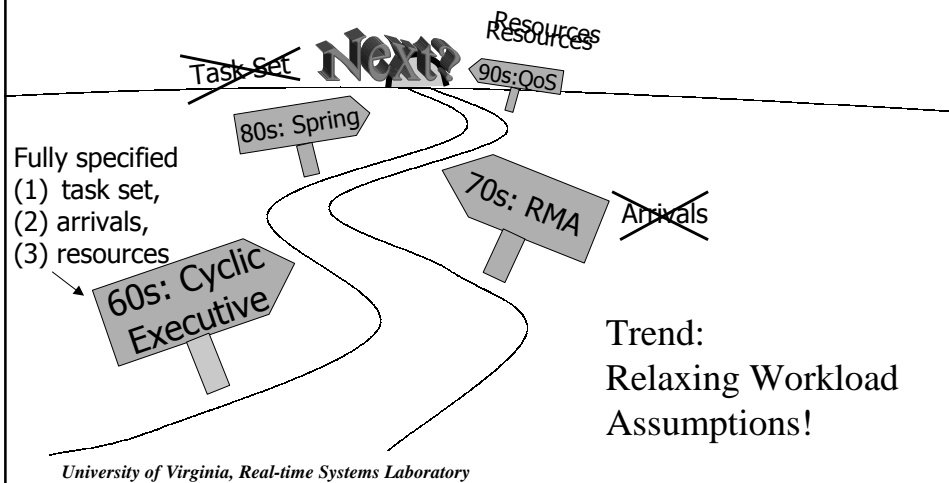


Open Systems, QoS, Service Performance Guarantees

Where is Real-Time Computing Going?

Identifying the Trends

The Real-Time Scheduling Theory Roadmap



The Changing Scope of Real-Time Computing

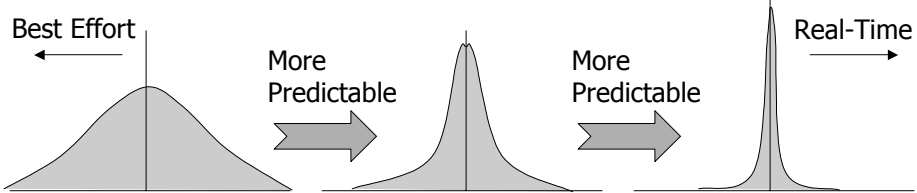
Classical View

- Definition: correctness of computation depends on the time at which results are generated.
- Applications: embedded systems
 - Perfect knowledge of resources
 - No conflicts of interest
- Assumptions: hard real-time
 - Meeting deadlines is very important for correctness
 - Deadlines are inflexible

University of Virginia, Real-time Systems Laboratory

New Real-Time Definitions?

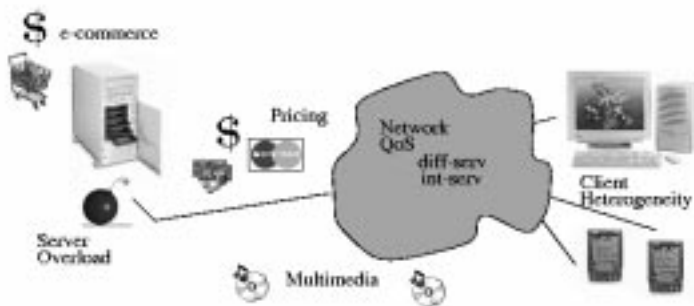
- Systems with low variance in performance measurements



- Systems with guaranteed metrics in which time is either in the *numerator* or in the *denominator*
 - Response time = Service time + Queuing time
 - Utilization = Used resource units/time
 - Service throughput = Produced data units/time

University of Virginia, Real-time Systems Laboratory

New Real-Time Applications (with **Unknown** Resource Demands)




- Open Systems
 - Interactions with unknown components
- Data-centric
 - Data-dependent resource requirements
 - Data pattern is unknown
- Platform-independent



New Assumptions

- Unknown resource requirements
- Elastic time constraints
 - Flexible periods (if any)
 - Flexible deadlines
- Adaptation capability
 - Changing algorithm version
 - Adaptive data quality
- Steady-state versus transient metrics

University of Virginia, Real-time Systems Laboratory



This Talk: Guarantees in Server End-Systems

- Applications:
 - Web servers, proxy servers, mail servers, DNS servers, routers, database servers, ...
- Guarantees:
 - Delay, throughput, ratio, utilization, ...
- Workloads:
 - Highly variable, no periodicity
- Adaptation:
 - Content-based, resolution, frame rate, compression

University of Virginia, Real-time Systems Laboratory

Problem with Current Real-Time Scheduling Theory

- Modern QoS-aware systems have large uncertainty in workload and resource availability
- Classical real-time scheduling is not robust under uncertainty
 - Fine-grained assumptions about task computational requirements
 - Fine-grained assumptions about resource capacity
- How to control performance of open QoS-aware systems under load and resource uncertainty?

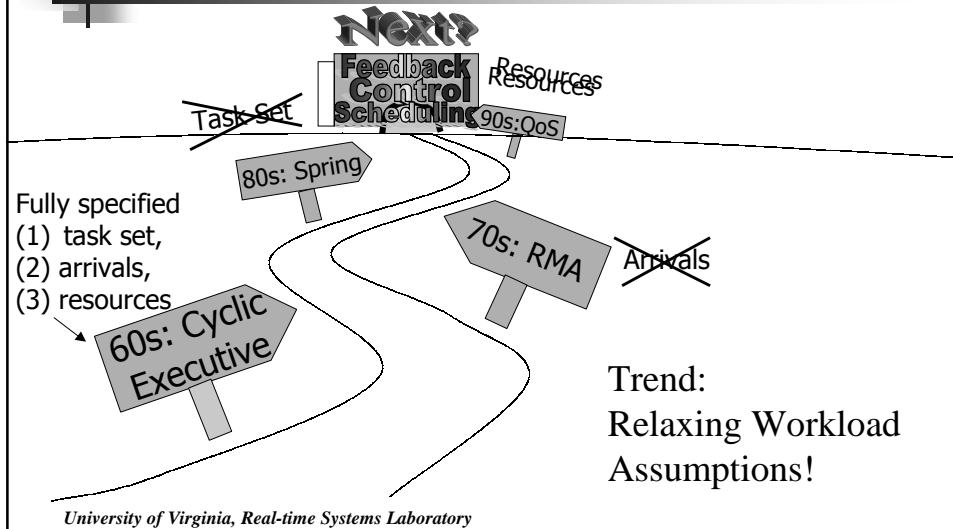
University of Virginia, Real-time Systems Laboratory

A Framework for Robust QoS

- Challenges:
 - Make the least/weakest assumptions regarding workload parameters (e.g., arrival times, execution times, deadlines, and number of arrived tasks)
 - Control perturbations which may violate these assumptions
- Direction:
 - Move from an “open-loop” to a “closed-loop” QoS provisioning paradigm
 - Drop the *a priori* assumptions; use continuous feedback for performance correction
 - Use feedback control theory to analyze closed loop performance

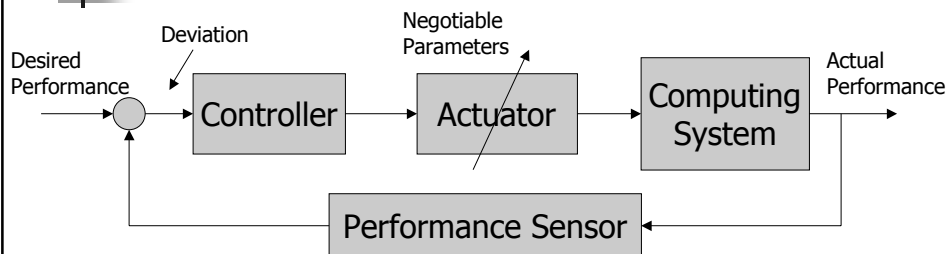
University of Virginia, Real-time Systems Laboratory

The Real-Time Scheduling Theory Roadmap



What is Control Theory?

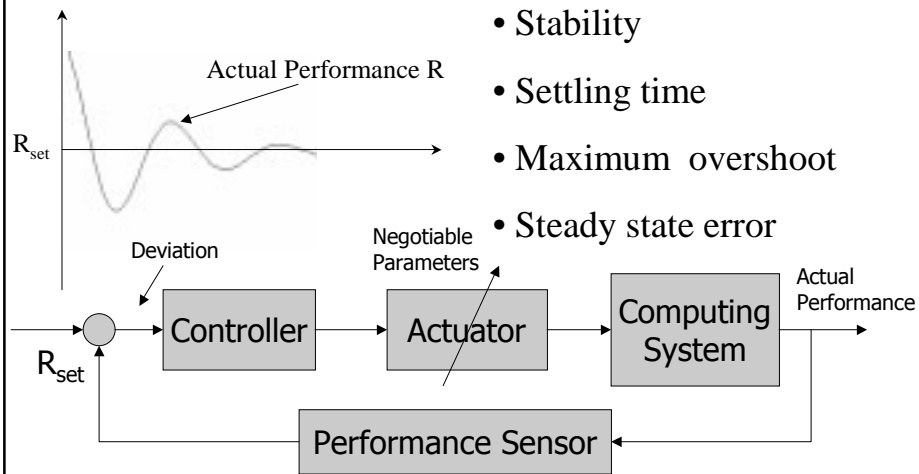
Control Loop Operation:



- Model the controlled process by a *differential equation*
- Feedback control is a process of continuous monitoring and correction of deviation from desired performance
- Control theory is involved with controller design

University of Virginia, Real-time Systems Laboratory

Convergence Guarantees



University of Virginia, Real-time Systems Laboratory

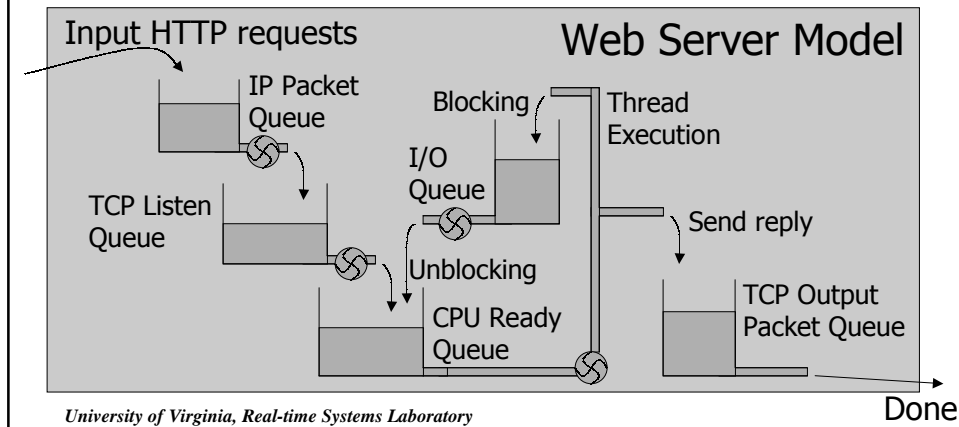
Why Control Theory?

- Successful track record in physical process control
- Robust performance guarantees in the face of uncertainty, non-linearities, time-variations, etc.
- Does not require accurate system models
- Utilizes feedback to improve performance
- Software services involve queue-induced dynamics which may be expressed by differential equations akin to those of physical systems

University of Virginia, Real-time Systems Laboratory

Example: Web Server Modeling

Why web servers can be modeled by difference equations!



Why Control Theory? - Comparisons

- Several theoretical foundations have been used for resource allocation in real-time systems
- Compare control theory versus
 - real-time scheduling
 - queuing theory
 - optimization

University of Virginia, Real-time Systems Laboratory

Feedback Control Versus Scheduling

- Scheduling
 - Open loop
 - Assumes accurate models of worst-case computation times and resource requirements
- Control theory
 - Closed loop
 - No need for accurate models

University of Virginia, Real-time Systems Laboratory

Feedback Control Versus Queuing Theory

- Queuing theory
 - Off-line predictive analysis
 - Assumptions about the arrival process
 - Difficult to analyze some distributions
- Control-theory
 - On-line input/output difference equations
 - No assumptions about the arrival process
 - Utilize run-time feedback for error correction

University of Virginia, Real-time Systems Laboratory

Feedback Control Versus Optimization

- Optimization
 - Works best if the performance problem is formulated as one of maximizing or minimizing some metric
- Control-theory
 - Works well if the performance problem is one of maintaining an invariant, or is a tradeoff between two conflicting metrics
 - Optimization problems can be cast as finding an equilibrium where:
 - Marginal cost = Marginal reward

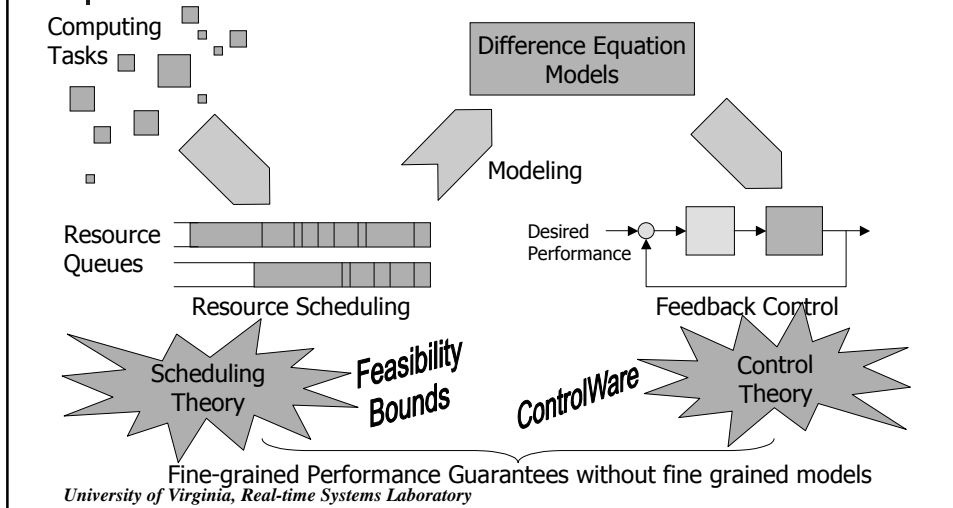
University of Virginia, Real-time Systems Laboratory

Problem with Control Theory

- Control is exerted on aggregate metrics
 - Can't have a control loop per task
 - Set point must be on global/average state
- Guarantees are needed on a per-task basis
- Real-time scheduling theory must bridge the gap such that guarantees on aggregate state imply guarantees to individual tasks

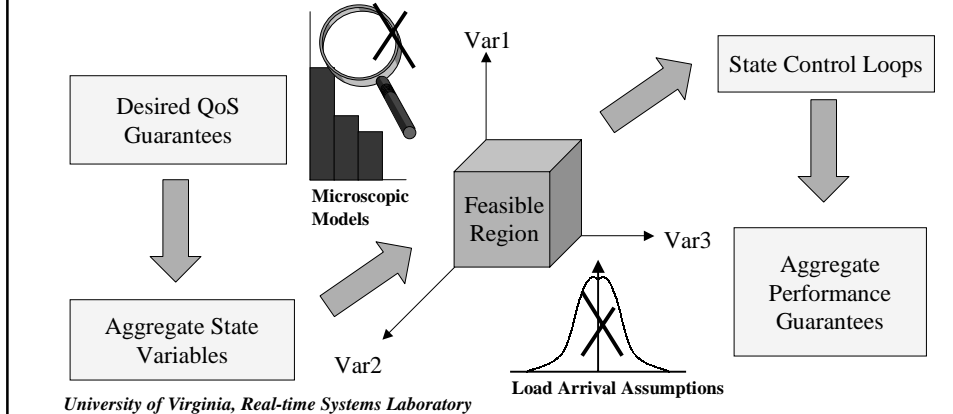
University of Virginia, Real-time Systems Laboratory

Theoretical Elements of Robust Server QoS Control



Robust Guarantees Under Uncertainty

- Achieving desired temporal behavior without fine-grained knowledge



Summary: Research Directions

- Thrust I: Extending schedulability theory to deal with inexact or unknown resource requirements, such that:
 - Guarantees on aggregate state \rightarrow guarantees on individual task deadlines
- Thrust II: Applying results from feedback control theory to control aggregate software state to achieve desired performance

University of Virginia, Real-time Systems Laboratory

Outline of Presentation

- A New Paradigm for Real-Time Computing?
- Advances in Aperiodic Schedulability Bounds
 - A bound for aperiodic tasks
 - Extensions
- Software Performance Control
 - Concepts
 - Case Studies
 - Web Server Control
 - Adaptive Web Cache Control
 - ControlWare
- Future Directions in Real-Time Computing

University of Virginia, Real-time Systems Laboratory

Aperiodic Schedulability Bounds

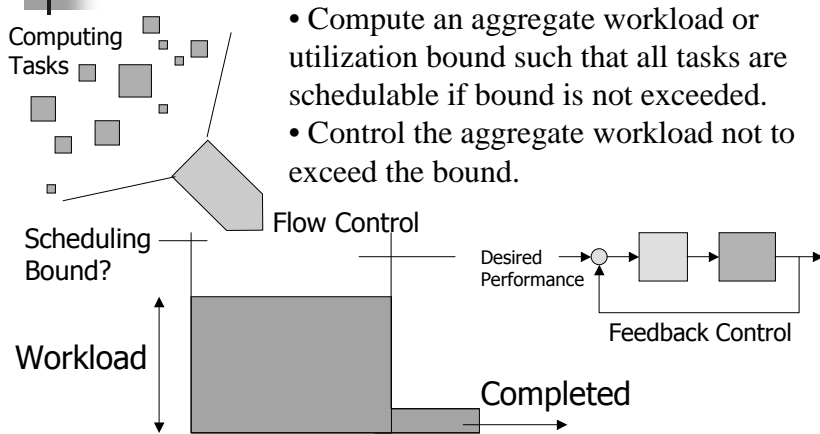
Uniprocessor Results
Liquid Task Model

Aperiodic Schedulability Bounds

- Consider a web server serving randomly arriving web requests
- Each request has a deadline
- Can one invent an aggregate measurable utilization-like metric (we call it synthetic utilization, U), such that all deadlines are met as long as U is below some threshold, U_{max} ?
 - Feasible region: $0 < U < U_{max}$
- If so, synthetic utilization control guarantees meeting all deadlines

University of Virginia, Real-time Systems Laboratory

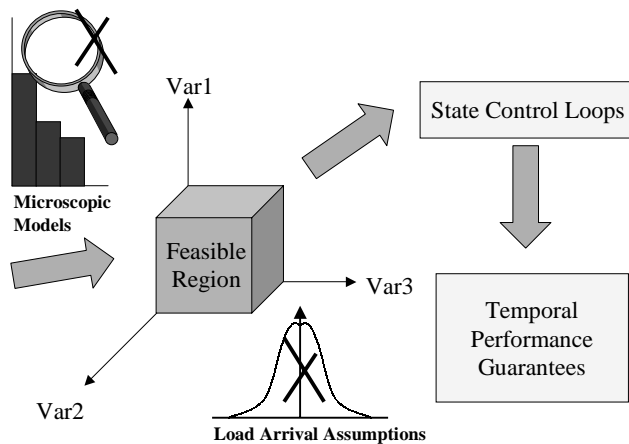
General Idea



University of Virginia, Real-time Systems Laboratory

Outline

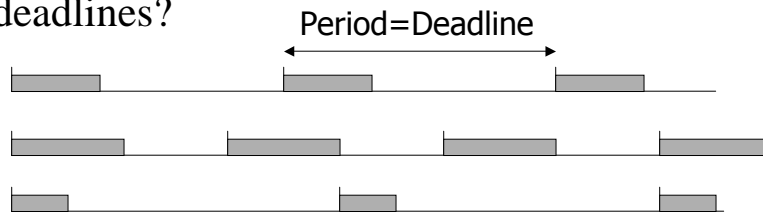
- Compute the “feasible region” of a single processor under a random load arrival pattern.
- This feasible region has one dimension: synthetic utilization!
- Extend to multiple processors
- Extend to an arbitrary scheduling policy



University of Virginia, Real-time Systems Laboratory

Historical Perspective: Utilization Bounds

- Consider a set of periodic tasks
- Each task invocation must finish by the end of its period
- How to tell if all invocations will meet their deadlines?



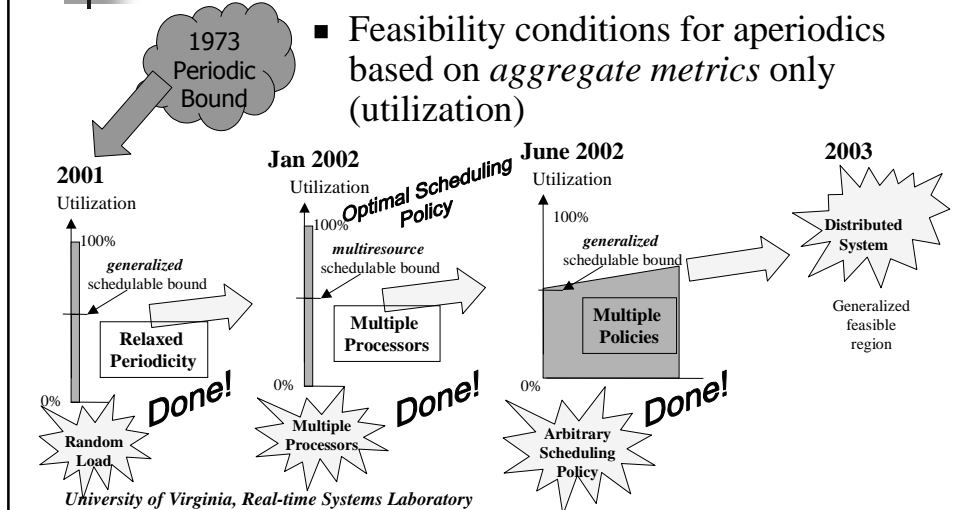
University of Virginia, Real-time Systems Laboratory

Liu and Layland Utilization Bound

- Well-known result (1973):
 - Assume that each task T_i executes for C_i every period P_i .
 - Processor utilization needed for this task is:
$$U_i = C_i/P_i$$
 - The task set is schedulable by an optimal fixed-priority scheduling policy if $\sum_i C_i/P_i < 0.69$
 - Optimal fixed priority policy is rate-monotonic (higher rate = higher priority)

University of Virginia, Real-time Systems Laboratory

Recent Results

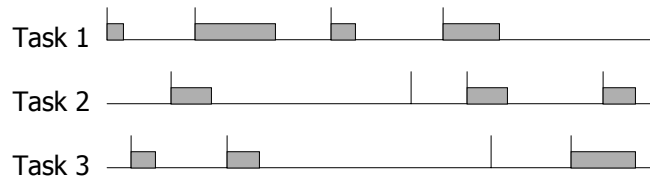


Rethinking the Basic Concepts

- The task model
- The notion of synthetic utilization
- The classification of scheduling policies
- The sense of optimality of a real-time scheduling policy

The Task Model: Acyclic Tasks

- Tasks have multiple invocations
- Deadline of one invocation is the arrival time of the next
- **No relation** between the execution times and deadlines of successive invocations of a task
- Execution times are greater than or **equal to zero**



University of Virginia, Real-time Systems Laboratory

Equivalence Result

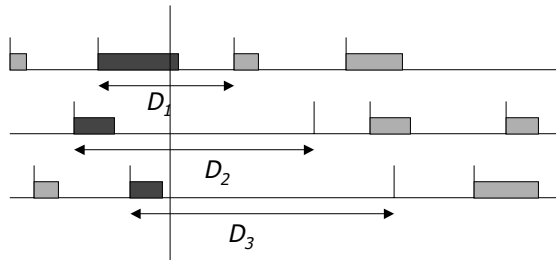
- Any set of aperiodic jobs can be converted into an equivalent set of acyclic tasks, and vice versa
- Utilization bound for acyclic tasks = utilization bound for aperiodic tasks

University of Virginia, Real-time Systems Laboratory

The Notion of *Synthetic* Utilization

- Synthetic utilization $U(t)$ is a function of time, t
- $U(t)$ is defined over the *current* invocations

$$U(t) = \sum_i C_i/D_i$$

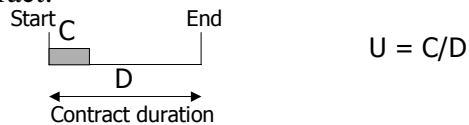


University of Virginia, Real-time Systems Laboratory

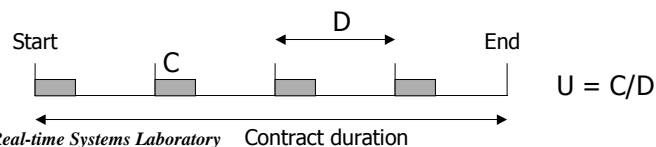
Schedulability Analysis

- Model: a mix of periodic and aperiodic tasks
- Contracts: All task invocation deadlines are met for duration of contract

- Aperiodic Task Contract:



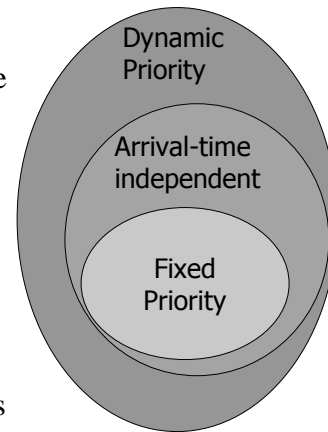
- Finite Periodic Task Contract:



University of Virginia, Real-time Systems Laboratory

Arrival-Time-Independent Scheduling

- **Fixed-priority scheduling:**
 - All invocations of a task have same priority
- **Dynamic-priority scheduling:**
 - Invocation priorities are assigned arbitrarily
- **Arrival-time-independent scheduling:**
 - Invocation priorities are not a function of invocation arrival times



University of Virginia, Real-time Systems Laboratory

Why Arrival-Time Independent Scheduling?

- Easy to implement on current non-real-time operating systems with fixed-priority support (e.g., UNIX, the #1 OS for web servers)
 - Requires a finite number of priority levels
- If priority depends on time, an infinite number of priority levels may be needed (e.g., EDF)

University of Virginia, Real-time Systems Laboratory

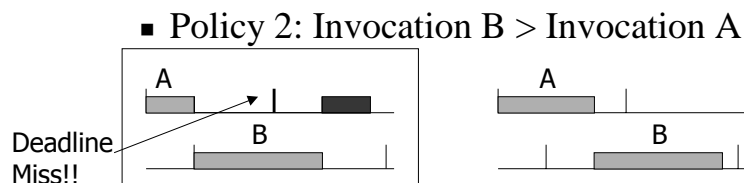
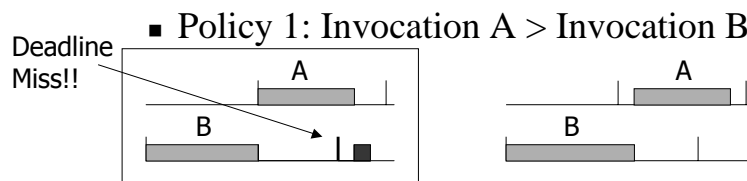
The Sense of Optimality of a Scheduling Policy

- Traditional sense of optimality:
 - A task set is schedulable by an optimal policy in a class whenever it is schedulable by some other policy of the same class.
- No arrival-time-independent scheduling policy is optimal for aperiodic tasks

University of Virginia, Real-time Systems Laboratory

Lack of an Optimal Policy

- Example:



University of Virginia, Real-time Systems Laboratory

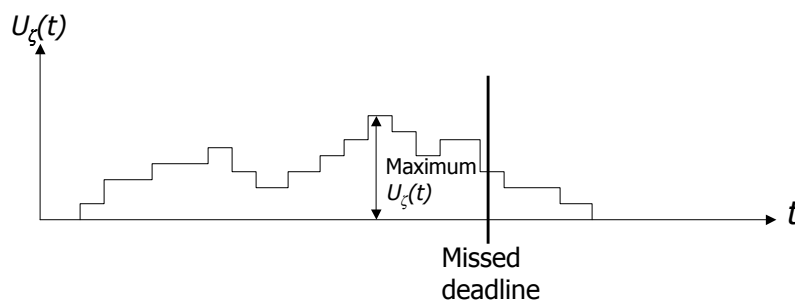
Generalized Sense of Optimality

- A scheduling policy is optimal in a class if it maximizes the schedulable synthetic utilization bound among all policies in the class
- “Backward Compatibility”:
 - Rate monotonic is the optimal fixed-priority policy (for periodic tasks)
 - EDF is optimal dynamic-priority policy
 - *New*: Deadline monotonic is the optimal arrival-time independent policy

University of Virginia, Real-time Systems Laboratory

Main Idea of Derivation

- **Minimize**, over all arrival patterns ζ , the maximum $U_\zeta(t)$ that precedes a missed deadline



University of Virginia, Real-time Systems Laboratory

Steps of Derivation

- Consider an unschedulable task invocation in a busy period
- Within the busy period minimize the maximum $U(t)$ w.r.t.:
 - Task invocation execution times
 - Task invocation arrival times
 - Task invocation deadlines
- Show that an arbitrary policy is either deadline monotonic or has a higher utilization bound

University of Virginia, Real-time Systems Laboratory

Main Results

- A set of n acyclic tasks is schedulable using an optimal arrival-time-independent policy if:

$$U(t) \leq \frac{1}{2} + \frac{1}{2n} \quad n < 3$$

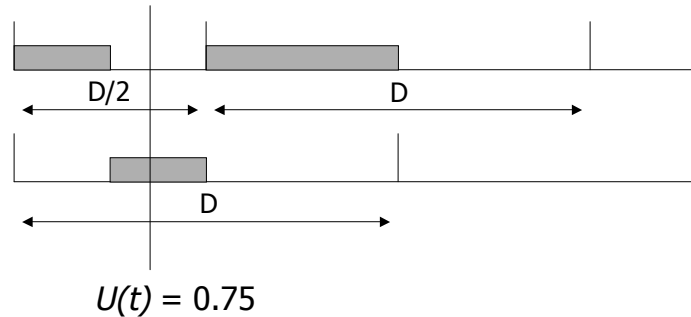
$$U(t) \leq \frac{1}{1 + \sqrt{\frac{1}{2}(1 - \frac{1}{n-1})}} \quad n \geq 3$$

- Deadline-monotonic scheduling is the optimal arrival-time independent policy for acyclic tasks

University of Virginia, Real-time Systems Laboratory

Example of Tightness

- Consider $n=2$



University of Virginia, Real-time Systems Laboratory

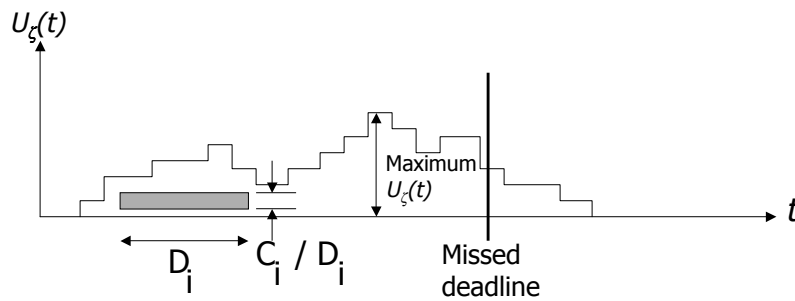
Extensions

- Multiprocessor Model
- Liquid Tasks
- Arbitrary Scheduling Policy

University of Virginia, Real-time Systems Laboratory

Quick-and-Dirty Multiprocessor Derivation

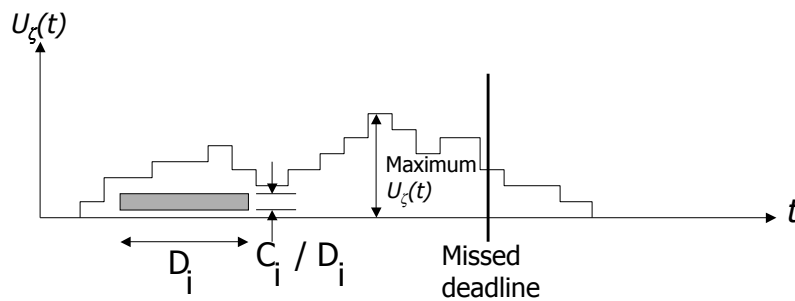
- Observe that each task T_i contributes C_i to the area under the $U_\zeta(t)$ curve – see figure below.



University of Virginia, Real-time Systems Laboratory

Corollary

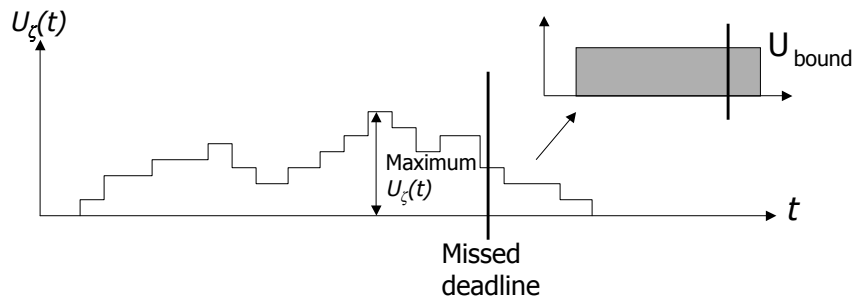
- The total area under the $U_\zeta(t)$ curve is $\sum C_i$ carried over all arrived tasks



University of Virginia, Real-time Systems Laboratory

Derivation

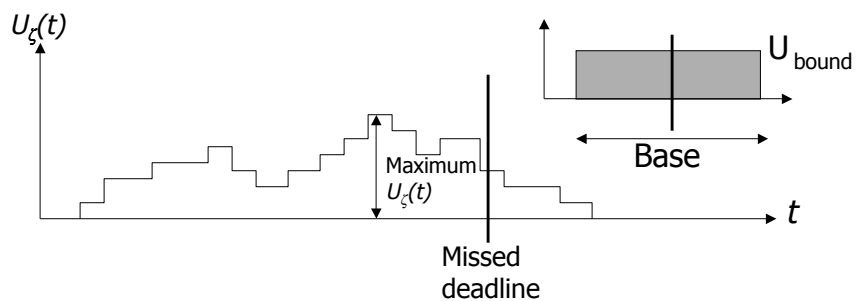
- **Minimize**, the sum ΣC_i across all unschedulable patterns. Say minimum is C_{\min}
- Minimize curve height while area = C_{\min}



University of Virginia, Real-time Systems Laboratory

Example: Deadline Monotonic Bound for Multiprocessors

- $C_{\min} = C_i + m (D_i - C_i)$
- $\text{Base}_{\max} = 2 D_i$



University of Virginia, Real-time Systems Laboratory

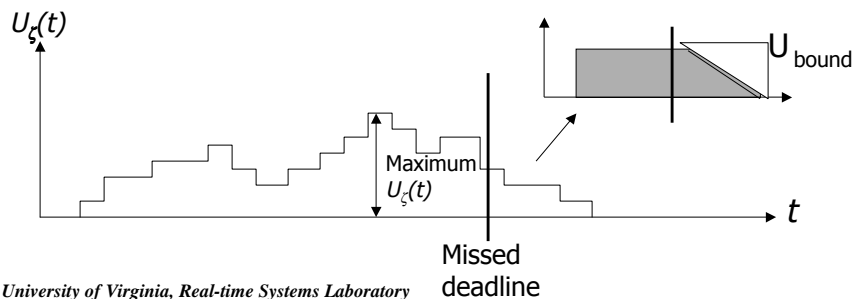
Example Continued

- $C_{\min} = C_i + m (D_i - C_i)$
- $\text{Base}_{\max} = 2 D_i$
- $U_{\text{bound}} = (C_{\min} / \text{Base}_{\max}) / m = C_i / (2mD_i) + (D_i - C_i) / (2D_i)$
- When $m \rightarrow \text{infinity}$, and $C_i \rightarrow 0$, the bound $\rightarrow 0.5$

University of Virginia, Real-time Systems Laboratory

Derivation Refined

- **Minimize**, the sum ΣC_i across all unschedulable patterns. Say minimum is C_{\min}
- Minimize the hight while area = C_{\min} *subject to constraints on utilization curve*



University of Virginia, Real-time Systems Laboratory

The Liquid Task Model

- A convenient model for server workload modeling
- Generally, each request has a deadline chosen from a small set of values (e.g., one per class)
- In the liquid task model, all requests have different computation times, but $C_i/D_i \rightarrow 0$
- Reflects the case of high-performance servers where any individual request consumes an infinitesimal fraction of server capacity.

University of Virginia, Real-time Systems Laboratory

Main Results

- A set of aperiodic *liquid* tasks is schedulable using an optimal arrival-time-independent policy if:

$$U(t) \leq \frac{1}{1 + \sqrt{\frac{1}{2}}}$$

- Deadline-monotonic scheduling is the optimal arrival-time independent policy for liquid tasks independently of the number of processors

University of Virginia, Real-time Systems Laboratory

Main Results (continued)

- A set of randomly arriving requests will meet their delay bounds under priority-based scheduling if:

$$U(t) \leq 1 + \beta - \sqrt{1 + \beta^2}$$

- β is a property of the scheduling policy (the minimum ratio of deadline of low priority task to deadline of high priority task which can preempt it).
- $U(t)$ is synthetic utilization

University of Virginia, Real-time Systems Laboratory

Monitoring Synthetic Utilization

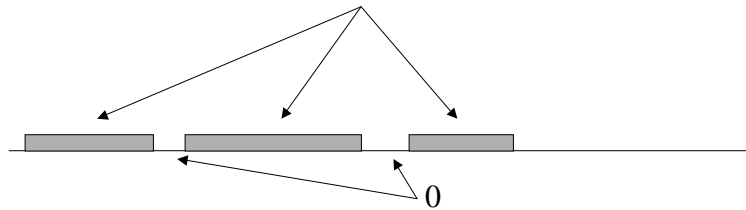
- Aperiodic Task Contracts:
 - On arrival: $U_a = U_a + (C/D)_{contract}$
 - On expiration: $U_a = U_a - (C/D)_{contract}$
- Periodic Task Contracts:
 - On arrival: $U_p = U_p + (C/D)_{contract}$
 - On expiration: $U_p = U_p - (C/D)_{contract}$
- Admission test: $U_a + U_p < U_{bound}$
- On Processor Idle Time: set $U_a = 0$

University of Virginia, Real-time Systems Laboratory

Synthetic versus Real Utilization

- An important property is that:
synthetic utilization < real utilization.
→ Hence, server is not underutilized!
- Proof:

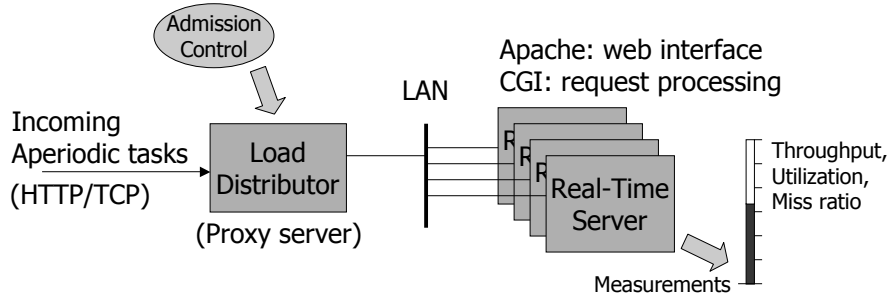
Real utilization = 1
Synthetic utilization < 0.58



University of Virginia, Real-time Systems Laboratory

Experimental Testbed and Evaluation

- A real-time computing cluster is developed under Linux
- 4 worker Linux PCs connected by 100Mbps LAN to a front-end load distributor

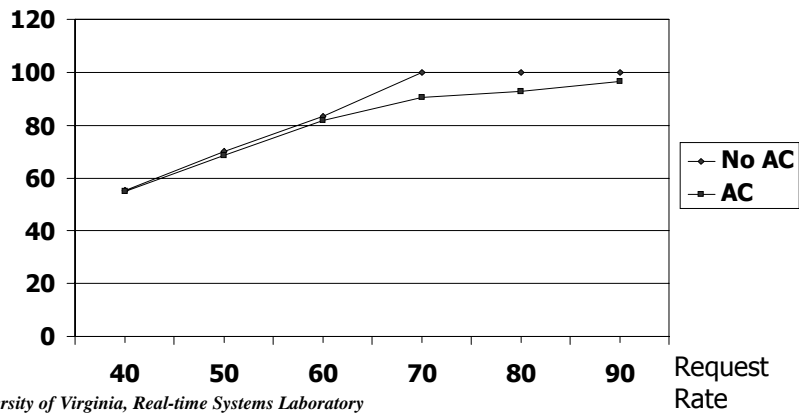


University of Virginia, Real-time Systems Laboratory

Server Utilization

Utilization-based admission control does not underutilize server

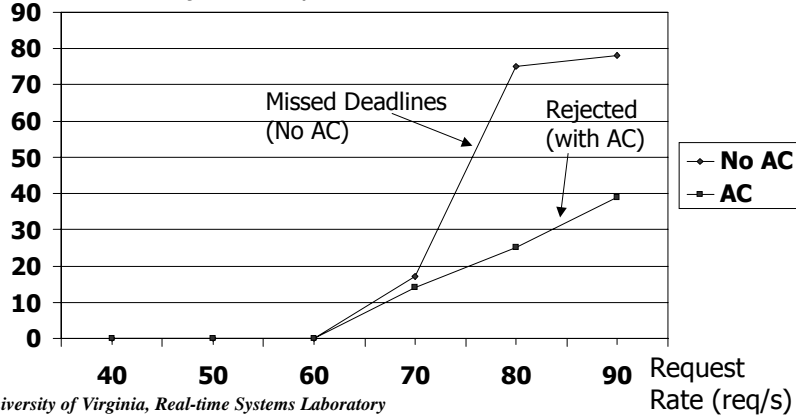
Cluster Utilization



Miss Ratio Profile

Utilization-based admission control improves task success ratio

% missed or rejected requests





Summary

- If synthetic utilization is maintained around 0.58, then:
 - All deadlines will be met
 - Real utilization will be high
- Control theory can be used to maintain any performance metric around a desired value
 - Synthetic utilization set point < 0.58

University of Virginia, Real-time Systems Laboratory



Further Information

- Tarek Abdelzaher and Chenyang Lu “An Aperiodic Utilization Bound for High-Performance Services,” RTAS 2001, TaiPei, Taiwan, June 2001
- Tarek Abdelzaher, Bjorn Andersson, Jan Jonsson, “A Multiprocessor Utilization Bound for Liquid Aperiodic Tasks”, RTAS 2002, San Jose, CA, Sept 2002

University of Virginia, Real-time Systems Laboratory



Outline of Presentation

- A New Paradigm for Real-Time Computing?
- Advances in Aperiodic Schedulability Bounds
 - A bound for aperiodic tasks
 - Extensions
- Software Performance Control
 - Concepts
 - Case Studies
 - Web Server Control
 - Adaptive Web Cache Control
 - ControlWare
- Future Directions in Real-Time Computing

University of Virginia, Real-time Systems Laboratory



Control Theory for Software Performance Control

Towards more robust QoS

Scope of Application of Control Theory

- Any system that can be described by linear differential equations is a candidate for control-theoretic analysis
 - Mechanical systems
 - Electric circuits
 - Temperature control systems
 - Hydraulic systems
 - Flow/level control in water pipes/tanks
- Desired performance is achieved by utilizing feedback

University of Virginia, Real-time Systems Laboratory

The Concept of Software Feedback

- Feedback control has been used in software in different ways
 - Operating systems: UNIX scheduler
 - Networks: TCP congestion control
- Formal feedback control theory was generally not used
 - Software systems are very nonlinear ad hoc systems, generally not amenable to an analytic formulation

University of Virginia, Real-time Systems Laboratory

Can Software be Modeled by Difference Equations?

- Time-related software performance metrics such as response time and queuing delay depend on *queue state dynamics*, e.g., of:
 - The CPU ready queue
 - Semaphore queues
 - Socket queues
 - I/O device queues
- A queue acts as a capacitor (a water tank) which integrates the difference between input and output flows, and hence can be modeled by a differential equation

University of Virginia, Real-time Systems Laboratory

Examples of Control Theory Application in Computer Science

- Network flow control (TCP/IP - RED)
 - C. Hollot et al. (U.Mass, INFOCOM 2001)
- Lotus Notes admission control
 - S. Parekh et al. (IBM, IEEE ISINM 2001)
- Apache Server Utilization Control
 - T. F. Abdelzaher et al. (UVA, IEEE TPDS 2001)
- Apache QoS differentiation
 - C. Lu et al. (UVA, IEEE RTAS 2001)

University of Virginia, Real-time Systems Laboratory

What is Software Feedback Control Used for?

- In a computing system, feedback control iteratively adjusts *resource allocation* to approach desired performance, e.g.:
 - to maintain a given server response time, or
 - to achieve a given server utilization
 - To maintain a given performance ratio
- Sensors: measure current software performance
- Actuators: manipulate resource allocation to affect performance

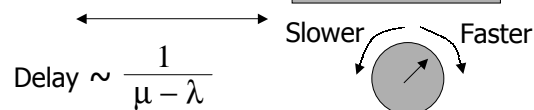
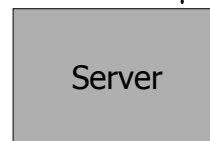
University of Virginia, Real-time Systems Laboratory

Example

- Control server resource allocation to achieve a given response time (e.g., 2s)
 - Delay > 2s $\rightarrow \mu \uparrow$
 - Delay < 2s $\rightarrow \mu \downarrow$

Arriving Requests
Request rate = λ

Service rate = $\mu > \lambda$



Hardware clock frequency
And voltage control

University of Virginia, Real-time Systems Laboratory

When to use Feedback Control?

- Real-time schedulability theory, queuing theory, and optimization algorithms define accurate conditions which can be used for resource allocation and admission control to achieve predictable desired performance.
- Feedback control is useful when the above cannot be applied due to uncertainty in load and resource parameters

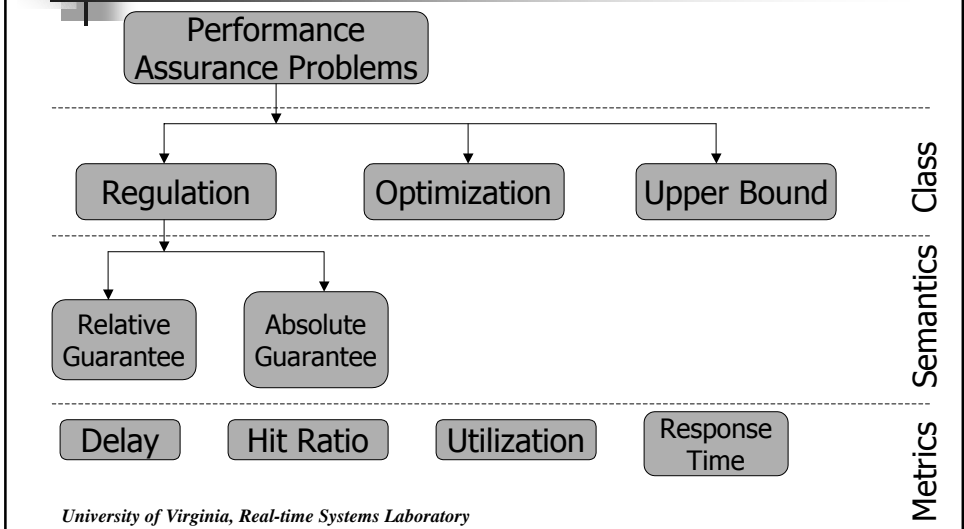
University of Virginia, Real-time Systems Laboratory

A Methodology for Applying Feedback Control to Software

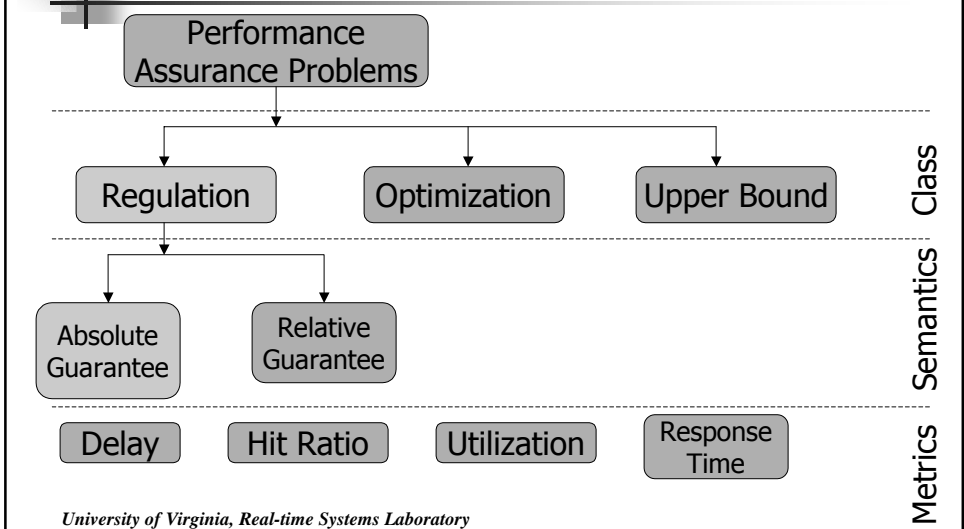
- QoS Mapping: QoS problem \rightarrow Feedback problem
 - Determine the controlled performance metric (*output*) and its desired value (*set point*)
 - Determine the available actuators (*input*)
 - Map the QoS control problem into a set of control loops
- Modeling:
 - Model the input-output relation as a difference equation:
$$Output_k = \sum_i a_i Output_{k-i} + \sum_i b_i Input_{k-i}$$
- Controller design:
 - Use control theory to find function f , such that:
$$Input = f(set\ point - output) \text{ makes } output \rightarrow set\ point$$

University of Virginia, Real-time Systems Laboratory

QoS Guarantees as a Feedback Problem: A Taxonomy



QoS Guarantees as a Feedback Problem: Absolute Guarantees



Regulation Problems: Absolute Guarantees

- Absolute guarantees:
 - Server guarantees a specific level of performance for a particular class of clients
 - Example 1: Admission control to maintain server queue length at 10 requests
 - Length $> 10 \rightarrow$ overload
 - Length $< 10 \rightarrow$ underutilization
 - Example 2: Voltage/frequency scaling to maintain server delay at 2sec.
 - Delay $> 2s \rightarrow$ overload
 - Delay $< 2s \rightarrow$ too much power consumption

University of Virginia, Real-time Systems Laboratory

Absolute Guarantee Problems

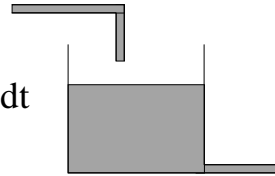
- Level control (queue length, utilization, ...)
- Delay control (response time, wait time, ...)

What is the fundamental difference?

University of Virginia, Real-time Systems Laboratory

Absolute Guarantee Problems

- Level control (queue, utilization, ...)
 - Linear
 - Liquid tank model
 - $\text{Level} = \int (\text{Flow}_{\text{in}} - \text{Flow}_{\text{out}})dt$
- Delay control
 - Nonlinear
 - Queuing dynamics model
 - $\text{Delay} = \int dt / \text{Flow}_{\text{out}}$



University of Virginia, Real-time Systems Laboratory

A Case Study on Level Control: Utilization Control in Web Servers*

Utilization control can be used for:

- Overload avoidance
- Performance isolation among hosted sites
- Prioritization among client classes
- Capacity sharing

*This study has been conducted at Hewlett Packard Labs, Palo Alto, USA, by Tarek Abdelzaher and Nina Bhatti

University of Virginia, Real-time Systems Laboratory

Overload Avoidance

- Goal: maintain server utilization around the maximum load threshold (e.g., 85%)
 - Above 85% → overload
 - Below 85% → underutilization
- Load is controlled by adapting web content
 - Multiple versions of content are available *a priori*
 - Server load is monitored
 - Delivered version is chosen based on load condition
 - Degrading content quality decreases utilization

University of Virginia, Real-time Systems Laboratory



Example: Image Compression

```

```



74KB GIF
High QoS



8.4 KB JPEG
Low QoS

University of Virginia, Real-time Systems Laboratory



Why Feedback Control?

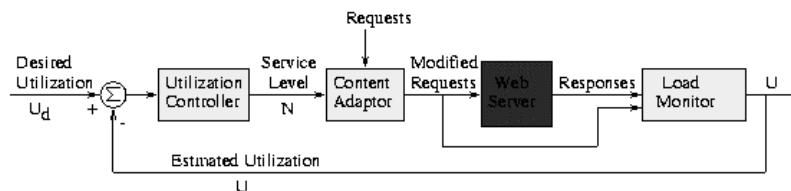
- Utilization-based admission control can maintain server utilization without feedback control, e.g., as follows:
 - Utilization < 70% → admit more requests (at high QoS)
 - 70% < Utilization < 85% → admit more requests (at low QoS)
 - Utilization > 85% → stop admitting requests
- Problem:
 - Per request-admission control is inferior to per-client admission control
 - Don't know the additional load per admitted client. How many more *clients* to admit?
 - Need a feedback-based “trial and error” approach

University of Virginia, Real-time Systems Laboratory



Utilization Control Loop

- Load Monitor (measures utilization)
- Utilization Controller (determines degradation)
- Content Adaptor (serves appropriate content version)



University of Virginia, Real-time Systems Laboratory



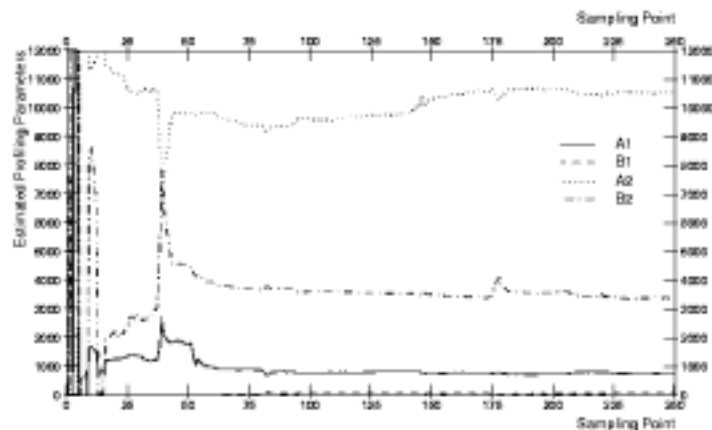
Designing the Sensor: A Model for Utilization

- OS utilization measurements are very noisy
- Alternative:
 - Apache server was subjected to a variable request rate for static and dynamic content of different sizes
 - At each sampling time we measured
 - Request rate for static content, R_{static}
 - Byte rate of static content sent, W_{static}
 - Request rate for dynamic content, R_{dyn}
 - Byte rate of dynamic content sent, W_{dyn}
 - Recursive least squares estimator was used to correlate measurements to server utilization

University of Virginia, Real-time Systems Laboratory

A Model for Utilization

$$U = 0.75 R_{static} + 0.035 W_{static} + 10.4 R_{dyn} + 3.3 W_{dyn}$$



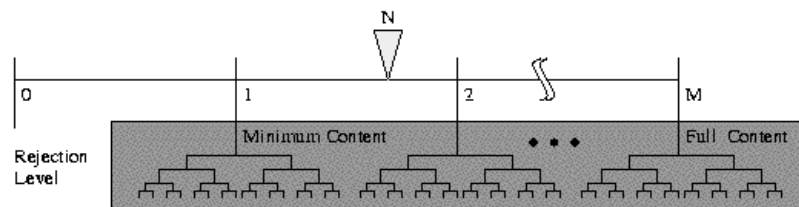
Designing the Actuator

- Actuator chooses content quality
- On the surface, it seems that only a finite number of actuator outputs are possible (e.g., good content versus degraded content)
- Linear feedback control requires a continuous “content degradation” range.
- How to resolve the above conflict?

University of Virginia, Real-time Systems Laboratory

Continuous Content Adaptor

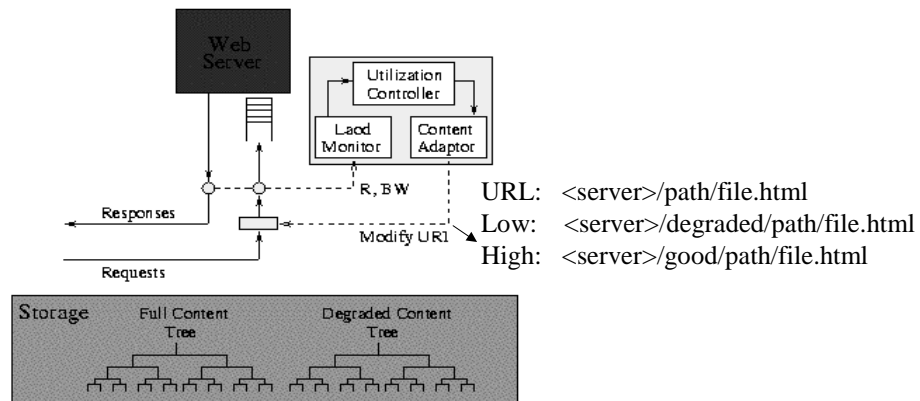
- Degrades or rejects a fraction of requests depending on controller output, N .



University of Virginia, Real-time Systems Laboratory



The Overload Avoidance Loop

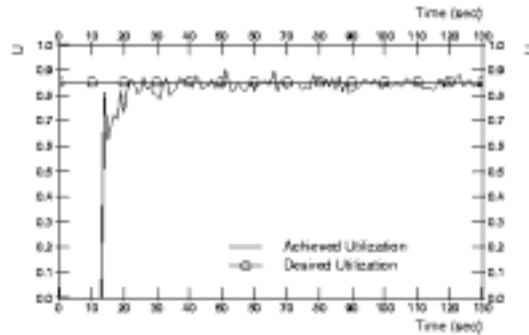


University of Virginia, Real-time Systems Laboratory



Experiment: Efficacy of Utilization Control

- Offered Load = 300% (at t=13)
- Desired utilization = 85%



Controlling Server Utilization

University of Virginia, Real-time Systems Laboratory



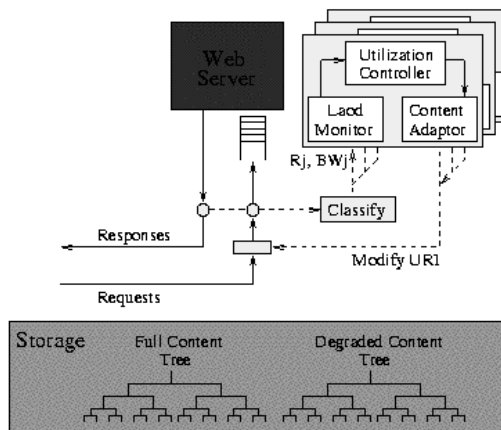
Application of Utilization Control in Web Servers

- Performance isolation
 - Server resources can be partitioned among hosted sites such that two web sites hosted on the same server have their own server shares
- Prioritization
 - The server can prioritize clients without priority-based scheduling (e.g., on standard UNIX platforms)
- Excess capacity sharing
 - Performance isolation may be extended to allow sharing available excess capacity

University of Virginia, Real-time Systems Laboratory



Performance Isolation



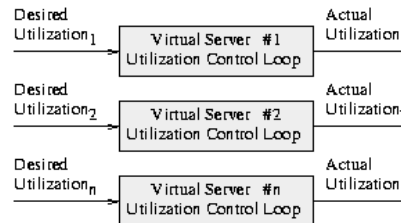
- Virtual site specification:
 - Maximum request rate R and bandwidth W
 - $U = a R + b W$
- Per-site utilization control loops

University of Virginia, Real-time Systems Laboratory



Virtual Server Utilization Control

- Utilization of each virtual server is controlled by its own utilization control loop

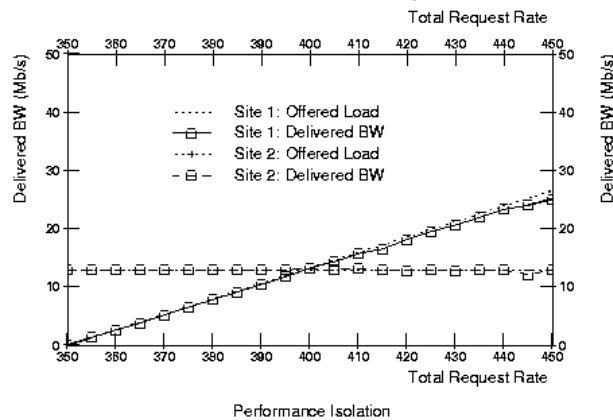


University of Virginia, Real-time Systems Laboratory



Example: Co-hosting Two Sites

- Sites are unaffected by overload

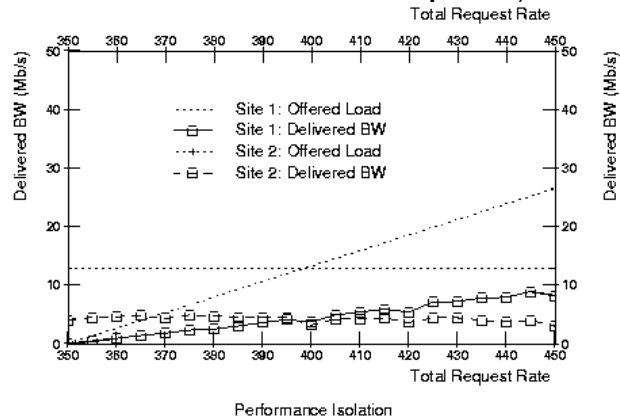


University of Virginia, Real-time Systems Laboratory



... compared to Co-hosting on Apache

- Sites do not achieve capacity



University of Virginia, Real-time Systems Laboratory



Service Prioritization

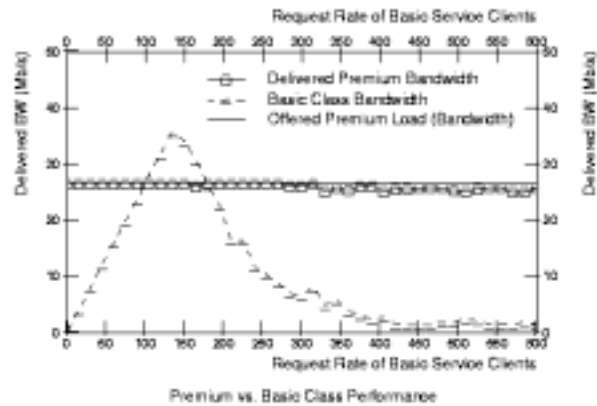
- Requests have priority values depending on their class
- Each class is allocated the capacity not used by higher-priority classes
- A separate utilization control loop is used for each priority class
 - $U_{\text{(desired)}} = 100 - \sum U_{\text{(higher priority)}}$

University of Virginia, Real-time Systems Laboratory



Example: Two Priority Classes

- Basic clients are degraded before premium



University of Virginia, Real-time Systems Laboratory



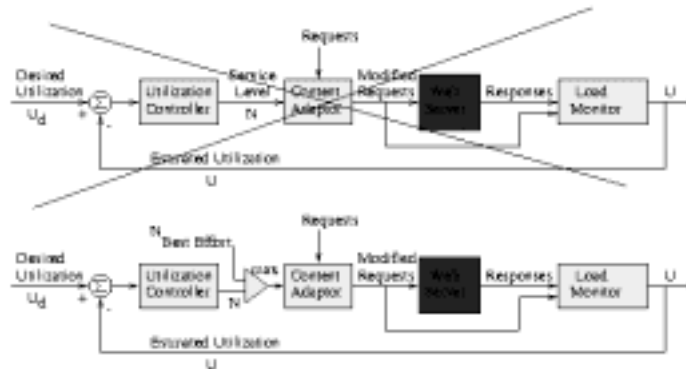
Excess Capacity Sharing versus Policing

- When server resources are partitioned among sites, what if one site exceeds its allocation while others have spare capacity?
 - At low aggregate load, individual virtual servers should be allowed to exceed capacity
 - At high aggregate load, virtual servers should be policed to capacity

University of Virginia, Real-time Systems Laboratory



Realization of Sharing/Policing

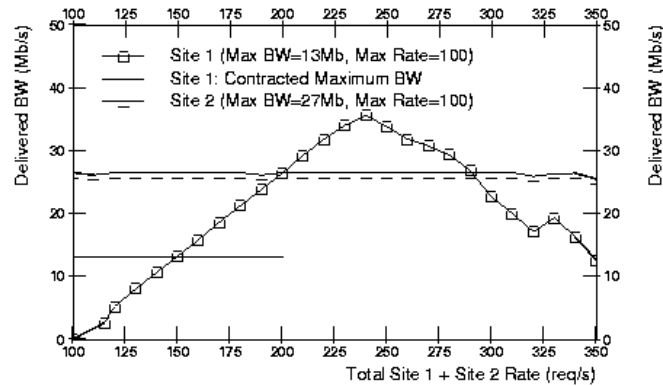


University of Virginia, Real-time Systems Laboratory



Sharing at Low Load

- A site is allowed to use excess capacity



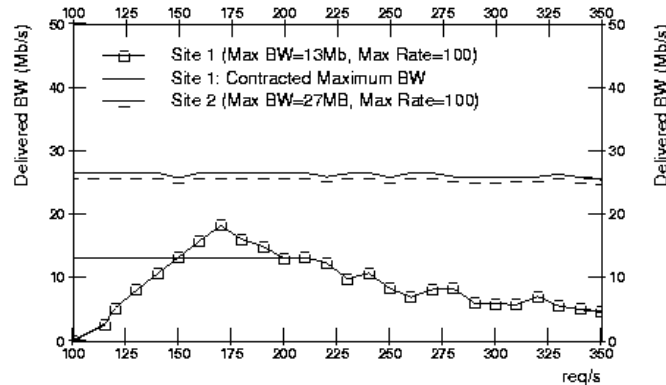
Sharing Excess Capacity (Low Background Load)

University of Virginia, Real-time Systems Laboratory



Policing at High Load

- A site is policed at overload



Sharing Excess Capacity (High Background Load)

University of Virginia, Real-time Systems Laboratory



Summary of Utilization Control

- Utilization control loops are of great use for controlling throughput-like metrics in web servers
- The problem has a control-theoretic formulation
- The feedback control loop is successful in providing desired performance

University of Virginia, Real-time Systems Laboratory

A Case Study on Delay Control: Delay Control in Web Servers*

- Delay is inversely proportional to flow (nonlinear control)
- Consider a class of real-time clients with an adjustable server resource share (i.e., service rate)
- Option 1 (feedback): Choose service rate. Measure difference from desired delay. Adjust service rate, ...
- Option 2 (queuing): Choose service rate to achieve desired delay from equation:
 - $\text{Delay} = \frac{1}{\mu - \lambda}$

*This work is in collaboration with Lui Sha and Xue Liu from UIUC
University of Virginia, Real-time Systems Laboratory

Delay Control

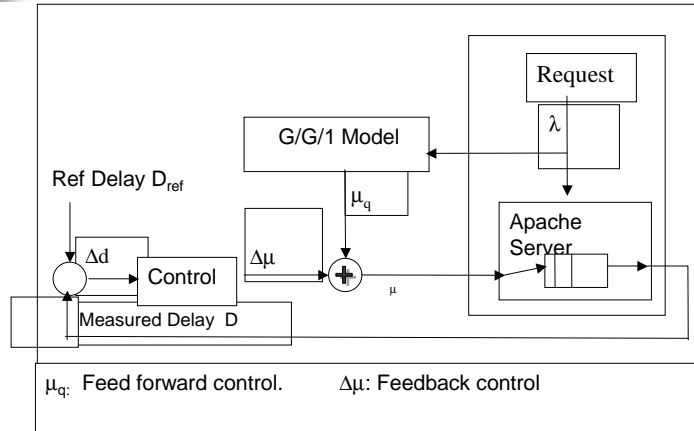
Option 3: A combined approach

- A queuing model predicts delay from flow and service rate measurements
 - $\text{Delay} = \frac{1}{\mu - \lambda}$
- A service rate is chosen to achieve target delay
- Control loop computes a service rate adjustment to reduce residual delay deviation to zero.

University of Virginia, Real-time Systems Laboratory

Lui Sha, Xue Liu

Queueing Model Based Performance Control

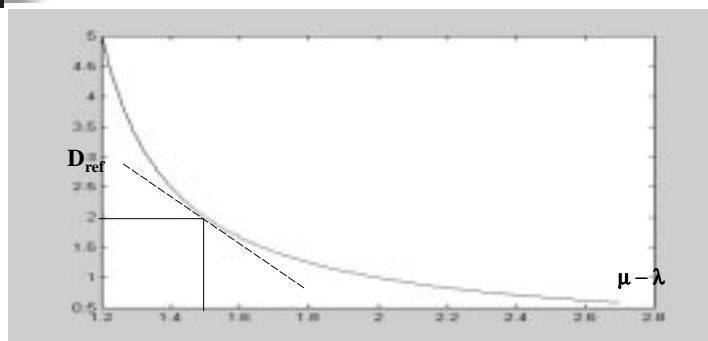


This figure is courtesy of Lui Sha, Xue Liu

University of Virginia, Real-time Systems Laboratory

Lui Sha, Xue Liu

Keeping the System in Neighborhood of linearization



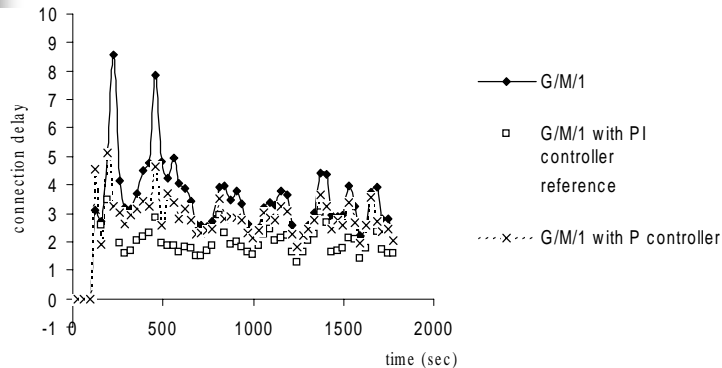
Remark: The queuing predictor linearizes the control loop

This figure is courtesy of Lui Sha, Xue Liu

University of Virginia, Real-time Systems Laboratory

Lui Sha, Xue Liu

An Apache Web Server Experiment



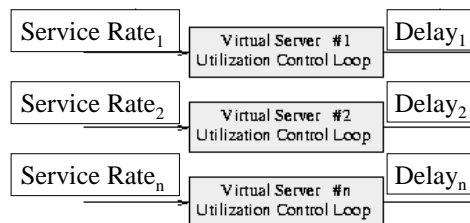
Graphs shown are a subset from Queueing Model Based Network Server Performance Control, by L. Sha, X. Liu (UIUC), Y. Lu, T. Abdelzaher (UVA)

University of Virginia, Real-time Systems Laboratory

Lui Sha, Xue Liu

Delay Control with Multiple Classes?

- Service rate of each virtual server is controlled by its own control loop?

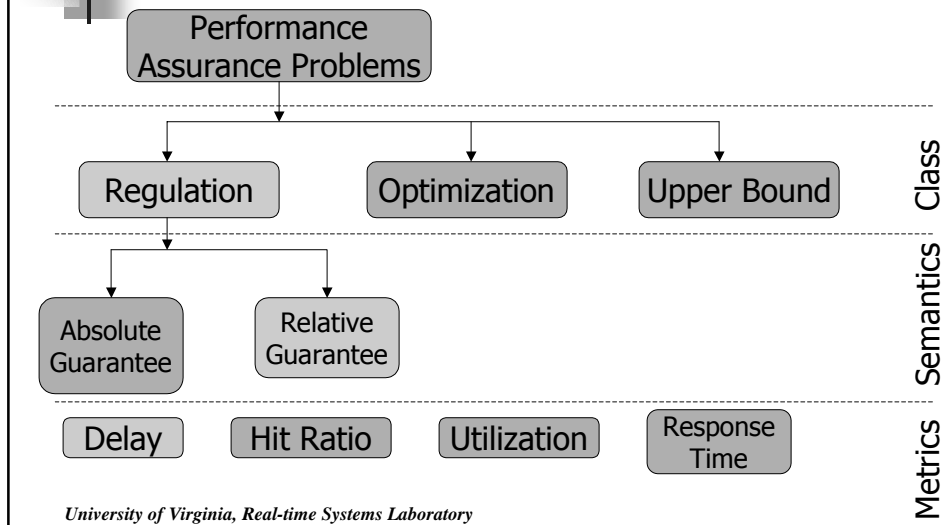


- Unlike throughput, it is much more difficult to achieve performance isolation between delay control loops
 - Throughput is a function of *only the total number* of requests served (not the order of service)
 - Delay is a function of the *order* in which requests are served. Scheduling support is required

University of Virginia, Real-time Systems Laboratory

The rest of Part III has not been presented at ARTES 2002

QoS Guarantees as a Feedback Problem: Relative Guarantees



Relative Guarantees

Relative guarantees:

- Server supports multiple classes of clients
- Different classes have different performance levels, such that a specified ratio between their performance levels is observed
- Example: Class A clients have $1/3$ the delay of Class B clients
 - Ratio $> 1/3$: Class A too slow
 - Ratio $< 1/3$: Class B too slow

University of Virginia, Real-time Systems Laboratory

Case Study: Relative Delay Differentiation in Web Servers

- Problem statement:
 - N classes of users/traffic
 - Average delay of class j is D_j
 - It is required that
$$D_1:D_2:\dots:D_N = K_1:K_2:\dots:K_N$$
 - K_1, K_2, \dots, K_N are specified weights
- Control-theoretical formulation?

University of Virginia, Real-time Systems Laboratory

Control-Theoretical Formulation

- The differentiation objective
$$D_1:D_2:\dots:D_N = K_1:K_2:\dots:K_N$$
- One feedback loop per class
- The feedback control objective

$$\frac{D_i}{D_1 + D_2 + \dots + D_N} = \frac{K_i}{K_1 + K_2 + \dots + K_N}$$

- Error e_i :

$$e_i = \frac{K_i}{K_1 + K_2 + \dots + K_N} - \frac{D_i}{D_1 + D_2 + \dots + D_N}$$

University of Virginia, Real-time Systems Laboratory

Control Loop Output

- Adjust resource allocation of each class j by ΔR_j
- $\Delta R_j = f(e_j)$, where
 - f is linear
 - $f(0) = 0$
- The resource conservation property

$$\sum_j (\Delta R_j) = 0$$

- Proof

$$\sum_j (\Delta R_j) = \sum_j f(e_j) = f(\sum_j e_j) = f(0) = 0$$

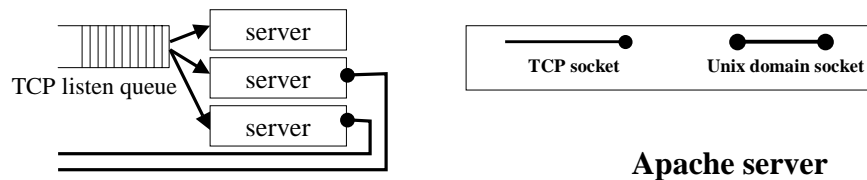
University of Virginia, Real-time Systems Laboratory

Time Invariant Formulation

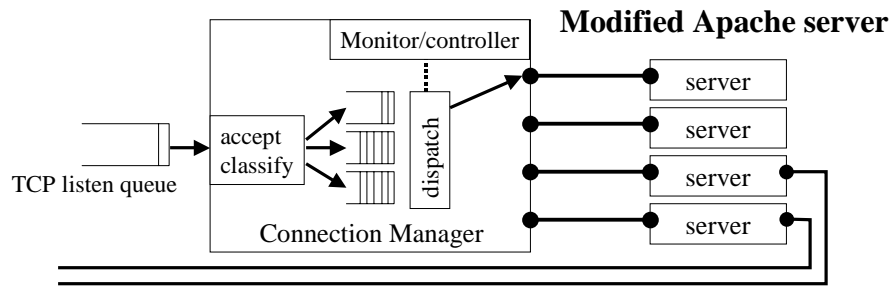
- The differentiation objective
$$D_2 = n_2 D_1, D_3 = n_3 D_1, \dots, D_N = n_N D_1, n_1 = 1$$
- Let $\alpha_j = n_2 n_3 \dots n_N / n_j$
- Let $d_j = \alpha_j D_j, d_{av} = (\sum d_j) / N$
- The feedback control objective
 - $d_j = d_{av}$
- Error $e_i = d_{av} - d_j$

University of Virginia, Real-time Systems Laboratory

Modifying Apache Architecture



Apache server

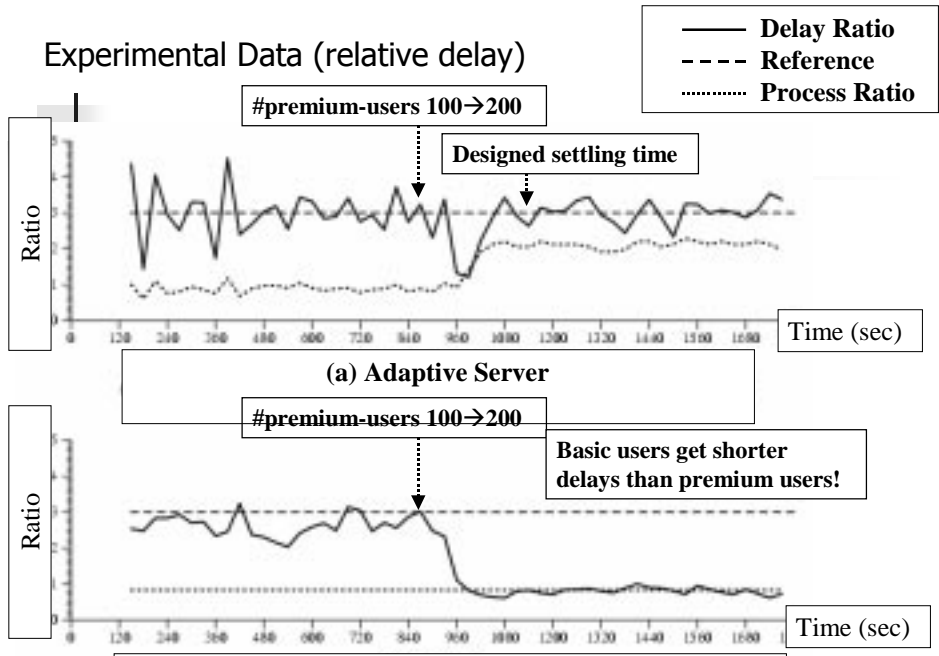


Modified Apache server

University of Virginia, Real-time Systems Laboratory

Courtesy of Chenyang Lu

Experimental Data (relative delay)



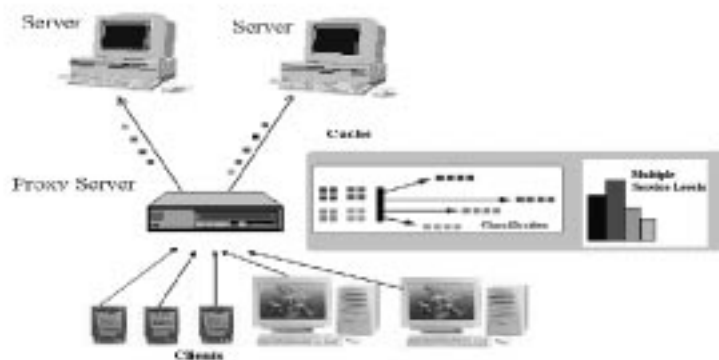
Courtesy of Chenyang Lu

Enhancing Relative Delay Control with Queuing Theory

- As with absolute delay control, relative delay control can be enhanced with queuing theory, by solving for μ_i the set of equations:
 - $\text{Delay}_1/\text{Delay}_2 = (\lambda_2 - \mu_2)/(\lambda_1 - \mu_1)$
 - $\text{Delay}_2/\text{Delay}_3 = (\lambda_3 - \mu_3)/(\lambda_2 - \mu_2)$
 - ...
 - $\mu_1 + \mu_2 + \dots + \mu_n = 1$
- Resources are allocated to each class, i , proportionally to μ_i
- Feedback control is used to adjust resource allocation

University of Virginia, Real-time Systems Laboratory

Case Study II: Differentiated Web Caching



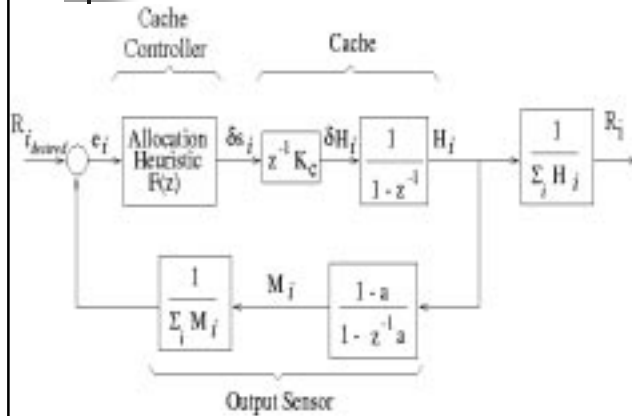
- Classify requests
- Different class has different level of service

University of Virginia, Real-time Systems Laboratory

Courtesy of Ying Lu

Controller Design

Allocation Heuristics Function



How to map error, the difference of set-point and measured value, to disk space change of the class?

To design the allocation heuristics $\Delta S_i(e_i)$, we specify the desired behavior of the closed loop

- R_i follows $R_{i,desired}$ within one sampling time

$$R_i = z^{-1} R_{i,desired}$$

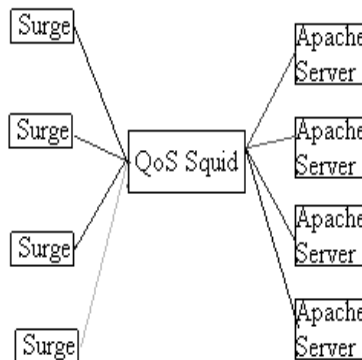
$$F(z) = \frac{\Delta S(z)}{E(z)} = \frac{(1-z^{-1}a) \sum_i H_i}{K_c}$$

$$\Delta S_i(e_i) = \frac{1}{K_c} (e_i[k] - a e_i[k-1])$$

Courtesy of Ying Lu

University of Virginia, Real-time Systems Laboratory

Experimental Setup₁



- Evaluation is conducted on a testbed of nine AMD-based Linux machines interconnected by Ethernet switch (100MHz)
- Clients:
 - Surge (Scalable URL Reference Generator), a tool that generates references matching empirical measurement
- Servers:
 - Apache
- To test the performance of the cache under saturation, we configure the ratio of cache size to population of files requested by one class to be roughly 1 to 10

University of Virginia, Real-time Systems Laboratory

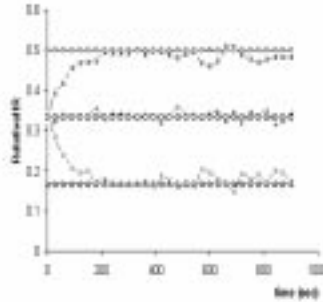
Courtesy of Ying Lu

Performance

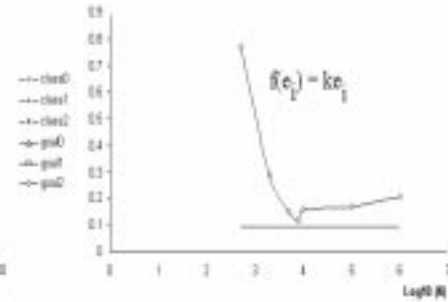
Three classes

Set-point $H_0: H_1: H_2=1:2:3$

Controller: $f(e_i) = \frac{\sum H_i}{K_c} (e_i[k] - ae_i[k-1])$



Relative Hit Rate

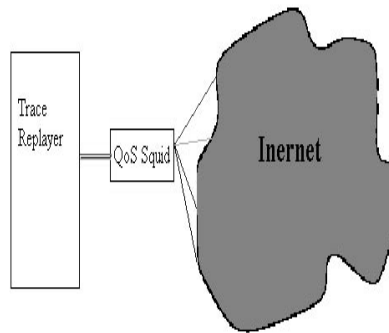


Aggregate Absolute Error

University of Virginia, Real-time Systems Laboratory

Courtesy of Ying Lu

Experimental Setup₂

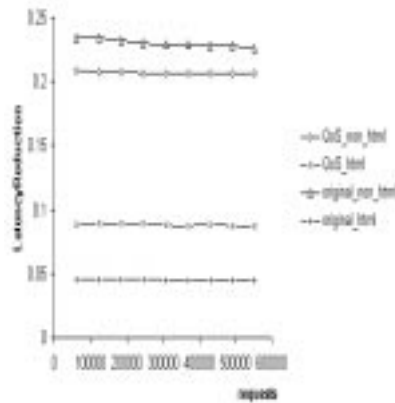


- Clients
 - replay NLNR sanitized access logs
 - class0: html files
 - class1: non-html files
- Servers
 - real servers on the internet

University of Virginia, Real-time Systems Laboratory

Courtesy of Ying Lu

Latency Reduction



Backbone latency reduction

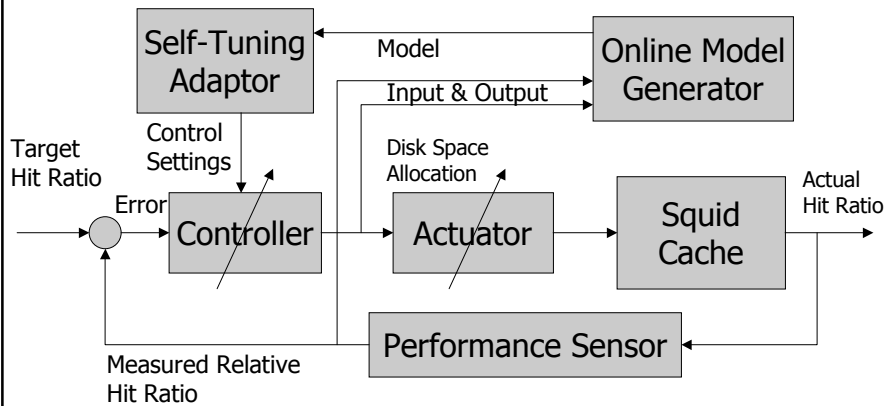
$$\frac{\sum_{j \in \alpha} latency_j}{\sum_{i \in \beta} latency_i}$$

- α includes all the pages that hit in the cache
- β includes all the requested pages

University of Virginia, Real-time Systems Laboratory

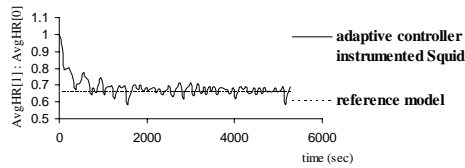
Courtesy of Ying Lu

Adaptive Control

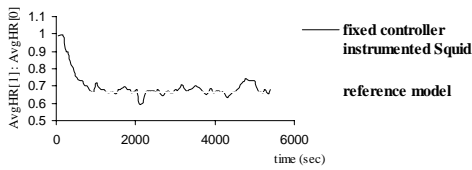


University of Virginia, Real-time Systems Laboratory

Adaptive Controller Performance for Synthetic Log



Experiment with Adaptive Controller

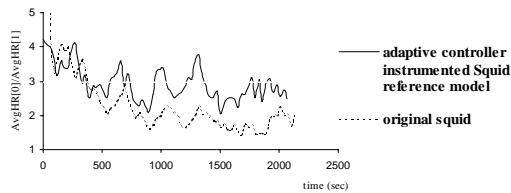


Experiment with Fixed Controller

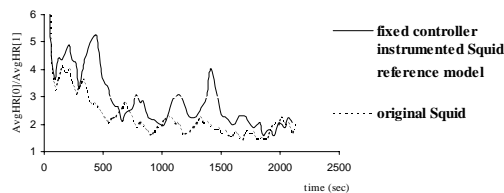
University of Virginia, Real-time Systems Laboratory

Courtesy of Ying Lu

Adaptive Controller Performance for Empirical Log



Experiment with Adaptive Controller

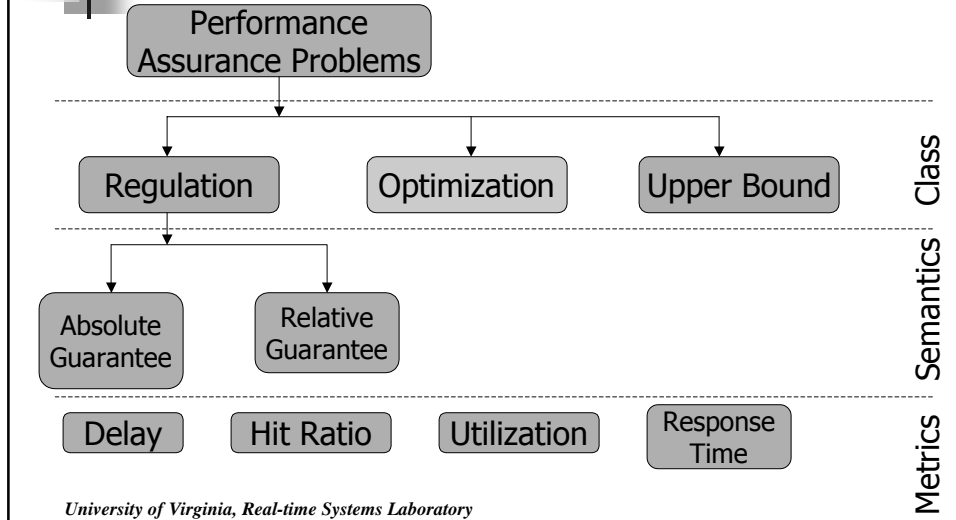


Experiment with Fixed Controller

University of Virginia, Real-time Systems Laboratory

Courtesy of Ying Lu

QoS Guarantees as a Feedback Problem: A Taxonomy

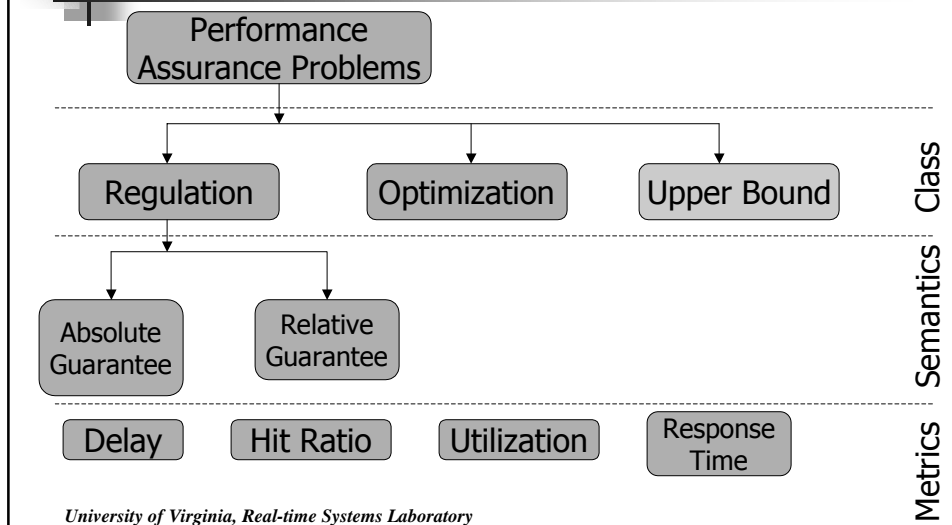


Optimization

- Express the problem as one of achieving an equilibrium where:
 - Marginal cost = Marginal benefit.
- Set point = Marginal cost – Marginal benefit = 0
- Example:
 - Maximize power savings by optimal cache decay in microprocessor architectures

University of Virginia, Real-time Systems Laboratory

QoS Guarantees as a Feedback Problem: A Taxonomy



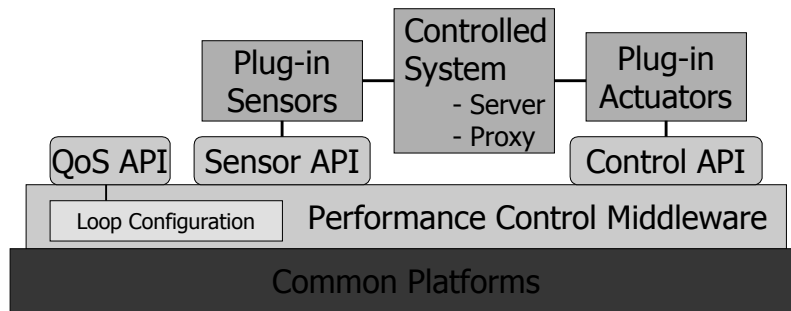
Upper Bound Control

- Remember that delay control in the presence of multiple classes is a difficult problem
- Real-time scheduling theory helps:
 - Translate the problem into one of utilization control using synthetic utilization bounds
 - The control loop guarantees that all deadlines are met as long as the utilization bound is not exceeded
 - The aperiodic utilization bound is used (can be derived for any scheduling policy)

University of Virginia, Real-time Systems Laboratory

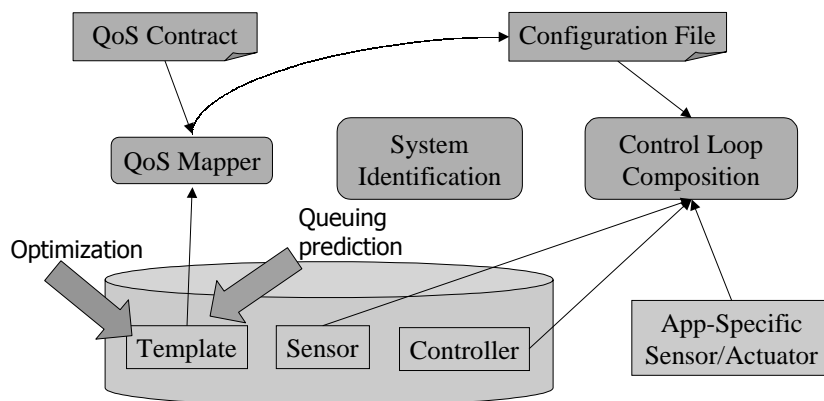
ControlWare Architecture

- API for plug-in performance sensors and actuators
- Common sensor/actuator library
- Engine for mapping QoS specifications to control loops
- Run-time enforcement of feasible region conformance



University of Virginia, Real-time Systems Laboratory

ControlWare-Assisted QoS Provisioning Methodology



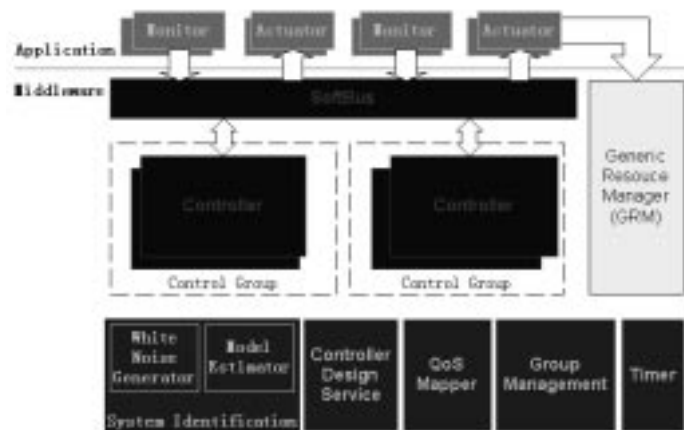
University of Virginia, Real-time Systems Laboratory

QoS Mapping

- User specifies QoS problem
- Problem type: the semantics of performance guarantee (1 template per type)
 - Absolute convergence guarantee
 - Relative guarantee
 - Prioritization guarantee
- Problem instance: the semantics of the controlled variable (1 sensor per instance)
 - Delay
 - Hit ratio
- QoS mapper generates feedback loops

University of Virginia, Real-time Systems Laboratory

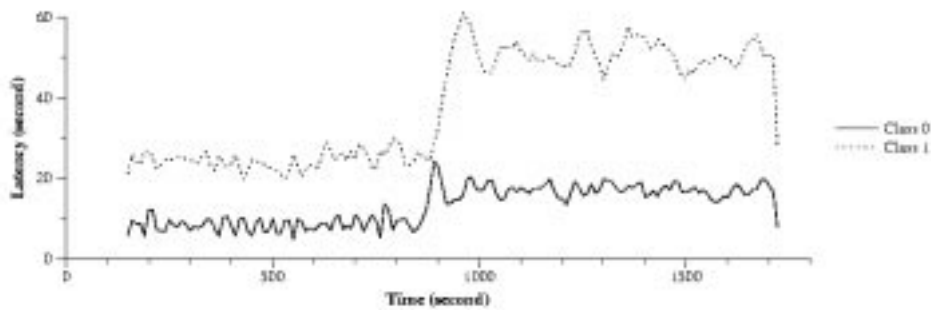
ControlWare Architecture



University of Virginia, Real-time Systems Laboratory

Example 1: Apache Web Server

- Performance



University of Virginia, Real-time Systems Laboratory

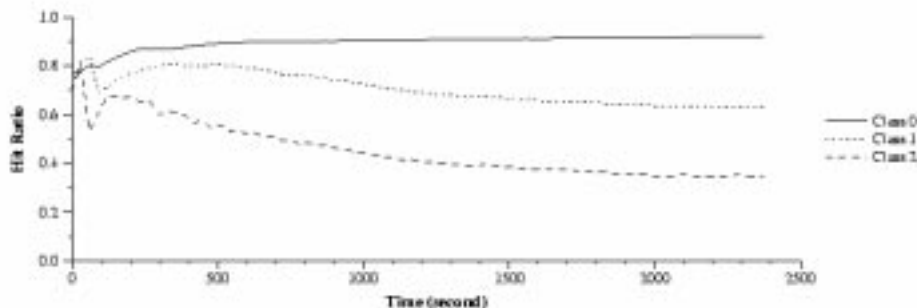
Example 2: Squid Cache Server

- QoS guarantee
 - relative hit ratio guarantee
 - $D_0 : D_1 : D_2 = 3 : 2 : 1$
- Template: relative guarantee template
- Actuator: change cache space allocated to each class

University of Virginia, Real-time Systems Laboratory

Example 2: Squid Cache Server

■ Performance



University of Virginia, Real-time Systems Laboratory

Conclusions

- A first step towards an underlying analytic foundation and design methodology for performance control in open software systems
- A middleware library that embodies the control loop prototypes
- Schedulability theory to relate aggregate state to fine-grained performance guarantees

University of Virginia, Real-time Systems Laboratory

Prior Results

www.cs.virginia.edu/~zaher/publications.html

- Run-time modeling tools:
 - Automated profiling (RTAS '00)
 - QoS Portability (IWQoS '02)
- Capacity planning and resource assignment:
 - Overload/throughput control (CDC '00, IWQoS '99)
 - Performance isolation (IEEE TPDS '01)
- Delay control (RTSS '02)
- Service differentiation tools:
 - Server delay differentiation (RTAS '01),
 - Cache hit ratio differentiation (ICDCS '01)
 - Router delay differentiation (Infocom '02)
 - Prioritization (IWQoS '99)
- ControlWare (ICDCS '02)

University of Virginia, Real-time Systems Laboratory

Future Work

- Study the characteristic features (e.g., non-linearities) of software feedback control systems
- Establish a better understanding of the limitations of control theory in software control
- Integrate control theory with real-time scheduling theory and queuing theory for robust fine-grained guarantees on temporal behavior
- Implement successful performance control mechanisms in the QoS control middleware
- Extend the framework to distributed control

University of Virginia, Real-time Systems Laboratory



Outline of Presentation

- A New Paradigm for Real-Time Computing?
- Advances in Aperiodic Schedulability Bounds
 - A bound for aperiodic tasks
 - Extensions
- Software Performance Control
 - Concepts
 - Case Studies
 - Web Server Control
 - Adaptive Web Cache Control
 - ControlWare
- Future Directions in Real-Time Computing

University of Virginia, Real-time Systems Laboratory



Future Directions in Real-Time Computing

After feedback control?
Real-Time Sensor Networks

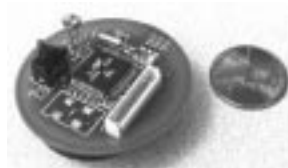
Smart Dust

- What's next?
- A new paradigm for massively distributed computing
 - 100,000s of tiny devices which can be mass-produced at feasible prices
 - Each device has
 - CPU + memory
 - Wireless communication capabilities
 - Sensors/actuators
- Applications: environmental monitoring, surveillance, medical assistance, disaster recovery, ...

University of Virginia, Real-time Systems Laboratory

Current Hardware (Motes)

- Specifications:
 - 8 bit microprocessor
 - 128K ROM
 - 8K RAM
 - 4MHz Clock
- Price (today: \$150)
- In 5 years:
 - Price → below \$1
 - Size → below 1 cubic mm.



University of Virginia, Real-time Systems Laboratory

Applications

- **Smart Paint:**
 - Paint bridges and other large structures with “smart paint”
 - Thousands of in-paint processors will monitor corrosion, fatigue and other bridge data reporting alarms as needed
- **Surveillance:**
 - Thousands of nodes are dropped from the sky on a disaster area
 - Surveillance data is collected, such as location of survivors and possible threats
- **Wild life monitoring:**
 - A sensor network is currently deployed on a island off the cost of Maine, USA, to monitor bird mating season in natural habitat

University of Virginia, Real-time Systems Laboratory

Example: Surveillance with Large Embedded Networks

- Large, dense networks of sensor nodes
 - Radio radius ≈ 30 m \rightarrow surveillance over 9 km²
- 10,000 Devices
- Sensors: motion, magnetic, temperature, light, sound, vibration, infrared, ...

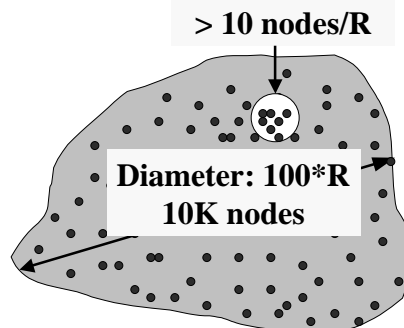


Figure is courtesy of Chenyang Lu

University of Virginia, Real-time Systems Laboratory

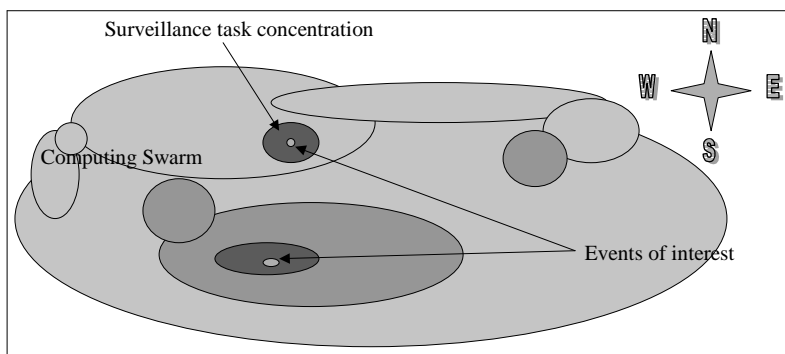
Challenges

- New resource constraints
 - Power
 - Wireless bandwidth
- New task/computation models
 - Distributed computational clouds
 - Density-based task and resource models
 - Data-centric (liquid) load flow
- New scheduling algorithms
 - Communication-centric
 - Distributed
 - Probabilistic

University of Virginia, Real-time Systems Laboratory

Operating System Challenge: Spatially Distributed Tasks

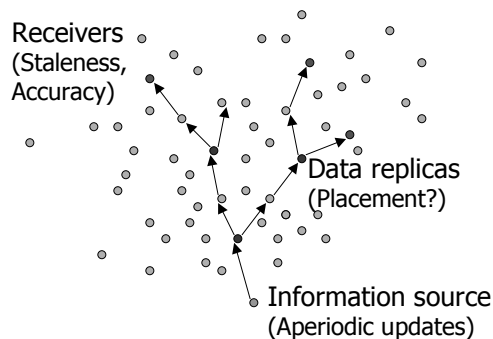
- Spatial task model and resource allocation
 - Environment-driven task population control
 - Real-time task density allocation algorithms



University of Virginia, Real-time Systems Laboratory

Information Services Challenge: Power/QoS trade-offs

- Middleware to control power/information-quality trade-offs in sensor networks.



Goal: Find the optimal replica placement and update propagation policy to minimize power usage subject to per-receiver data QoS constraints (e.g., staleness and accuracy).

University of Virginia, Real-time Systems Laboratory

Networking Challenges: A New Protocol Stack

- Network layer
 - Attribute-based addressing
 - ID-less QoS-sensitive routing
- Transport layer
 - Mobile endpoints
 - Group-to-group real-time communication
- Multicast/intra-group communication
 - Relaxed semantics suitable for real-time applications
- New network programming paradigms
- Distributed testing??

University of Virginia, Real-time Systems Laboratory



Conclusion

- Sensor networks are massively distributed embedded sensing and control systems which interact in real-time with a physical environment
- We have very little towards analytic tools for predicting and controlling the temporal behavior of such systems
- Fundamentally new system models are needed
- Fundamentally new approaches are needed for real-time scheduling and schedulability analysis

University of Virginia, Real-time Systems Laboratory



Final Word

- Software performance guarantees will become a part of vendor's contractual obligations towards consumer
- The search for analytic foundations to provide such guarantees is gaining popularity
- New models are emerging for computation
- New paradigms are needed for achieving predictable behavior in large-scale software systems (e.g., large servers or large distributed systems)
- Initial steps have been made to merge real-time scheduling theory, queuing theory and feedback control theory to produce a framework for robust QoS

University of Virginia, Real-time Systems Laboratory



Acknowledgements

- I would like to acknowledge:
 - Chenyang Lu, for implementing QoS-aware web servers
 - Ying Lu and Avneesh Saxena for their work on differentiated caching services
 - Ronghua Zhang for implementing ControlWare
 - Vivek Sharma, Bjorn Andersson, and Jan Jonsson for their help with the multiprocessor aperiodic utilization bound
 - Shelby Funk, Sanjoy Baruah, Deji Chen and Al Mok for finding flaws in an earlier derivation of the bound
 - Jack Stankovic, Lui Sha, Sang Son, Gang Tao, Nina Bhatti, Kang Shin, Kevin Skadron, Jorg Liebeherr, and Xue Liu for their collaboration and help

University of Virginia, Real-time Systems Laboratory