On the Use of Online Performance Models in Autonomic Computing

Prof. Daniel Menascé
Department of Computer Science
E-Center for E-Business
George Mason University
Fairfax, VA, USA
Menasce@cs.gmu.edu
www.cs.gmu.edu/faculty/menasce.html
Defining Features of Autonomic Computing Systems
(www.research.ibm.com/autonomic/overview/elements.html)

1. Self-aware
2. Self-configurable and self-reconfigurable
3. Self-optimizing
4. Self-healing
5. Self-protecting
6. Adaptable to the environment
7. Use open standards
8. Transparent use of IT resources to achieve business/personal user goals.
Defining Features of Autonomic Computing Systems
(www.research.ibm.com/autonomic/overview/elements.html)

1. Self-aware
2. Self-configurable and self-reconfigurable
3. Self-optimizing
4. Self-healing
5. Self-protecting
6. Adaptable to the environment
7. Use open standards
8. Transparent use of IT resources to achieve business/personal user goals.
Rest of this talk …

- Novel uses for performance models
- Two examples of self-regulating systems:
  - A three-tiered e-commerce system
  - QoS-aware software components
- Concluding Remarks
Rest of this talk …

- Novel uses for performance models
- Two examples of self-regulating systems:
  - A three-tiered e-commerce system
  - QoS-aware software components
- Concluding Remarks
What are performance models good for?

- At the design stage:
  - Compare competing design alternatives.
    - A large number of low capacity servers vs. a small number of large capacity servers?
What are performance models good for?

- At the design stage:
  - Compare competing design alternatives.
    - A large number of low capacity servers vs. a small number of large capacity servers?

- During production:
  - Medium and long-term (weeks and months):
    - Capacity planning.
Capacity Planning

- Response Time (sec)
- Load (sessions/sec)
- QoS Goal

3 mo. 6 mo. 9 mo.

© 2004 D. A. Menascé. All Rights Reserved.
What are performance models good for?

- At the design stage:
  - Compare competing design alternatives.
    - A large number of low capacity servers vs. a small number of large capacity servers?

- During production:
  - Medium and long-term (weeks and months):
    - Capacity planning.
  - Short-term (minutes):
    - Dynamic reconfiguration.
Rest of this talk …

- Novel uses for performance models
- Two examples of self-regulating systems:
  - A three-tiered e-commerce system
  - QoS-aware software components
Automatic QoS Control: Motivation

- Modern computer systems are complex and composed of multiple tiers.
Multi-tier Architecture

- Internet
- Firewall (FW)
- Load Balancer
- LAN 1
- LAN 2
- LAN 3
- LAN 4
- Web Servers
- Application Servers
- Database Server (e.g., mainframe)

© 2004 D. A. Menascé. All Rights Reserved.
Automatic QoS Control: Motivation

- Modern computer systems are complex and composed of multiple tiers.
- The workload presents short-term variations with high peak-to-average ratios.
Multi-scale time workload variation

3600 sec

60 sec

1 sec

© 2004 D. A. Menascé. All Rights Reserved.
Automatic QoS Control: Motivation

- Modern computer systems are complex and composed of multiple tiers.
- The workload presents short-term variations with high peak-to-average ratios.
- Many software and hardware parameters influence the performance of e-commerce sites.

Manual reconfiguration is not an option!
Need self-managing systems.
Computer System

Service Demand Computation

QoS Controller Algorithm

Workload Analyzer

Performance Model Solver

arriving requests

completing requests

$X_0$

$U_i$

QoS goals

© 2004 D. A. Menascé. All Rights Reserved.
\[ D_i = \frac{U_i}{X_0} \]

Equation: \( D_i = \frac{U_i}{X_0} \)

Diagram Description:
- **Computer System**
  - **Arriving Requests**
  - **Completing Requests**

**QoS Controller**
- **QoS Controller Algorithm**
- **Performance Model Solver**
- **Workload Analyzer**
- **Service Demand Computation**

Equation in the diagram:
\[ X_0 \]

Equation: \( X_0 \)
Computer System

Service Demand Computation (2)

Workload Analyzer (3)

QoS Controller Algorithm (5)

Performance Model Solver (4)

QoS Controller

arriving requests

(1) (2) (3) (4) (5) (6) (7) (8) (9) (10) (11) (12)

QoS goals

© 2004 D. A. Menascé. All Rights Reserved.
Controller Interval

- reconfiguration commands
- i-th interval
- measurements from servers in e-commerce site
- controller algorithm
- (i+1)-th interval
- measurements from servers in e-commerce site
- reconfiguration commands

© 2004 D. A. Menascé. All Rights Reserved.
Combined QoS Metric

\[ QoS = w_R \times \Delta QoS_R + w_P \times \Delta QoS_P + w_X \times \Delta QoS_X \]

\( w_R \), \( w_P \), and \( w_X \) are relative weights that indicate the relative importance of response time, throughput, and probability of rejection.
QoS Metric

\[ QoS = w_R \times \Delta QoS_R + w_P \times \Delta QoS_P + w_X \times \Delta QoS_X \]

\( w_R \), \( w_P \), and \( w_X \) are relative weights that indicate the relative importance of response time, throughput, and probability of rejection.

\( \Delta QoS_R \), \( \Delta QoS_P \), and \( \Delta QoS_X \) are relative deviations of the response time, throughput, and probability of rejection metrics with respect to their desired levels.
QoS Metric

\[ QoS = w_R \times \Delta QoS_R + w_P \times \Delta QoS_P + w_X \times \Delta QoS_X \]

\( w_R \), \( w_P \), and \( w_X \) are relative weights that indicate the relative importance of response time, throughput, and probability of rejection.

\( \Delta QoS_R \), \( \Delta QoS_P \), and \( \Delta QoS_X \) are relative deviations of the response time, throughput, and probability of rejection metrics with respect to their desired levels.

\[ \Delta QoS_R, \Delta QoS_P, \text{ and } \Delta QoS_X \]

The QoS metric is a dimensionless number in the interval \([-1, 1]\).
QoS Metric

\[ QoS = w_R \times \Delta QoS_R + w_P \times \Delta QoS_P + w_X \times \Delta QoS_X \]

\( w_R, w_P, \) and \( w_X \) are relative weights that indicate the relative importance of response time, throughput, and probability of rejection.

\( \Delta QoS_R, \Delta QoS_P, \) and \( \Delta QoS_X \) are relative deviations of the response time, throughput, and probability of rejection metrics with respect to their desired levels.

The QoS metric is a dimensionless number in the interval \([-1, 1]\).

If all metrics meet or exceed their QoS targets, \( QoS = 0 \).
Response Time Deviation

$$\Delta QoS_R = \frac{R_{\text{max}} - R_{\text{measured}}}{\max(R_{\text{max}}, R_{\text{measured}})}$$

- $= 0$ if the response time meets its target.
- $> 0$ if the response time exceeds its target.
- $< 0$ if the response time does not meet its target.

- $\Delta QoS_R \leq 1 - \left( \sum_{i=1}^{K} D_i \right) / R_{\text{max}} < 1$
- $-1 < -(1 - R_{\text{max}} / R_{\text{measured}}) \leq \Delta QoS_R$

© 2004 D. A. Menascé. All Rights Reserved.
Probability of Rejection Deviation

\[ \Delta QoS_p = \frac{P_{\text{max}} - P_{\text{measured}}}{\max(P_{\text{max}}, P_{\text{measured}})} \]

- \( = 0 \) if the probability of rejection meets its target.
- \( > 0 \) and \( = 1 \) if the probability of rejection exceeds its target.
- \( < 0 \) and \( = -1 \) if the probability of rejection does not meet its target.
Throughput Deviation

\[ \Delta QoS_X = \frac{X_{measured} - X^*_\text{min}}{\max(X_{measured}, X^*_\text{min})} \]

- \( X^*_\text{min} = \min(\lambda, X_{\text{min}}) \)
- = 0 if the throughput meets its target.
- > 0 and < 1 if the throughput exceeds its target.
- < 0 and > -1 if the throughput does not meet its target.
Heuristic Optimization Approach

- The space of configuration points is searched using a combinatorial search technique.
- Each point has a QoS value computed through an analytic performance model.

\[
QoS = f(\vec{W}, c_1, c_2, \ldots, c_m)
\]
Heuristic Optimization Approach

- The space of configuration points is searched using combinatorial search techniques.
- Each point has a QoS value computed through an analytic performance model.
Heuristic Optimization Approach

- The space of configuration points is searched using combinatorial search techniques.
- Each point has a QoS value computed through an analytic performance model.
A Queuing Model is Used to Compute QoS Values
Prototype Configuration

QoS Controller

Database Server

Web Server

Application Server

100 Mbps Hub

Workload Generator

Workstation

TPC-W site

© 2004 D. A. Menascé. All Rights Reserved.
Experiment Results

Arrival rate

© 2004 D. A. Menascé. All Rights Reserved.
Results of QoS Controller
Experiment Results

Arrival rate

QoS is not met!
Variable interarrival and service times of requests

- Real workloads exhibit high variability in:
  - Traffic intensity
  - Service demands at various system resources

- Need to investigate the efficiency of the proposed self-managing technique under these conditions

- Consider variability in requests inter-arrival time and requests service times at physical resources (e.g., CPU, disk)
Effect of Varying the COV of the Service Time (Ca = 1)

Cs = 1

Cs = 2

Cs = 4
Effect of Varying the COV of the Interarrival Time (Cs = 1)

- **Ca = 1, Cs = 1**
  - Controller Interval vs. Average QoS
  - Graphs show the performance of different algorithms:
    - No Controller
    - Beam Search
    - Hill Climbing

- **Ca = 2, Cs = 1**
  - Similar to Ca = 1, Cs = 1

- **Ca = 4, Cs = 1**
  - Similar to Ca = 1, Cs = 1

**Ca = 1**

**Ca = 2**

**Ca = 4**

© 2004 D. A. Menascé. All Rights Reserved.
Extreme values for Ca and Cs

![Graph showing Ca = 4, Cs = 4](image)

- Controller Interval
- Average QoS
- Lines represent No Controller, Beam Search, and Hill Climbing
Sensitivity of Controller to SLAs

- Need to investigate the controller behavior in the case of a variation in the SLAs
- We ran experiments for stricter and more relaxed SLAs
  - Base: $R_{\text{max}} = 1.2 \text{ sec}, X_{\text{min}} = 5 \text{ req/sec}, P_{\text{max}} = 0.05$
  - Strict: $R_{\text{max}} = 1.0 \text{ sec}, X_{\text{min}} = 7 \text{ req/sec}, P_{\text{max}} = 0.03$
  - Relaxed: $R_{\text{max}} = 1.5 \text{ sec}, X_{\text{min}} = 4 \text{ req/sec}, P_{\text{max}} = 0.10$
- Used $Ca = Cs = 2$
Sensitivity of Controller to SLAs

Controller Sensitivity

Average QoS Relative Variation

Controller Interval

- ▲ No Controller Relaxed SLA
- ■ No Controller Strict SLA
- ○ Beam Search Relaxed SLA
- □ Beam Search Strict SLA
Rest of this talk …

- Novel uses for performance models
- Two examples of self-regulating systems:
  - A three-tiered e-commerce system
  - QoS-aware software components
- Concluding Remarks
Characteristics of the new generation of distributed software systems

- Highly distributed
- Component-based (for reusability)
- Service-oriented architectures (SOA)
- Unattended operation
- Hostile environments
- Composed of a large number of “replaceable” components discovered at run-time
- Run on a multitude of (unknown and heterogeneous) hardware and network platforms
Requirements of Next Generation of Large Distributed Systems

- Adaptable and self-configurable to changes in workload intensity:
  - QoS requirements at the application and component level must be met.

- Adaptable and self-configurable to withstand attacks and failures:
  - Availability and security requirements must be met.

self-configurable, self-optimizing, self-healing, and self-protecting
Q-Applications and Q-components

Q-application

Q-component

Q-component

Q-component

Q-component

Service directory

registration

© 2004 D. A. Menascé. All Rights Reserved.
Q-Applications and Q-components

Q-application

Service directory

Q-component

Q-component

Q-component

Q-component

Q-component

Q-component

discovery

© 2004 D. A. Menascé. All Rights Reserved.
Q-Applications and Q-components

Q-application

Q-component

Q-component

Q-component

Q-component

QoS Negotiation

© 2004 D. A. Menascé. All Rights Reserved.
Q-Applications and Q-components

Q-application

Q-component

Q-component

Q-component

Q-component

Service Access

© 2004 D. A. Menascé. All Rights Reserved.
QoS-Aware Software Components: Q-Components

- Engage in QoS Negotiations (accept, reject, counter-offer)
- Provide QoS guarantees for multiple concurrent services
- Maintain a table of QoS commitments
- Service dispatching based on accepted QoS commitments
- Q-components are the building blocks of QoS-aware applications
Architecture of a typical software component

Service Registration

Service Dispatcher

Service 1

...
Architecture of a Q-component (QoS Negotiation)

- QoS Request Handler
- Service Registration
- Service Dispatcher
  - Service 1
  - \ldots
  - Service k
Architecture of a Q-component (QoS Negotiation)

- QoS Request Handler
- QoS Negotiator
- QoS Evaluator
- Service Registration
- Service Dispatcher
- Service 1
- Service k

Table of QoS commitments
Architecture of a Q-component (QoS Negotiation)

- QoS Request Handler
- QoS Negotiator
- QoS Evaluator
- Performance Model Solver
- Service Registration
- Table of QoS commitments
- Service Dispatcher
- Service 1
- ... Service k
Architecture of a Q-component – Service Requests

Service Registration

Service Dispatcher

Service 1

Service k

Table of QoS commitments

Performance Monitor

© 2004 D. A. Menascé. All Rights Reserved.
Architecture of a Q-component

- QoS Request Handler
- QoS Negotiator
- QoS Evaluator
- Performance Model Solver
- Service Registration
- Performance Monitor
- Table of QoS commitments
- Service Dispatcher
  - Service 1
  - ...
### Successful QoS Negotiation

<table>
<thead>
<tr>
<th>Client</th>
<th>Q-component</th>
</tr>
</thead>
<tbody>
<tr>
<td>QoSRequest(rid,Sid,N,Rmax,Xmin)</td>
<td>Request in ToC</td>
</tr>
<tr>
<td>Accept(rid,token)</td>
<td></td>
</tr>
<tr>
<td>ServiceReq (... ,token)</td>
<td></td>
</tr>
<tr>
<td>ReplyReq (...)</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>ServiceReq (... ,token)</td>
<td></td>
</tr>
<tr>
<td>ReplyReq (...)</td>
<td></td>
</tr>
<tr>
<td>EndSession (token)</td>
<td>Request removed from ToC</td>
</tr>
</tbody>
</table>

© 2004 D. A. Menascé. All Rights Reserved.
On-time Accepted Counteroffer

<table>
<thead>
<tr>
<th>Client</th>
<th>Q-component</th>
</tr>
</thead>
<tbody>
<tr>
<td>QoSRequest(rid,Sid,N,Rmax,Xmin)</td>
<td>Request in ToC</td>
</tr>
<tr>
<td>CounterOffer (rid,N’,token)</td>
<td>timeout</td>
</tr>
<tr>
<td>AcceptCounterOffer (token)</td>
<td></td>
</tr>
</tbody>
</table>

© 2004 D. A. Menascé. All Rights Reserved.
## Expired Accepted Counteroffer

<table>
<thead>
<tr>
<th>Client</th>
<th>Q-component</th>
</tr>
</thead>
<tbody>
<tr>
<td>QoSRequest(rid, Sid, N, Rmax, Xmin)</td>
<td>Request in ToC</td>
</tr>
<tr>
<td>CounterOffer (rid, N’, token)</td>
<td>timeout</td>
</tr>
<tr>
<td>AcceptCounterOffer (token)</td>
<td>Request removed from ToC</td>
</tr>
<tr>
<td>ExpiredCounterOffer (rid)</td>
<td></td>
</tr>
</tbody>
</table>
# Rejected Counteroffer

<table>
<thead>
<tr>
<th><strong>Client</strong></th>
<th><strong>Q-component</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>QoSRequest(rid, Sid, N, Rmax, Xmin)</td>
<td>Request in ToC</td>
</tr>
<tr>
<td>CounterOffer (rid, N’, token)</td>
<td>timeout</td>
</tr>
</tbody>
</table>
| RejectCounterOffer (token)          | Request removed from ToC                                           

© 2004 D. A. Menascé. All Rights Reserved.
## Rejected QoS Negotiation

<table>
<thead>
<tr>
<th><strong>Client</strong></th>
<th><strong>Q-component</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>QoSRequest(rid,Sid,N,Rmax,Xmin)</td>
<td>RejectQoSRequest (rid)</td>
</tr>
</tbody>
</table>
# Decision Table for QoS Negotiation

<table>
<thead>
<tr>
<th>Reason</th>
<th>Remedy</th>
<th>Current</th>
<th>Others</th>
<th>Decision</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>1. Current and other requests are satisfied</strong></td>
<td>Accept</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>2. Only Current is Violated</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Only MAXR is violated</td>
<td>Decrease N</td>
<td>OK</td>
<td>OK</td>
<td>Counter Offer</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Not OK: MINX is violated or N=0</td>
<td>OK</td>
<td>Reject</td>
</tr>
<tr>
<td>Only MINX is violated</td>
<td>Increase N</td>
<td>OK</td>
<td>OK</td>
<td>Counter Offer</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Not OK &amp; MAXR is violated</td>
<td>OK</td>
<td>Reject</td>
</tr>
<tr>
<td>MINX &amp; MAXR are violated</td>
<td>Decreasing N reduces X and increasing N increases R. So, there is no solution.</td>
<td>Reject</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>3. Current Request and Others are Violated</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Only MAXR is violated</td>
<td>Decrease N</td>
<td>OK</td>
<td>OK</td>
<td>Counter Offer</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Not OK: MINX is violated or N=0</td>
<td>OK or not OK</td>
<td>Reject</td>
</tr>
<tr>
<td>Only MINX is violated</td>
<td></td>
<td>X could be increased by increasing N. But this would further violate the QoS of other classes.</td>
<td>Reject</td>
<td></td>
</tr>
<tr>
<td>MINX &amp; MAXR are violated</td>
<td>Decreasing N reduces X and increasing N increases R. So, there is no solution.</td>
<td>Reject</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>4. Only Other Requests are Violated</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Any</td>
<td>Decrease N</td>
<td>OK</td>
<td>Not OK: N=1 but others still violated</td>
<td>Counter Offer</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Not OK: MINX violated or N=0</td>
<td>OK or not OK</td>
<td>Reject</td>
</tr>
</tbody>
</table>

© 2004 D. A. Menascé. All Rights Reserved.
# Building a Performance Model

**New Request:** \( \text{Sid} = 3, N = 12 \)

### Base Matrix of Service Demands (in msec):

<table>
<thead>
<tr>
<th>Service</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>25</td>
<td>34</td>
<td>20</td>
</tr>
<tr>
<td>Disk 1</td>
<td>30</td>
<td>50</td>
<td>24</td>
</tr>
<tr>
<td>Disk 2</td>
<td>28</td>
<td>42</td>
<td>31</td>
</tr>
</tbody>
</table>

### Table of Commitments (ToC):

<table>
<thead>
<tr>
<th>Commitment ID</th>
<th>Service ID</th>
<th>N</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>13</td>
<td></td>
</tr>
</tbody>
</table>

### Matrix of Service Demands (in msec)

<table>
<thead>
<tr>
<th>Class</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>34</td>
<td>20</td>
<td>25</td>
<td>25</td>
<td>34</td>
<td>20</td>
</tr>
<tr>
<td>Disk 1</td>
<td>50</td>
<td>24</td>
<td>30</td>
<td>30</td>
<td>50</td>
<td>24</td>
</tr>
<tr>
<td>Disk 2</td>
<td>42</td>
<td>31</td>
<td>28</td>
<td>28</td>
<td>42</td>
<td>31</td>
</tr>
</tbody>
</table>

**Vector N:**

| 10 | 15 | 8  | 20 | 13 | 12 |
Building a Performance Model

New Request:  Sid = 3, N = 12

Base Matrix of Service Demands (in msec):

<table>
<thead>
<tr>
<th>Service</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>25</td>
<td>34</td>
<td>20</td>
</tr>
<tr>
<td>Disk 1</td>
<td>30</td>
<td>50</td>
<td>24</td>
</tr>
<tr>
<td>Disk 2</td>
<td>28</td>
<td>42</td>
<td>31</td>
</tr>
</tbody>
</table>

Table of Commitments (ToC):

<table>
<thead>
<tr>
<th>Commitment ID</th>
<th>Service ID</th>
<th>N</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>13</td>
<td></td>
</tr>
</tbody>
</table>

Matrix of Service Demands (in msec)

<table>
<thead>
<tr>
<th>Class</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>34</td>
<td>20</td>
<td>25</td>
<td>25</td>
<td>34</td>
<td>20</td>
</tr>
<tr>
<td>Disk 1</td>
<td>50</td>
<td>24</td>
<td>30</td>
<td>30</td>
<td>50</td>
<td>24</td>
</tr>
<tr>
<td>Disk 2</td>
<td>42</td>
<td>31</td>
<td>28</td>
<td>28</td>
<td>42</td>
<td>31</td>
</tr>
</tbody>
</table>

Vector N:

| 10 | 15 | 8  | 20 | 13 | 12 |
Experimental Evaluation

- Built Q-components in Java
- Generated random workload and captured the workload for replay.
- Measured the QoS of the random workload without the Q-component.
- Replayed the random workload against the Q-component and selected a stricter QoS level than obtained without the Q-component
- Ran several experiments and computed average and confidence intervals.
Service 0
Results:
Service 1

Results:
Service 2

Results:
<table>
<thead>
<tr>
<th>Svc No.</th>
<th>No. Dropped Sessions</th>
<th>No. of Sessions</th>
<th>% Drop</th>
<th>% Resp. Time Reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>24</td>
<td>440</td>
<td>5</td>
<td>11</td>
</tr>
<tr>
<td>1</td>
<td>19</td>
<td>470</td>
<td>4</td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>59</td>
<td>590</td>
<td>10</td>
<td>7</td>
</tr>
<tr>
<td>Total</td>
<td>102</td>
<td>1500</td>
<td>7</td>
<td>9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Svc No.</th>
<th>No. Dropped Sessions</th>
<th>No. of Sessions</th>
<th>% Drop</th>
<th>% Resp. Time Reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>52</td>
<td>440</td>
<td>12</td>
<td>21</td>
</tr>
<tr>
<td>1</td>
<td>66</td>
<td>470</td>
<td>14</td>
<td>16</td>
</tr>
<tr>
<td>2</td>
<td>148</td>
<td>590</td>
<td>25</td>
<td>12</td>
</tr>
<tr>
<td>Total</td>
<td>266</td>
<td>1500</td>
<td>18</td>
<td>16</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Svc No.</th>
<th>No. Dropped Sessions</th>
<th>No. of Sessions</th>
<th>% Drop</th>
<th>% Resp. Time Reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>92</td>
<td>440</td>
<td>21</td>
<td>28</td>
</tr>
<tr>
<td>1</td>
<td>140</td>
<td>470</td>
<td>30</td>
<td>26</td>
</tr>
<tr>
<td>2</td>
<td>263</td>
<td>590</td>
<td>45</td>
<td>20</td>
</tr>
<tr>
<td>Total</td>
<td>102</td>
<td>1500</td>
<td>33</td>
<td>24</td>
</tr>
</tbody>
</table>
Stock Quote Service

Delayed Service

Response Time (sec)

sessionID

Non QoS-Aware

Q-Component
### Summary of Results for Stock Quouotes Application

<table>
<thead>
<tr>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requested Response Time Reduction</td>
<td>20.0%</td>
</tr>
<tr>
<td>Obtained Response Time Reduction</td>
<td>27.0%</td>
</tr>
<tr>
<td>Percent dropped sessions</td>
<td>33.5%</td>
</tr>
<tr>
<td>Percent counter offers</td>
<td>35.0%</td>
</tr>
<tr>
<td>Percent accepted requests</td>
<td>31.5%</td>
</tr>
</tbody>
</table>
Concluding Remarks

- Performance models can be used to build QoS controllers for complex multi-tiered systems:
  - Controlled system exhibits better QoS values even in case of high variability in request’s inter-arrival and service times
  - Short term workload forecasting improves the QoS, especially when the workload intensity gets close to system saturation level
  - Dynamic adjustment of the controller interval length improves the QoS further
  - Even when basic model assumptions are violated, the models are robust enough to track the evolution of the performance metrics as the workload and configuration parameters change.
Concluding Remarks (Cont’d)

- Performance models can be used by software components to make admission control decisions.
  - QoS components should be able to negotiate QoS requests and perform admission control
  - QoS negotiation overhead is small (it did not exceed 10% of the CPU service demand in our experiments).
Bibliography


Thanks.

For slides go to:

http://www.cs.gmu.edu/faculty/menasce.html

and click on Talks (after July 16, 2004).