Using Performance Models to Design Self-Configuring and Self-Optimizing Computer Systems

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

© 2004 D. A. Menascé. All Rights Reserved.
Huge number of devices
Huge number of data sources
Many different data formats
Heterogeneous devices
Widely varying capacity
Wired and wireless
Widely varying QoS requirements

Node failures
Connectivity failures
Security attacks
Limited battery power
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
Important Technologies

- Web Services:
  - SOAP, UDDI, WSDL
- Grid Computing
- Peer to Peer Networks
- Wireless Networking
- Sensor and ad-hoc networks

Challenges

- Dynamically changing application structure.
- Hard to characterize the workload.
  - unpredictable
  - dynamically changing services
  - application adaptation
- Difficult to build performance models.
  - moving target
- Multitude of QoS metrics at various levels of a distributed architecture
  - response time, jitter, throughput, availability, survivability, recovery
time after attack/failure, call drop rate, access failure rate, packet delay,
packet drop rate.
- Tradeoffs between QoS metrics (response time vs. availability, response time vs. security)
Challenges (cont’d)

- Need to perform transient, critical time (e.g., terrorism attack or catastrophic failures) analysis of QoS compliance. Steady-state analysis is not enough.
- Mapping of global SLAs to local SLAs
  - Cost and pricing issues.
- QoS monitoring, negotiation, and enforcement.
- Platform-neutral representation of QoS goals and contracts.
- Resource management: resource reservation, resource allocation, admission control.
  - non-dedicated resources

What we need …

- Design self-regulating (autonomic systems)
- Embed within each system component:
  - Monitoring capabilities
  - Negotiation capabilities (requires predictive modeling power)
  - Self-protection and recovery capabilities (for attacks, failures, and overloads)
- Push more of the desired system features to individual components
- Design QoS-aware middleware
  - QoS negotiation protocols
  - Mapping of global to local SLAs
  - QoS monitoring
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.
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.
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
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.
Controller Interval

reconfiguration commands

i-th interval

measurements from servers in e-commerce site.

controller algorithm

(reconfiguration commands)

(i+1)-th interval

measurements from servers in e-commerce site.

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

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

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

- \(w_R\), \(w_P\), 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.

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.

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

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

- \( X^*_\text{min} = \min(\lambda, X^*_\text{min}) \)
- = 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.

\[ QoS = f(\hat{W}, c_1, c_2, \ldots, c_m) \]
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.

Hill-Climbing Search

© 2004 D. A. Menascé. All Rights Reserved.
Beam Search

A Queuing Model is Used to Compute QoS Values
Prototype Configuration

Experiment Results

Arrival rate
Results of QoS Controller

Experiment Results

Arrival rate

QoS is not met!
Variable inter-arrival 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)

<table>
<thead>
<tr>
<th>Cs</th>
<th>Average QoS</th>
<th>Controller Interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>No Controller</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>Beam Search</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>Hill Climbing</td>
</tr>
</tbody>
</table>

Cs = 1, Cs = 2

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

Ca = 1, Cs = 1

Controller Interval

Average QoS

No Controller  Beam Search  Hill Climbing

Ca = 2, Cs = 1

Controller Interval

Average QoS

No Controller  Beam Search  Hill Climbing

Ca = 4, Cs = 1

Controller Interval

Average QoS

No Controller  Beam Search  Hill Climbing

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

Extreme values for Ca and Cs

Ca = 4, Cs = 4

Controller Interval

Average QoS

No Controller  Beam Search  Hill Climbing

© 2004 D. A. Menascé. All Rights Reserved.
Dynamic Controller Interval and Workload Forecasting

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 \( C_a = C_s = 2 \)
Sensitivity of Controller to SLAs

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
Q-Applications and Q-components

- Q-application
  - Q-component
  - Q-component
  - Q-component
  - Q-component
  - Q-component

Service directory
- registration

- Q-component

Discovery
- Q-component

© 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
Architecture of a Q-component (QoS Negotiation)

QoS Request Handler

QoS Negotiator

QoS Evaluator

Performance Model Solver

Service Registration

Service Dispatcher

Service 1

... Service k

Table of QoS commitments

Architecture of a Q-component – Service Requests

Service Registration

Service Dispatcher

Service 1

... Service k

Table of QoS commitments

Performance Monitor
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>
On-time Accepted Counteroffer

Client

QoSRequest(rid, Sid, N, Rmax, Xmin)

CounterOffer (rid, N', token)

AcceptCounterOffer (token)

Request in ToC

Q-component

CounterOffer (rid, N', token)

AcceptCounterOffer (token)

Request in ToC

timeout

Expired Accepted Counteroffer

Client

QoSRequest(rid, Sid, N, Rmax, Xmin)

CounterOffer (rid, N', token)

AcceptCounterOffer (token)

ExpiredCounterOffer (rid)

Request in ToC

Request removed from ToC

timeout

Q-component

CounterOffer (rid, N', token)

AcceptCounterOffer (token)

ExpiredCounterOffer (rid)

Request removed from ToC

timeout
Rejected 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>RejectCounterOffer (token)</td>
<td>Request removed from ToC</td>
</tr>
</tbody>
</table>

Rejected 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></td>
</tr>
<tr>
<td>RejectQoSRequest (rid)</td>
<td></td>
</tr>
</tbody>
</table>
Decision Table for QoS Negotiation

1. Current and other requests are satisfied
   - Accept

2. Only Current is Violated
   - Only MAXR is violated
     - Decrease N
     - Current: OK
     - Others: OK
     - Decision: Counter Offer
     - OK: OK
     - Not OK: Reject
   - Only MINX is violated
     - Increase N
     - Current: OK
     - Others: OK
     - Decision: Counter Offer
     - OK: OK
     - Not OK: Reject
   - Both MAXR and MINX are violated
     - Decreasing N reduces R and increasing N increases R. So, there is no solution.
     - Reject

3. Only Current Request and Others are Violated
   - Only MAXR is violated
     - Decrease N
     - Current: OK
     - Others: OK
     - Decision: Counter Offer
     - OK: OK
     - Not OK: Reject
     - OK: OK
     - Not OK: Reject
   - Only MINX is violated
     - Increase N
     - Current: OK
     - Others: OK
     - Decision: Counter Offer
     - OK: OK
     - Not OK: Reject
     - OK: OK
     - Not OK: Reject
   - Both MAXR and MINX are violated
     - Decreasing N reduces R and increasing N increases R. So, there is no solution.
     - Reject

4. Only Other Requests are Violated
   - Any
     - Decrease N
     - Current: OK
     - Others: OK
     - Decision: Counter Offer
     - OK: OK
     - Not OK: Reject
     - OK: OK
     - Not OK: Reject

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

New Request: Sid = 3, N = 12

<table>
<thead>
<tr>
<th>Service</th>
<th>Commitment ID</th>
<th>Service ID</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Disk 1</td>
<td>2</td>
<td>3</td>
<td>24</td>
</tr>
<tr>
<td>Disk 2</td>
<td>3</td>
<td>1</td>
<td>20</td>
</tr>
</tbody>
</table>

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

Matrix of Service Demands (in msec)

<table>
<thead>
<tr>
<th>Service</th>
<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>
<td></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>
<td></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>
<td></td>
</tr>
</tbody>
</table>

Vector N: 10 15 8 20 13 12

© 2004 D. A. Menascé, All Rights Reserved.
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>7</td>
<td>10</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>102</strong></td>
<td><strong>1500</strong></td>
<td><strong>7</strong></td>
<td><strong>9</strong></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>25</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>266</strong></td>
<td><strong>1500</strong></td>
<td><strong>18</strong></td>
<td><strong>18</strong></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><strong>Total</strong></td>
<td><strong>102</strong></td>
<td><strong>1500</strong></td>
<td><strong>33</strong></td>
<td><strong>24</strong></td>
</tr>
</tbody>
</table>
Stock Quote Service

Real-time Stock Quote Response Time

Delayed Stock Quote Response Time

© 2004 D. A. Menascé. All Rights Reserved.
Concluding Remarks

- Performance models can be used to build QoS controllers for complex multi-tiered systems:
  - Controlled system provides 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.

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

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).

© 2004 D. A. Menascé. All Rights Reserved.
Bibliography


