

Evolution and Prospects of Quality of Service: A Personal Account

Calton Pu

Professor and J. P. Imlay, Jr. Chair in Software
Georgia Institute of Technology



History of the Term “QoS”

- Guaranteed network bandwidth
 - ◆ RSVP (1993): ReSerVation Protocol (Citeseer’s highest citation for QoS)
- Gradual broadening of QoS
 - ◆ DARPA’s Quorum program (1997-2000)
 - ◆ Statistical interpretation of resource allocation
 - ◆ Service-Level Agreements (SLAs)



#1: Quasar Project (OGI)

- With Jonathan Walpole (PI)
 - ◆ Approximately 1995 – 2002
- Dynamic adaptation for system software
 - ◆ Specialization of system software
 - ◆ Adaptive resource allocation
 - ◆ Distributed multimedia, real-rate applications

Real-Rate Applications

- Real-Rate applications
 - ◆ IP telephony
 - ◆ Videoconferencing
 - ◆ Remote sensors over network
- Common QoS requirements
 - ◆ *Delay-sensitive*: data must be delivered from sender to receiver in bounded time.
 - ◆ *Self-rate-regulated*: e.g. live video source from camera



Cyber Physical Systems

- Mission-critical applications
 - ◆ SCADA systems
 - ◆ Medical devices
 - ◆ Avionics software
- Need QoS (broad def.)
 - ◆ Dependability, predictable performance, security, availability, ...

Georgia Tech College of Computing

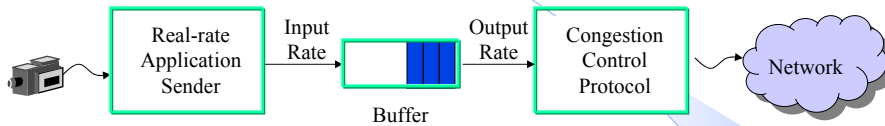
Real-Rate Applications in the Internet

```

    graph LR
      subgraph Sender_Side [Sender-Side]
        DE[Data Encoding]
        B1[Buffer]
        SCC[Sender of Congestion Control]
        DE --> B1
        B1 --> SCC
      end
      subgraph Receiver_Side [Receiver-Side]
        RSC[Receiver of Congestion Control]
        B2[Buffer]
        DD[Data Decoding]
        RSC --> B2
        B2 --> DD
      end
      SCC --- Network((Network))
      Network --- RSC
      RSC -.->|Congestion Control Feedback| SCC
  
```

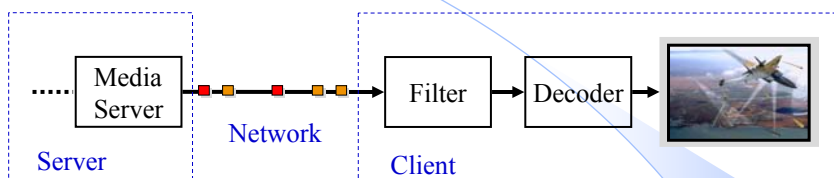
- A real-rate application using a congestion control protocol.

Real-Rate Application



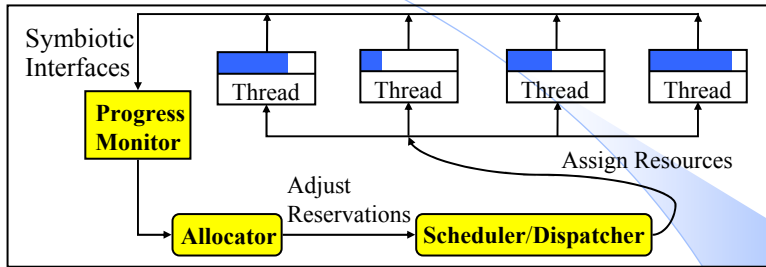
- Buffering delay between real-rate application & congestion control protocol:
 - ◆ It is caused by rate mismatches,
 - Input rate is controlled by the application adaptation.
 - Output rate is controlled by the congestion control.
- TCP-friendly congestion control

FB-Based CPU Scheduling



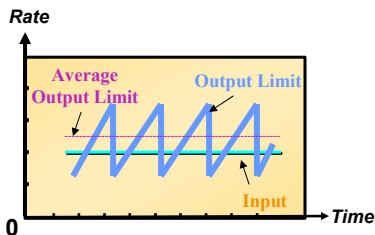
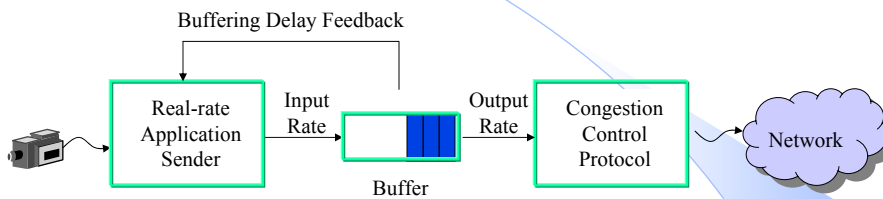
- Real-rate applications
 - ◆ Real-world performance demands
- Automated rate matching
 - ◆ fine-grain adjustment of allocation
 - ◆ dynamic response to variable rates
 - ◆ low programming complexity

Progress-Based Allocation



- Application progress monitor
- Resource reservation scheduler
- Dynamic allocator:
 - ◆ Feedback based allocator samples progress
 - ◆ Calculates resource needs, assigns allocation

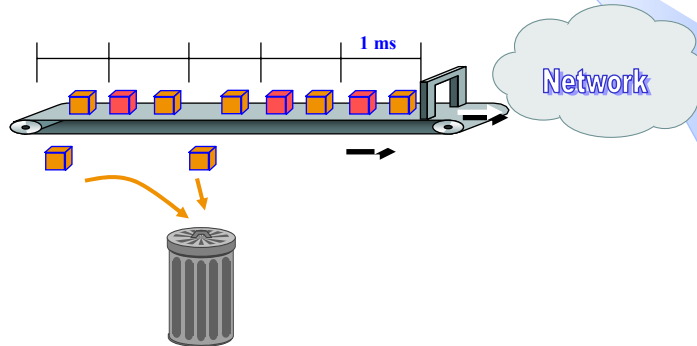
Network QoS Adaptation



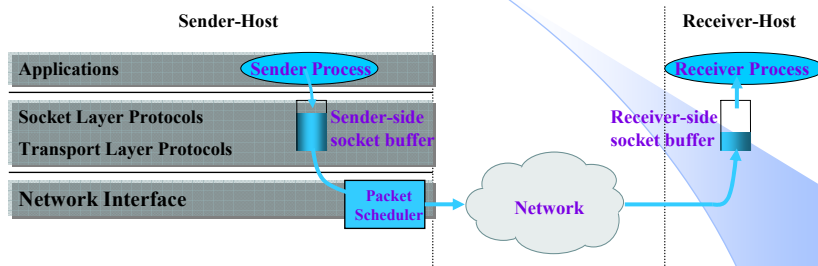
- Adaptation: Reducing input rate to be less than the average of output.
- System goes to the finite source case!

Bandwidth Reservation Mechanism

- Each stream is guaranteed a bandwidth *proportion (%)* over some *period (ms)*

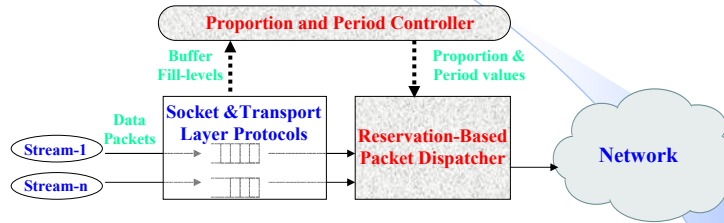


Inferring Requirements



- Monitor sender and *receiver socket buffer fill-levels*
- Receiver's socket buffer is on a remote host!
- Data rate requirements change dynamically

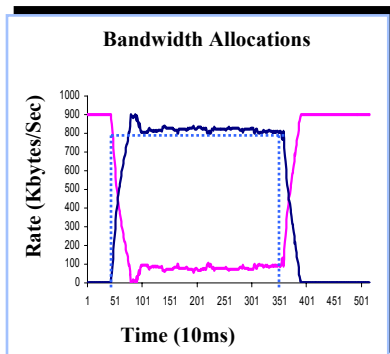
Packet Scheduler Architecture



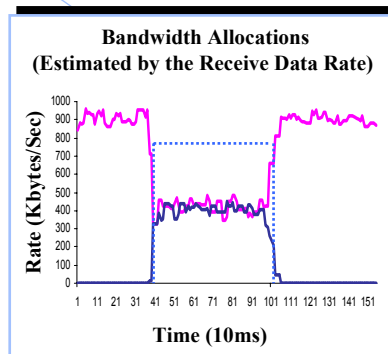
- Packet dispatcher - *reservation-based mechanism*
- Proportion and period controller - *progress-driven policy*

Competition for Bandwidth

Progress-driven Packet Scheduler



Linux Packet Scheduler

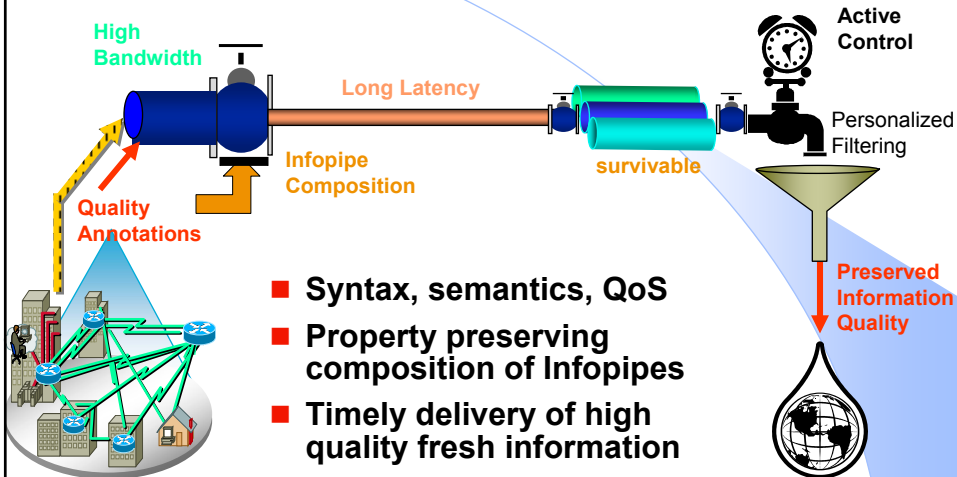


.... Real-Rate Stream's Bandwidth Requirement
 — Real-Rate Stream's Bandwidth Allocation
 — Best-effort Stream's Bandwidth Allocation

Broad Definition of QoS

- DARPA programs supporting QoS work
 - ◆ Quorum (1997)
 - ◆ Software-Enabled Control (SEC, 1998)
 - ◆ Program Composition for Embedded Systems (PCES, 2000)
 - ◆ Adaptive and Reflective Middleware Systems (ARMS, 2002)
 - ◆ Credits to D. Tennenhouse, H. Gill, J. Sztipanovits, D. Schmidt

#2: Infosphere Project

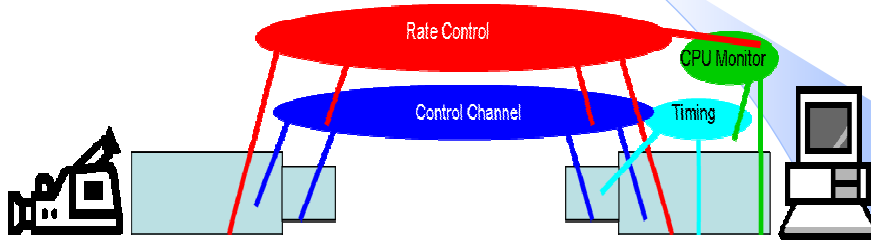


- Syntax, semantics, QoS
- Property preserving composition of Infopipes
- Timely delivery of high quality fresh information

■ Needs an *information-centric abstraction* that is *network aware* – the Infopipe

Video Service with QoS

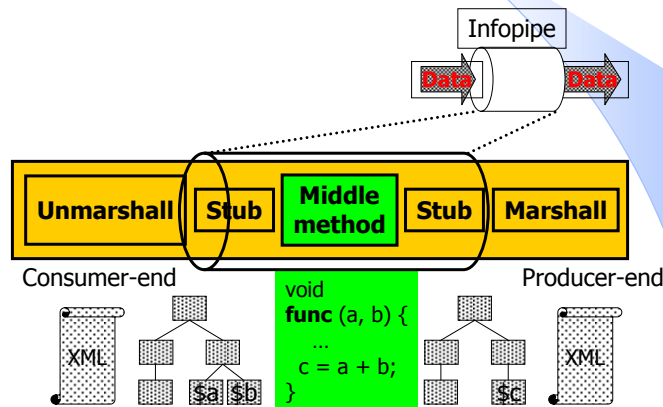
- Receiver subscribes to stream from video source
- Receiver must not be swamped or starved by video server



- Things to note:
 - ◆ Code for QoS gets distributed
 - ◆ QoS code interleaved

Infopipe - A Look Inside

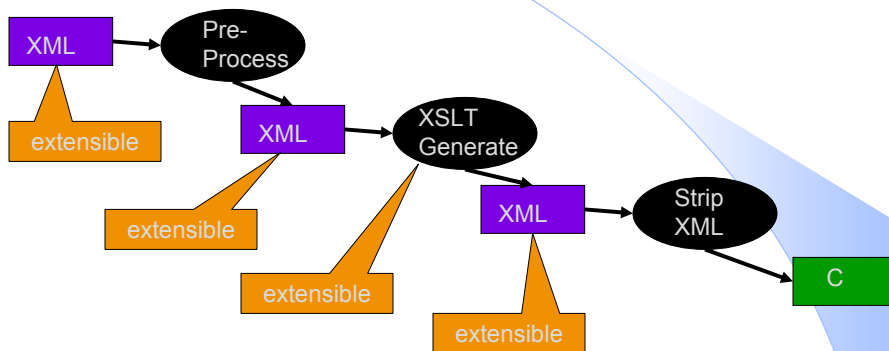
- Generate stub/marshalling code (for C and Java)



Infopipe Support for QoS

- Information flows contain time-sensitive information
 - ◆ QoS must be handled from end-to-end
- Adaptation may be distributed, too
 - ◆ Can make applications “brittle” with time
- Infopipes often heterogeneous
 - ◆ equipment, languages

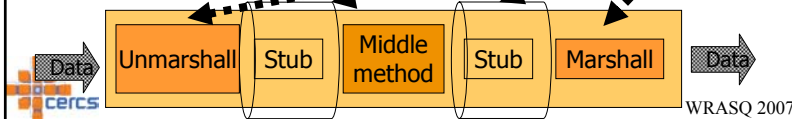
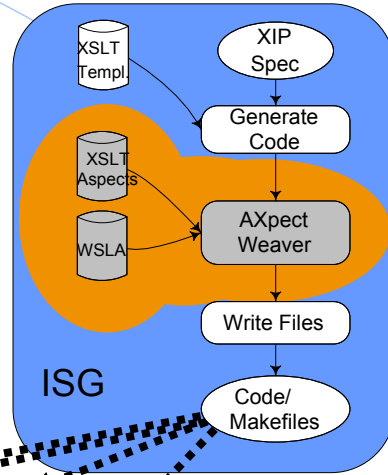
Infopipe Stub Generator



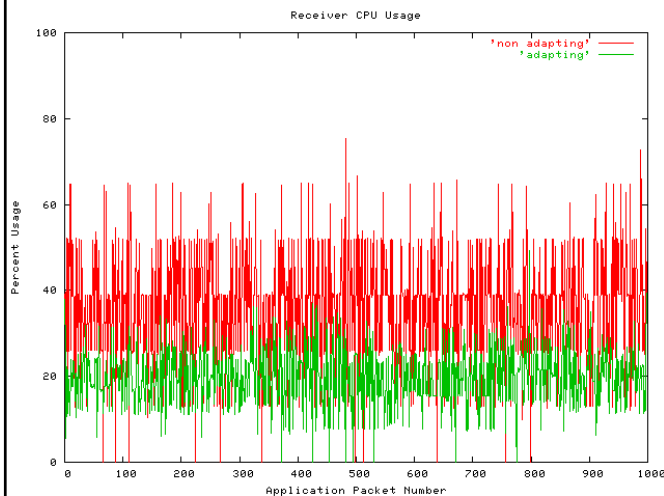
Multi-stage transformation of XML to XML

Generating Infopipe Code

- XIP in, code out
- C or C++ language output
- Choice of communication pkg



Does it run?



RED – No resource control

Avg: 36.1%

GREEN – CPU usage limited to 20% ±5% based on sender side adaptation

Avg: 19.8%

Why It Works

- XML Intermediate
 - ◆ Extensible – new data doesn't break the document
 - ◆ Store the entire system
- XPath – lets us ignore what we can't understand
- XSLT
 - ◆ XML, and is extensible
 - ◆ XSLT can read multiple documents
- Leverage our DSL –
 - ◆ We know something at the system level
 - ◆ We don't throw away perfectly good information

AXpect - Aspect

```

<xsl:template
  match="//filledTemplate[@name=$pipename]
          [@inside=$inside]//jpt:pipe-middle">
  struct timeval base;
  struct timeval end;
  <jpt:time-process>
  // take timing here
  gettimeofday(&base,NULL);
  <xsl:copy>
    <xsl:apply-templates select="@*|node()" />
  </xsl:copy>
  gettimeofday(&end,NULL);
  usec_to_process = (end.tv_sec - base.tv_sec ) *
                    1e6 + (end.tv_usec - base.tv_usec);
  fprintf(stdout,"Time to process: %ld\n", usec_to_process);
  </jpt:time-process>
</xsl:template>

```

XSLT
C code
Pointcut
Joinpoint

2: Sender WSLA

- Implements SLA pieces for sender
- Param'ed by the WSLA
- Can send messages over the control channel
- Responds to receiver feedback

The Complete Application

Aspect	Lines
control_sender.xsl	117
sla_sender.xsl	73
sender.xsl	30
control_receiver.xsl	125
timing.xsl	50
cpumon.xsl	14
sla_receiver.xsl	55
receiver.xsl	18
TOTAL	482

Where the Code Goes

Receiver-side											
Aspect	Affected File	Makefile	receiver.h	receiver.c	ppmIn.h	ppmIn.c	control.h	control.c	sla.c	sla.h	# Lines Added
timing			X		X						50
control_receiver		X	X	X	X	(X)	(X)				125
cpumon			X								14
sla_receiver		X	X			X	X	(X)	(X)		55

QoS code affects 13 of 18 files (from 6 AXpect files)

Sender-side											
Aspect	Affected File	Makefile	sender.h	sender.c	ppmOut.h	ppmOut.c	control.h	control.c	sla.c	sla.h	# Lines Added
control_receiver		X			X	(X)	(X)				117
sla_receiver		X			X	X	X	(X)	(X)		73

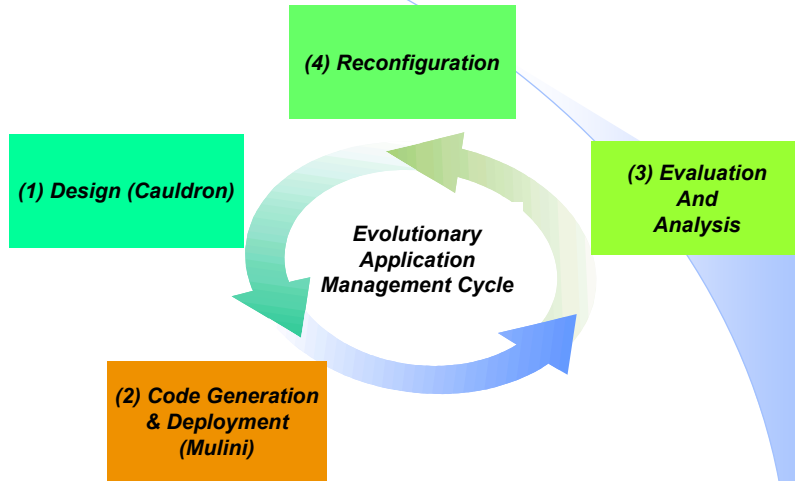
QoS code is $\approx 30\%$ of total

Total Aspect Lines	434
Base Implementation	976
Complete Application	1410

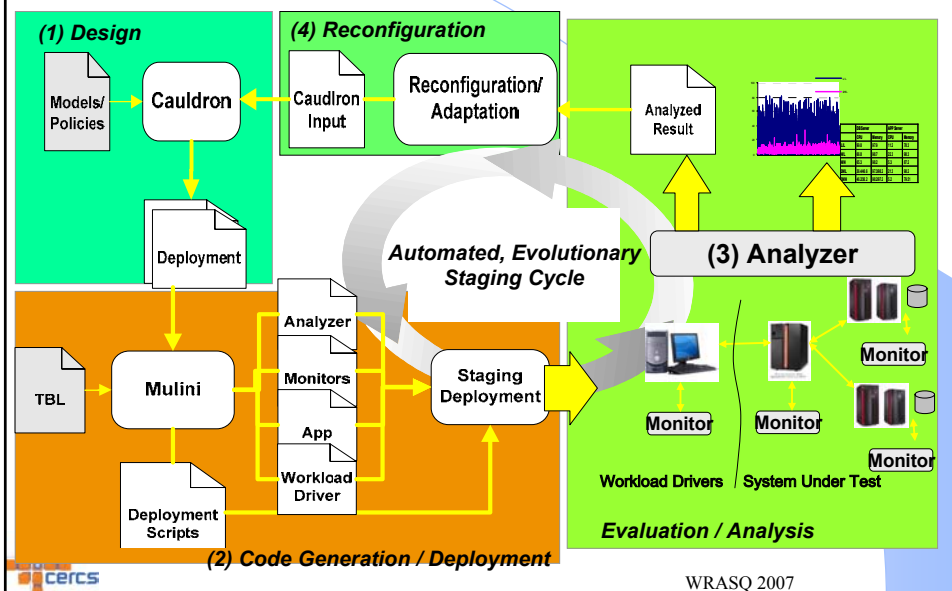
#3: Elba Project

- Goal: Automate staging cycle to achieve SLO/SLA
 - ◆ Systems design, deployment, evaluation, and reconfiguration
- Generate / translate staging plan
 - ◆ Reuse system deployment specifications
 - ◆ Reuse policy specifications
 - ◆ Declarative description of staging parameters
 - ◆ Automation works (NOMS'06)

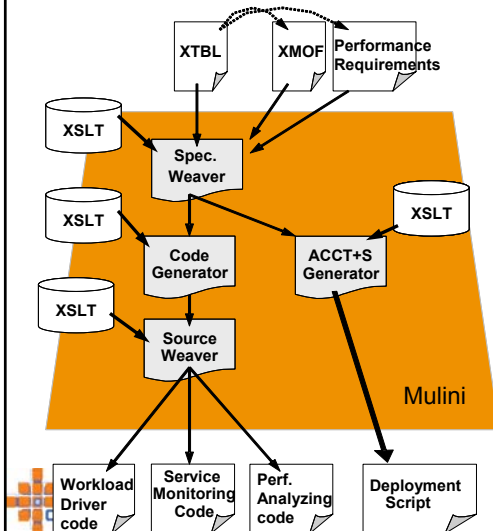
Application Management



Elba: Overall Picture



Elba: Mulini Code Generator



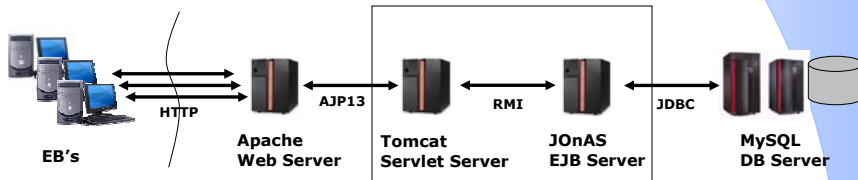
- Based on *Clearwater* approach (ASE'05)
 - ◆ XML-based specifications
 - ◆ XSLT templates for generation
- Maps high-level abstractions to low-level staging code
- Instruments application using aspect weaving technology

Code Generation Example

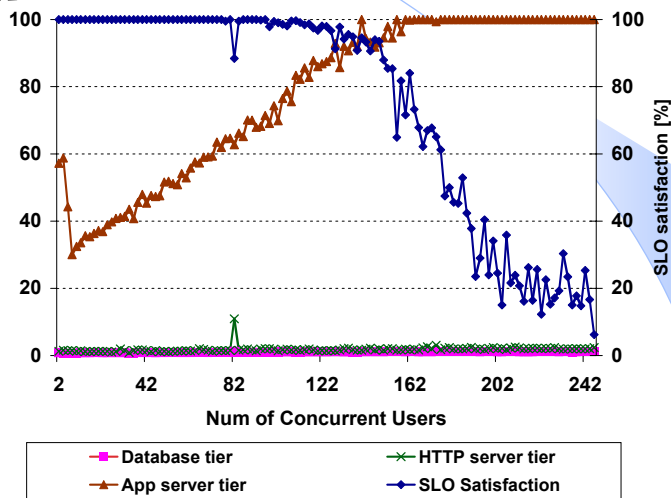
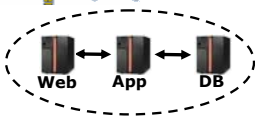
<pre>instance of LogicalServer { Id = "Tomcat_LS1"; Caption = "Tomcat Logical Server"; Description = "Logical Server for Tomcat "; IpAddress = "130.207.5.228"; HostName = "artemis.cc.gatech.edu"; }; instance of LogicalServerInLogicalApplication { LogicalApplication = "Tomcat"; LogicalServer = Tomcat_LS1"; }; instance of LogicalApplication { Id = "Tomcat"; Version = "5.0.19"; Caption = "Tomcat"; Description = "Tomcat application Server"; }; instance of LogicalApplication { Id = "MySQLDriver"; Version = "3.0.11"; Caption = "MySQLDriver"; Description = "MySQL driver"; }; instance of Activity { Id = "Tomcat_Installation"; ActivityType = "script"; }; instance of Activity { Id = "Tomcat_Installation"; ActivityType = "script"; }; instance of ActivityPredecessorActivity { DependenceType="Finish-Start"; AntecedentActivity="Tomcat_Installation"; DependentActivity="MySQLDriver_Installation"; };</pre> <p>(a) MOF</p>	<pre><Instance Class="LogicalApplication"> Name="Tomcat" <Variable Name="Id" Type="string">Tomcat</Variable> <Variable Name="Version" Type="string"> 5.0.19</Variable> <Variable Name="Entity" Type="string"> Activity_Tomcat_Installation</Variable> <Variable Name="Host" Type="string"> artemis.cc.gatech.edu</Variable> <Instance> <Workflow> <Work Type="Execution"></Work> <Work Type="EventSend"> <To> MySQLDriver_Installation</To></Work> <Work Type="Terminate"> Tomcat_Installation </Work> </Workflow> </Instance> <Instance Name="MySQLDriver" Class="LogicalApplication"> <Variable Name="Id" Type="string"> MySQLDriver</Variable> <Variable Name="Version" Type="string"> 3.0.11</Variable> <Variable Name="Entity" Type="string"> Activity_MySQLDriver_Installation</Variable> <Variable Name="Host" Type="string"> demeter.cc.gatech.edu</Variable> </Instance> <Workflow> <Work Type="OnEvent"> <From> Tomcat_Installation</From> </Work> <Work Type="Execution"></Work> <Work Type="Terminate"> MySQLDriver_Installation</Work> </Workflow></pre> <p>(b) XMOF</p>	<pre>sfProcessComponentName "Tomcat_Installation"; LogicalApplication_Tomcat extends LogicalApplication { Id "Tomcat"; Version "5.0.19"; Activity LAZY ATTRIB Activity_tomcat_installation; sfProcessHost "artemis.cc.gatech.edu"; } Activity_Tomcat_Installation extends Activity { Id "Tomcat_Installation"; Entity LAZY ATTRIB LogicalApplication_Tomcat; - extends EventSend { sendTo eventQueue:queue_Tomcat_Ignition; event "Activity_Tomcat_Installation_FS"; } -- extends Terminator { kill eventQueue:queue_Tomcat_Installation; } sfProcessComponentName "MySQLDriver_Installation"; -- extends OnEvent { registerWith queue_MySQLDriver_Installation ; Activity_Tomcat_Installation_FS extends DoNothing } LogicalApplication_MySQLDriver extends LogicalApplication { Id "MySQLDriver"; Version "3.0.11"; Activity LAZY ATTRIB Activity_MySQLDriver_Installation; sfProcessHost "demeter.cc.gatech.edu"; } Activity_MySQLDriver_Installation extends Activity { Id "MySQLDriver_Installation"; Entity LAZY ATTRIB LogicalApplication_MySQLDriver; - extends Terminator { kill eventQueue:queue_MySQLDriver_Installation; }</pre> <p>(c) SmartFrog</p>
---	---	---

Experimental Platforms

- Software
 - ◆ RUBiS (Multi-tier on-line auction)
 - ◆ Apache/Tomcat-JOnAS/MySQL
 - ◆ Redhat FC4
- Hardware: Netlab/Emulab

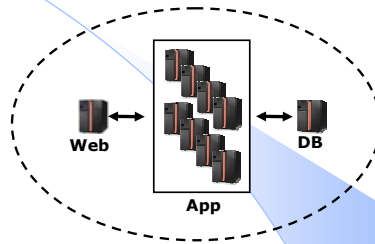


CPU Utilization



Scale of Experiment

Sample Metrics	
CPU utilization	
Memory utilization	
Paging rate	
Disk I/O rate	
Network bandwidth	
EJB cache size	
# of EJB instances	
# of connections to DB	
# of transactions	
# of threads	
# of DB connections	

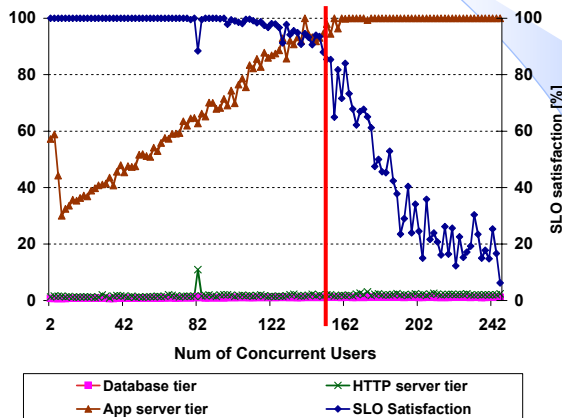


# metrics per node	40
# experiment sets ★	496
# nodes used ★	6944
# metric data pts collected	785,000,000

- ★ data not included in article
- ★ consecutively

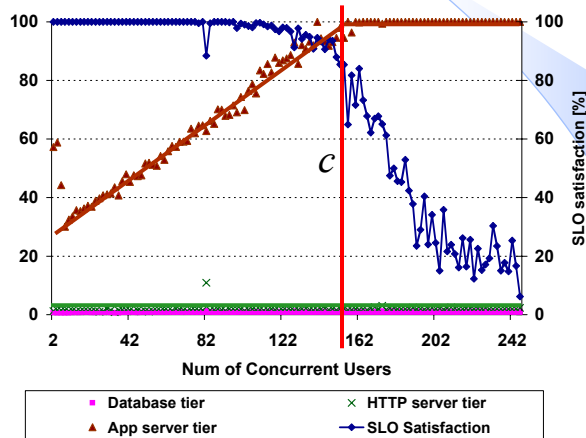
Detection Model

- Idea: Model plateau changes in the slopes of the metrics that correspond to the deterioration of the SLO-satisfaction.



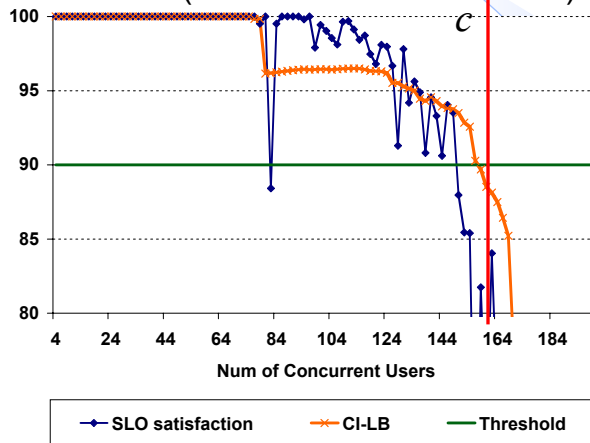
Intervention Analysis

- Transform to final form using the plateau property:



Determining an Intervention Point *c*

- Idea: Use (95%) confidence interval approximation for SLO satisfaction (at 90% of all transactions)



Determining an Intervention Point c

- **Idea:** Use confidence interval approximation for satisfaction function.

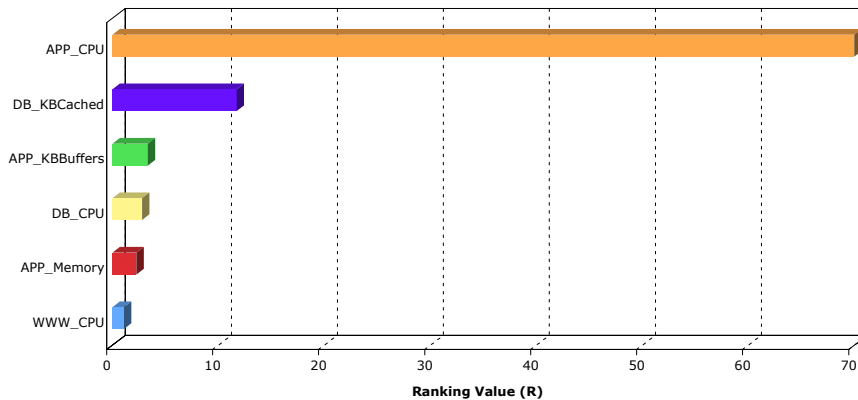
- Assumption of abrupt deterioration:

$$\forall i \in I: SAT_i \approx \text{const} \quad \forall i \in I': SAT_i \ll \frac{1}{|I|} \sum_{j \in I} SAT_j$$

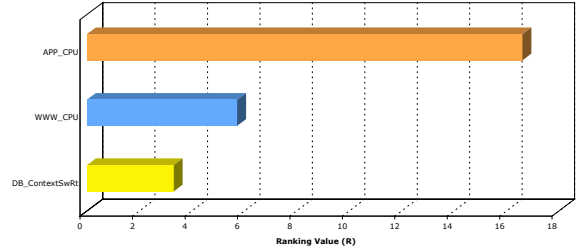
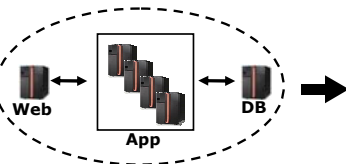
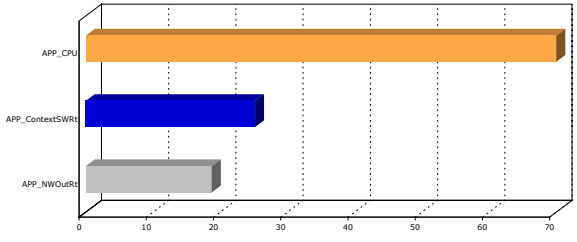
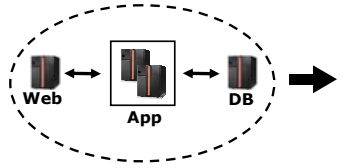
- Test confidence interval lower bound above ninety percent:

$$90\% \leq \frac{1}{n_0} \sum_{0 \leq i \leq w_0} SAT_i - \frac{1.96}{\sqrt{n_0 - 1}} \sqrt{\sum_{0 \leq i \leq w_0} (SAT_i - \frac{1}{n_0} \sum_{0 \leq j \leq w_0} SAT_j)^2}$$

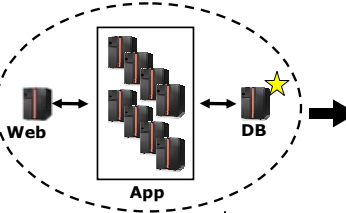
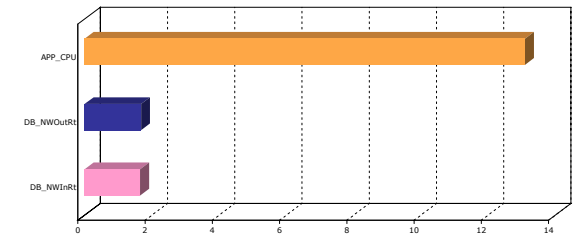
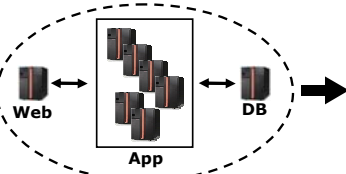
Results of Detection



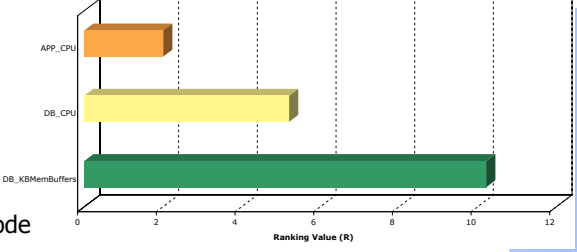
Scale-Out Analysis (a)



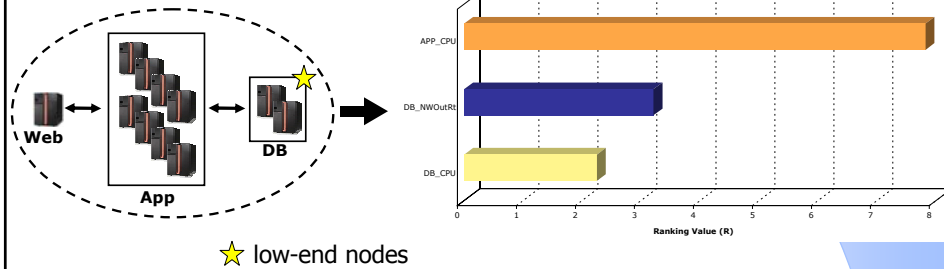
Scale-Out Analysis (b)



★ low-end node



Scale-Out Analysis (c)



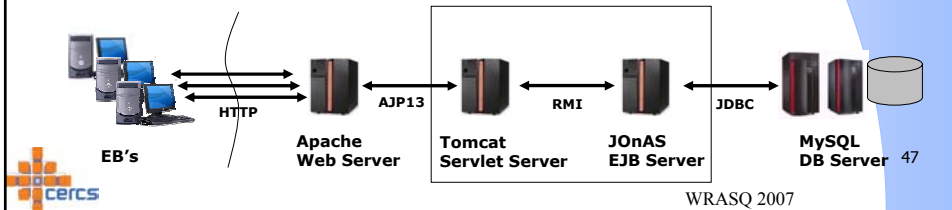
Some Lessons Learned

- Many of QoS properties are end-to-end
 - ◆ They are limited by the weakest link (or a single bottleneck) in the system
 - ◆ They require the cooperation of many system layers and components to reserve resources, e.g., RSVP
- These are *vertical* QoS properties

Vertical QoS Property

- Predictable performance (in the guaranteed sense) requires cooperation from all components

$$RTime = \sum WebServer + AppServer + DBServer$$



Horizontal QoS Property

- Predictable performance (in a statistical sense)
 - ◆ Elba: SLO measurement and automated analysis (90% SLO satisfaction level)
 - ◆ Infopipe: dynamic adaptation through automated composition (feedback-based control)
 - ◆ Can be “added on top”

Vertical QoS Property

- System availability (e.g., MTBF)
 - ◆ Requires built-in redundancy of all vulnerable components
 - ◆ System availability depends on the weakest link (least reliable components)

Horizontal QoS Property

- System availability through scale-out
 - ◆ N-Tier applications (RUBiS): add more App Servers to relieve performance bottleneck and increase availability
 - ◆ Automatically generated configurations
 - ◆ Can be “added on top”

Vertical QoS Aspects

- Some “difficult” QoS properties
 - ◆ Security (of software and information flow)
- Example: web browser information flow
 - ◆ Security-sensitive information should be isolated from millions of lines of “normal” code due to rich functionality
 - ◆ Isolation requires refactoring of all system layers (from web browser to OS kernel)

Information Analog of QoS

- Denial of service as degradation of QoS
- Denial of information as degradation of Quality of Information (QoI)
 - ◆ Examples of QoI: trust, privacy, accuracy, completeness, timeliness, spam-free, deception-free

Examples of DoI

- Known BIG problems
 - ◆ Email spam, web spam, blog spam,
- Problems that are getting big
 - ◆ Social network spam
 - ◆ Ad click fraud, phishing and identity theft
- Anticipated problems
 - ◆ VoIP spam (spit)

Vertical QoI Properties

- Privacy: limited by any single leak
- Trusted system software: limited by any single penetration
- Accuracy, completeness, timeliness (in the guaranteed sense)

Horizontal QoI Properties

- Accuracy, completeness, timeliness (in a statistical sense)
 - ◆ The difference between the Web and databases
- Spam-free, deception-free
 - ◆ 80% of spam is filtered in the backbone (MAAWG), and 80% at email servers
 - ◆ Defenses against spam and deception

Summary

- QoS research has come a long way
 - ◆ Definition of QoS continuously broadened
 - ◆ “Horizontal” QoS properties can be separated and combined (e.g., using AOP)
 - ◆ “Vertical” QoS properties pervade many system layers and components
- QoI research just starting