QoS-aware resource allocation and load-balancing in enterprise Grids using online simulation

Samuel Kounev (skounev@acm.org)
Joint work with
Ramon Nou (ramon.nou@bsc.es)
Ferran Julia (fjulia@ac.upc.edu)
Jordi Torres (torres@ac.upc.edu)

* *Universität Karlsruhe (TH)*
Technical University of Catalonia (UPC)
Barcelona Supercomputing Center (BSC)

Research Interests

Performance Engineering

Performance Measurement	• Platform benchmarking • Application profiling • Workload characterization • System load testing • Performance tuning and optimization
System Modeling & Simulation	• System architecture models • Analysis-oriented performance models • Performance prediction at design & deployment time • System sizing and capacity planning
Run-time Performance Management	• Dynamic system models • Online performance prediction • Autonomic resource management • Utility-based optimization • Energy efficient computing

Technology Domains

Distributed Component-based Systems

- Enterprise Java
- Microsoft .NET

Service-oriented Environments

- Web Services
- Service-oriented Grids

Event-based Systems

- Message-oriented middleware
- Distributed publish/subscribe systems
- Sensor-based systems
- RFID applications

MOTIVATION

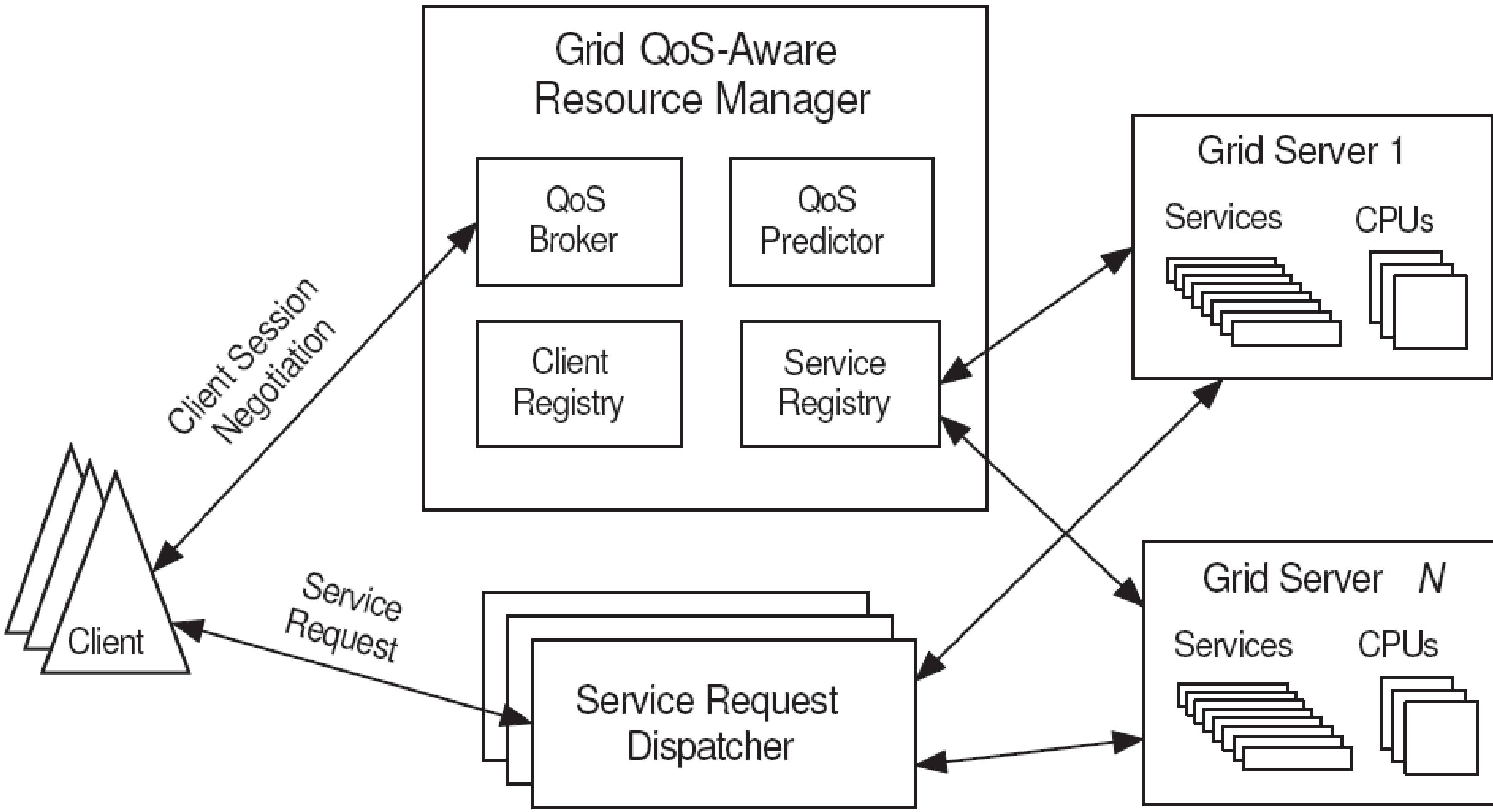
QoS-aware Resource Management in Grid Computing

Motivation

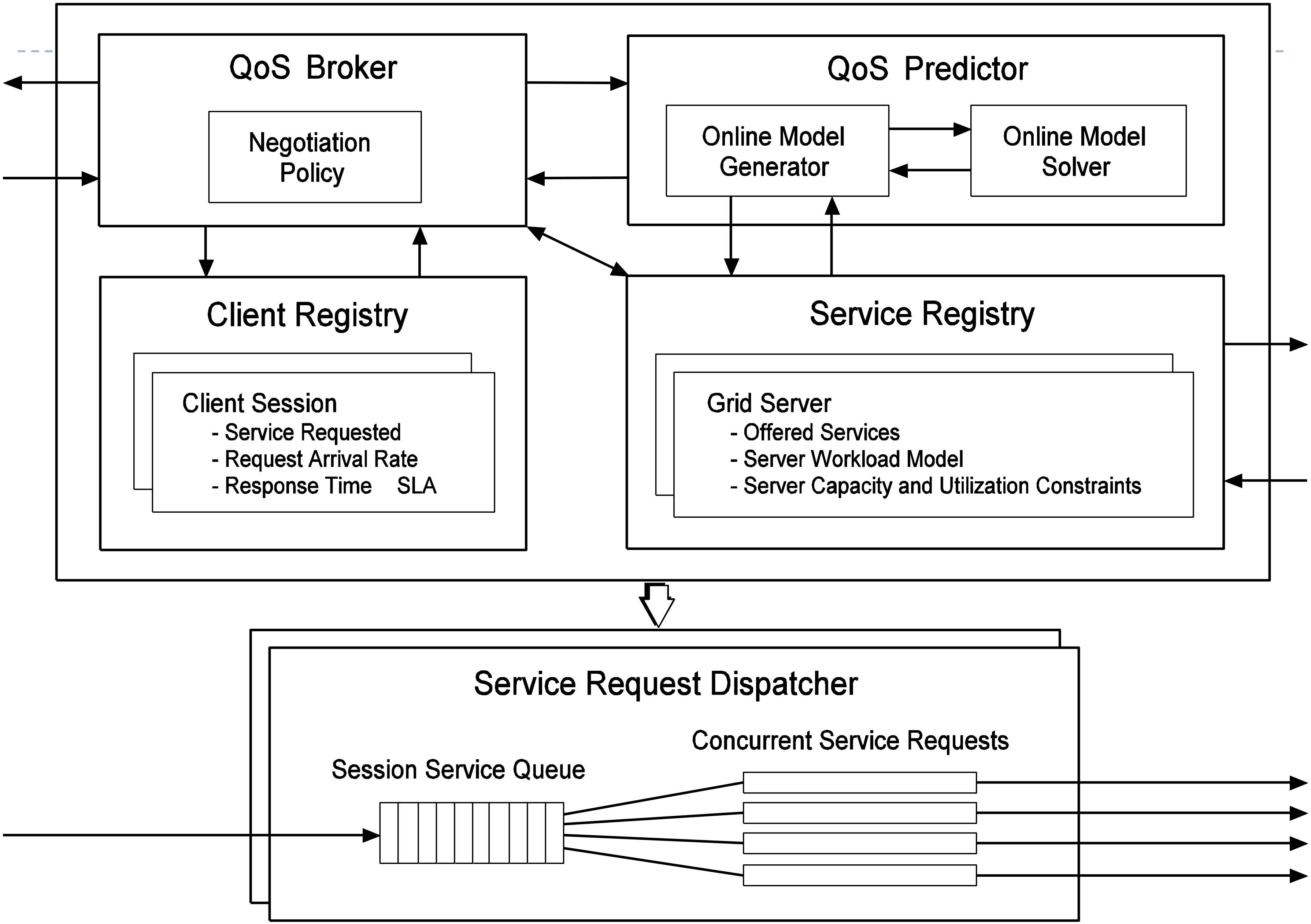
- Grid computing gaining grounds in the enterprise domain
- Grid and SOA technologies converging
- Enterprise Grid environments highly dynamic
  - Unpredictable workloads
  - Non-dedicated resources
- QoS management a major challenge
- Off-line capacity planning no longer feasible
- Methods for on-the-fly performance prediction needed
  - Can be used for QoS-aware resource management and
  - Utility-based performance optimization

QoS-AWARE RESOURCE MANAGER ARCHITECTURE

Resource Manager Architecture

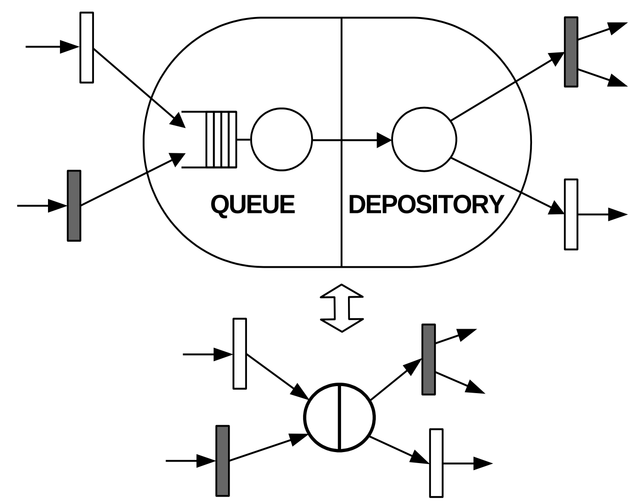


Resource Manager Architecture (2)



Queueing Petri Nets

- Combine Queueing Networks and Petri Nets
- Allow integration of queues into places of PNs
- Ordinary vs. Queueing Places
- **Queueing Place** = Queue + Depository



- Advantages:
  - Combine the modeling power and expressiveness of QNs and PNs.
  - Easy to model synchronization, simultaneous resource possession, asynchronous processing and software contention.
  - Allow the integration of hardware and software aspects.

QPME

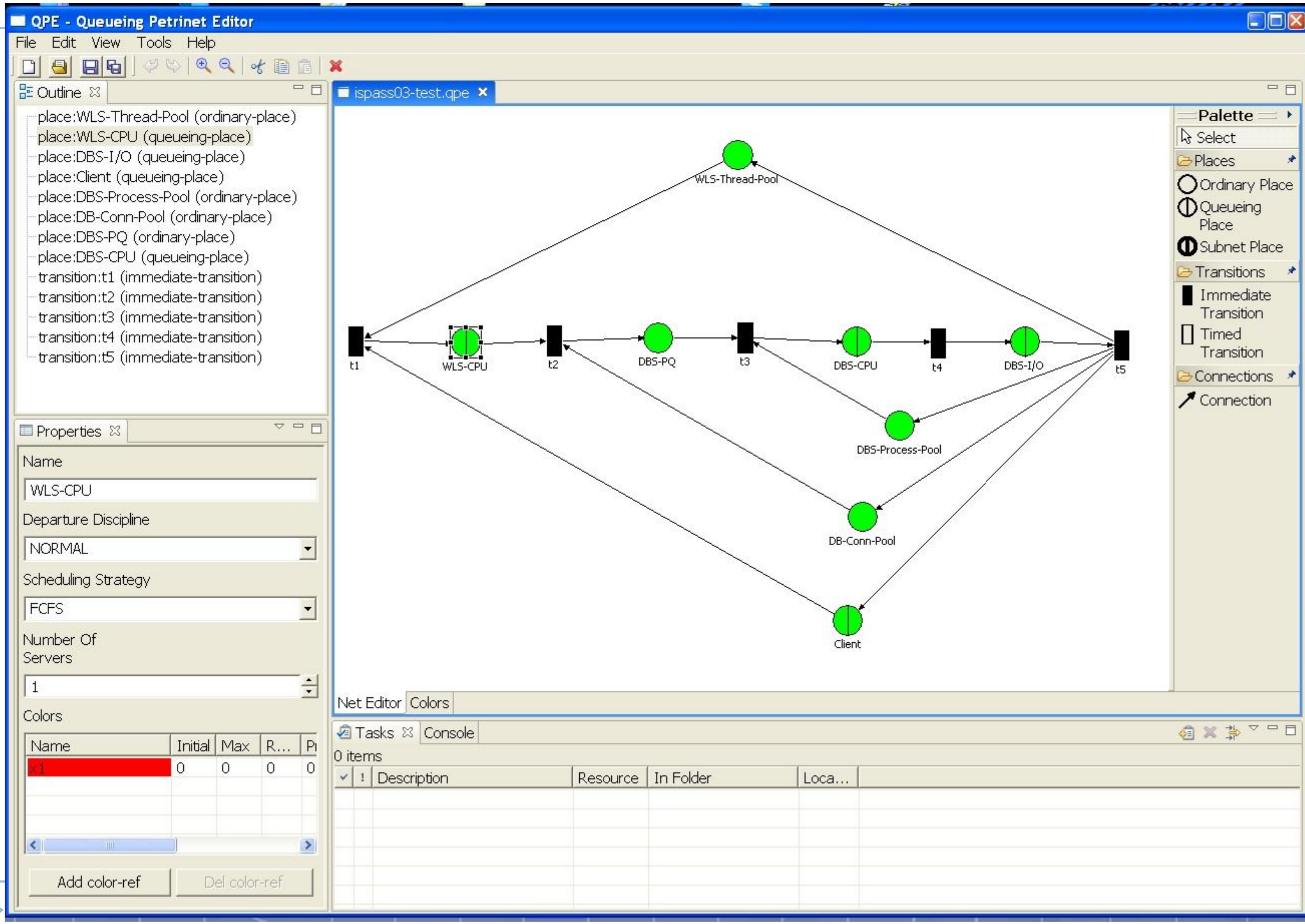
- A performance modeling tool based on QPNs
- QPME = Queueing Petri net Modeling Environment
- QPN Editor (QPE) and Simulator (SimQPN)
- Based on Eclipse/GEF
- Provides a user-friendly graphical user interface
- http://sdq.ipd.uka.de/people/samuel_kounev/projects/QPME

QPME (2)

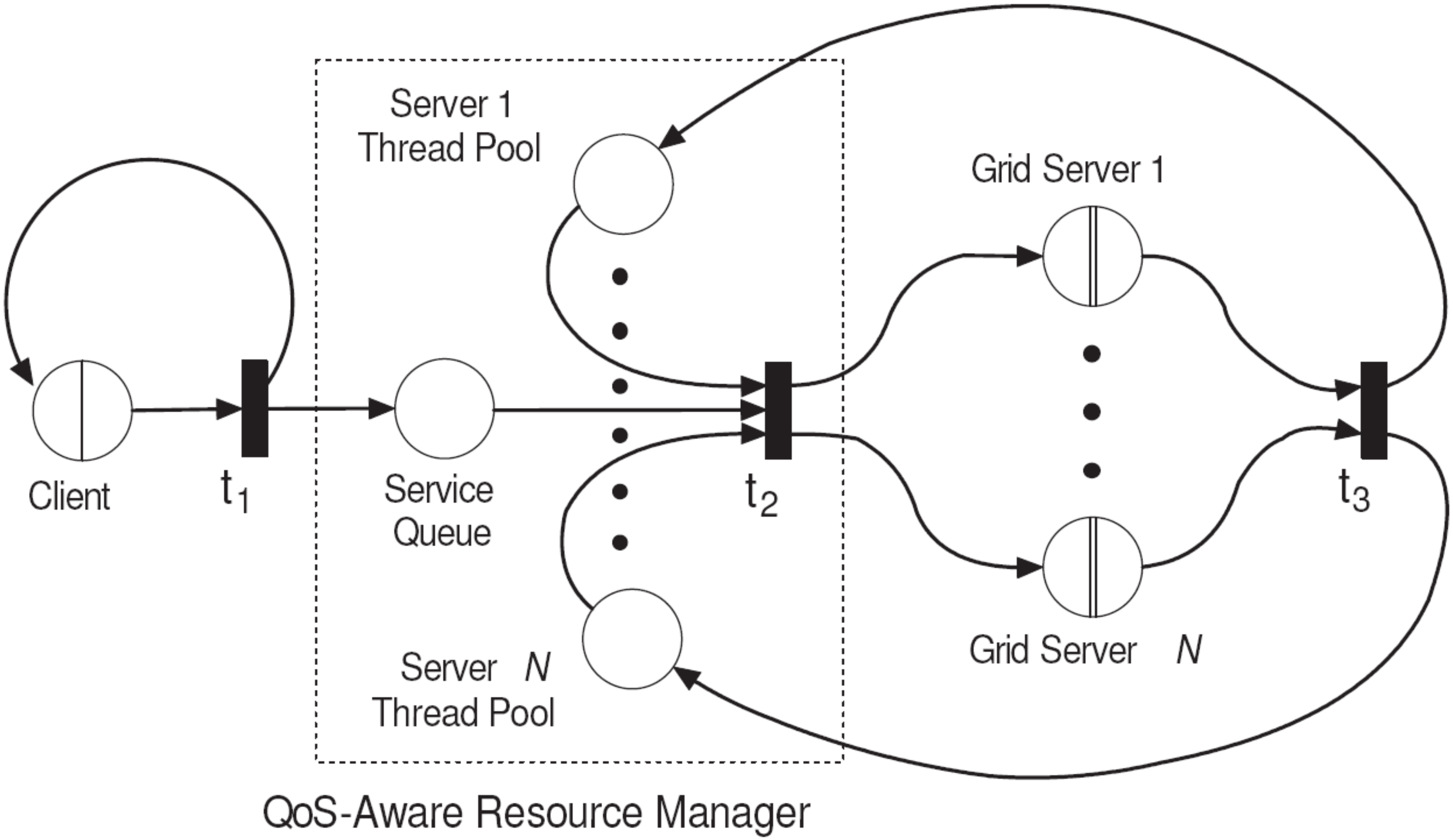
- First version released in January 2007
- Distributed to more than 70 research organizations worldwide
- Areas of usage
  - Online QoS control
  - Software performance engineering
  - Construction modeling and simulation area
  - Satellite communications
  - Dependability of safety-critical real time systems
  - Computational biology, modeling biological interaction networks
  - Logistics planning
  - Models of information flows

S. Kounev, R. Nou, F. Julia, J. Torres

QPME Screenshot



QoS Predictor



S. Kounev, R. Nou, F. Julia, J. Torres

Resource Allocation Algorithm

- New session request (v, λ, ρ) arrives
- Assign new session unlimited # threads on each server
- If required throughput cannot be sustained, reject request
- For each over-utilized server limit the number of threads
- If an SLA of an active session is broken, reject request
- Else if SLA of the new session broken, send *counter offer*
- Else accept request

Resource Allocation Algorithm

$S = \{s_1, s_2, \ldots, s_m\}$ Grid servers

$V = \{\upsilon_1, \upsilon_2, \ldots, \upsilon_n\}$ Services offered

$F \in [S \to 2^V]$ Services offered by a server

$\mathrm{C} = \{\mathrm{c}_1, \mathrm{c}_2, \ldots, \mathrm{c}_l\}$ Active client sessions where $\mathrm{c}_i = (\upsilon, \lambda, \rho)$

For $\mathrm{s} \in \mathrm{S}$:

$\mathrm{P(s)}$ server capacity (e.g. # CPUs)

$\overline{\mathrm{U}}(\mathrm{s})$ maximum utilization constraint

$\mathrm{T} \in [\mathrm{C} \times \mathrm{S} \to \mathrm{N}_0 \cup \{\infty\}]$ Thread allocation function

Resource Allocation Algorithm (2)

Define the following predicates

$$P_X^T(c) \text{ for } c \in C \text{ as } X^T(c) = c[\lambda]$$

$$P_R^T(c) \text{ for } c \in C \text{ as } R^T(c) <= c[\rho]$$

$$P_U^T(s) \text{ for } s \in S \text{ as } U^T(s) <= \overline{U}(s)$$

Configuration T is acceptable iff

$$\forall c \in C : P_X^T(c) \wedge P_R^T(c) \wedge (\forall s \in S : P_U^T(s))$$

Define also

$$A^T(s) = (\overline{U}(s) - U^T(s)) P(s)$$

$$I^T(\upsilon, \varepsilon) = \{ s \in S : \upsilon \in F(s) \wedge A^T(s) \geq \varepsilon \}$$

Resource Allocation Algorithm (3)

$C := C \cup \{\tilde{c}\}$
for each $s \in I^T(v,\epsilon)$ **do** $T(\tilde{c},s) := \infty$
if $(\exists\, c \in C : \neg P_X^T(c))$ **then reject** $\tilde{c}$
while $(\exists\, \hat{s} \in S : \neg P_U^T(\hat{s}))$ **do**
begin
$\quad T(\tilde{c},\hat{s}) := 1$
$\quad$ **while** $P_U^T(\hat{s})$ **do** $T(\tilde{c},\hat{s}) := T(\tilde{c},\hat{s}) + 1$
$\quad T(\tilde{c},\hat{s}) := T(\tilde{c},\hat{s}) - 1$
end
if $(\exists\, c \in C \setminus \{\tilde{c}\} : \neg P_X^T(c) \vee \neg P_R^T(c))$ **then reject** $\tilde{c}$
if $(\neg P_X^T(\tilde{c}) \vee \neg P_R^T(\tilde{c}))$ **then**
$\quad$ **send counter offer** $o = (v, X^T(\tilde{c}), R^T(\tilde{c}))$
else accept $\tilde{c}$

Workload Characterization On-The-Fly

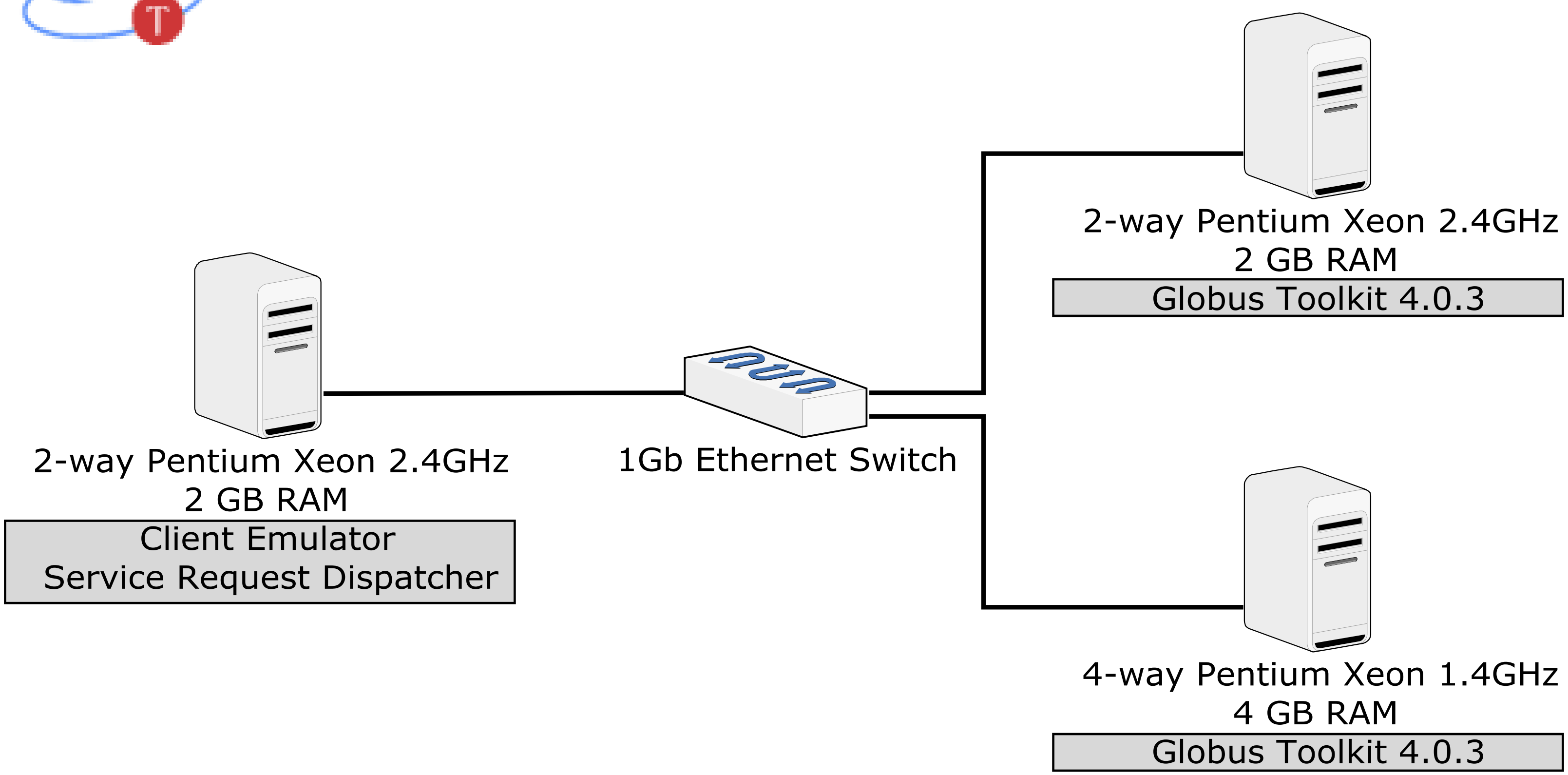
- What if no service workload model is available?
- Assumptions
  - Each service executes CPU-intensive business logic
  - No internal parallelism
  - Might call external third-party services
- Basic algorithm for estimating the CPU service times
  - Monitor service response time on each server
  - Iteratively, set *estimate* to lowest observed response time
- Enhanced algorithm
  - Monitor CPU utilization
  - Break down the measured response time into
    - Time spent using the CPU
    - Time spent waiting for external calls

Dynamic Reconfiguration

- Increasing use of virtualized servers
- Servers often available for launching on demand
- If the QoS requested by a client cannot be provided
  - Launch an additional server dynamically
- After a server failure
  - Reconfigure all sessions that had threads on the failed server
  - Some sessions might have to be canceled
- Extended resource allocation algorithm to support the above
- Algorithms can be easily enhanced to take into account
  - Costs associated with launching new servers
  - Revenue gained from new customer sessions
  - Costs incurred when breaking customer SLAs

CASE STUDY

Experimental Setup 1



S. Kounev, R. Nou, F. Julia, J. Torres

Workload Used

- Assume three services available
- Each service
  - executes CPU-intensive business logic
  - might call external third-party services
- Service workload model

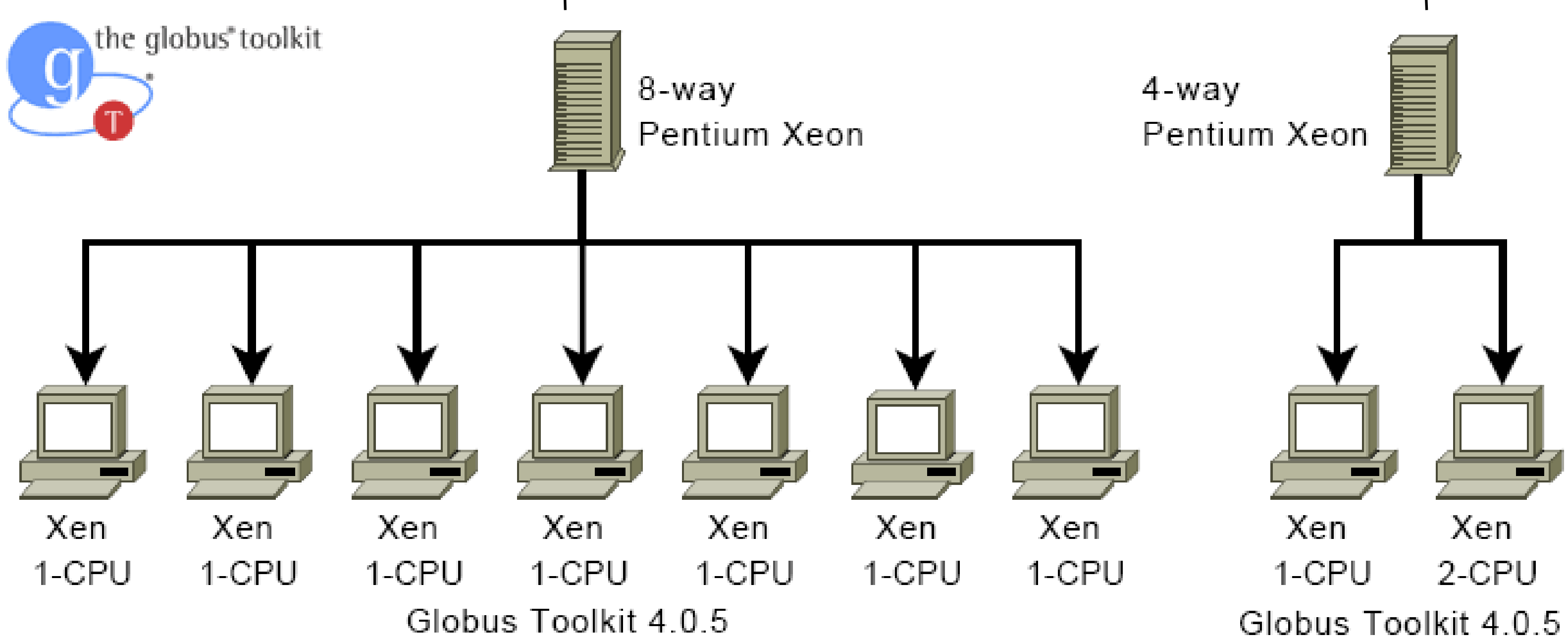
Service	Service 1	Service 2	Service 3
CPU resource demand on 2-way server	6.89	4.79	5.84
CPU resource demand on 4-way server	7.72	5.68	6.49
External Service Provider Time	2.00	3.00	0.00

- Workload model stored in service registry

Experimental Setup 2

8-way Pentium Xeon
2.60 GHz, 9 GB, 64 bit,
Xen hypervisor

4-way Pentium Xeon
3.16 GHz, 10 GB, 64 bit,
Xen hypervisor

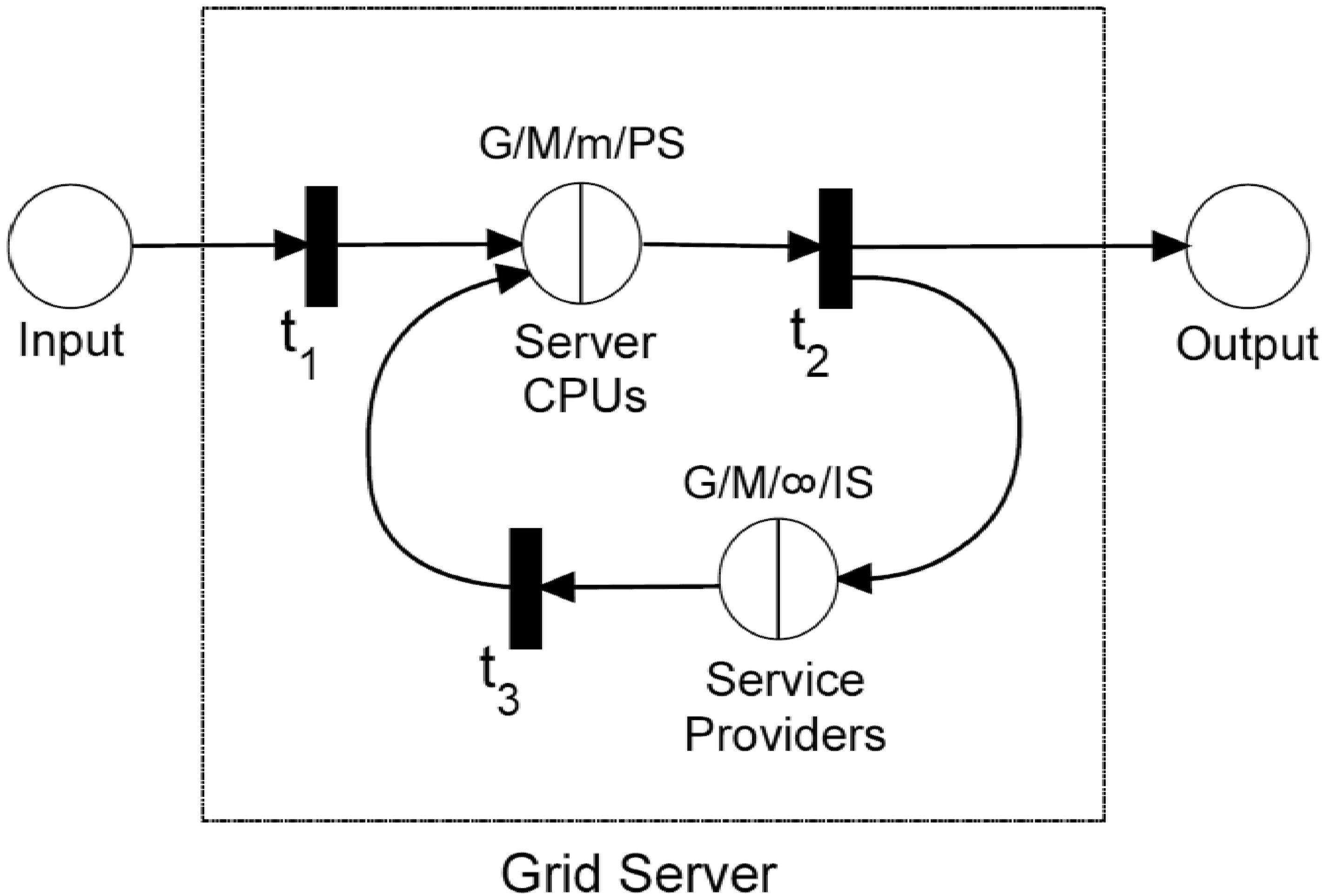


Workload Model

- One CPU on each server assigned to Domain-0
- Rest of the CPUs each assigned to one Grid server
- Service workload model

	Service 1	Service 2	Service 3
CPU service time on 1-way server (8-way machine)	7.48	5.28	6.05
CPU service time on 1-way server (4-way machine)	7.17	5.19	6.22
CPU service time on 2-way server (4-way machine)	7.04	5.07	6.04
External service provider time (sec)	2.00	3.00	*na*

Grid Server Model

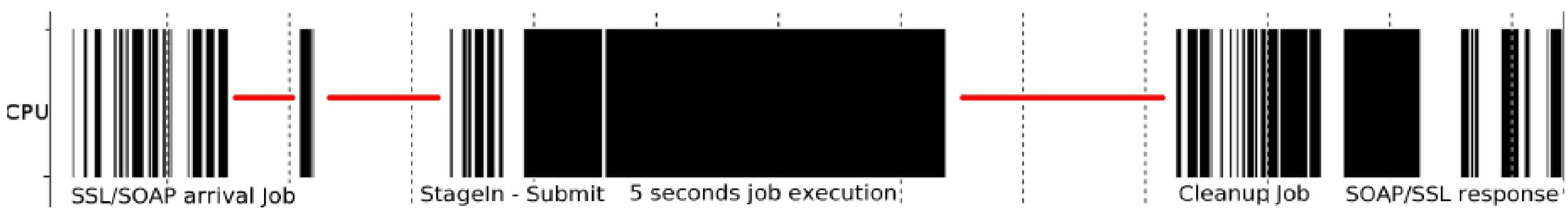


S. Kounev, R. Nou, F. Julia, J. Torres

Model Validation & Calibration

Services	No of threads allocated	Request interarrival time (sec)	Request response time (sec) measured	predicted	Error (%)
2	unlimited	4	11.43	10.47±0.033	8.3%
1—3	unlimited	8 / 8	13.66 / 12.91	12.21±0.019 / 11.17±0.031	11% / 13%
3	5	2.5	10.93	8.14±0.030	**25%**
1—3	2/2	8 / 8	18.15 / 9.79	15.58±0.23 / 7.8±0.05	14.1% / **20.3%**

- Model failed initial validation attempt
- Service execution trace (BSC-MF / Paraver)



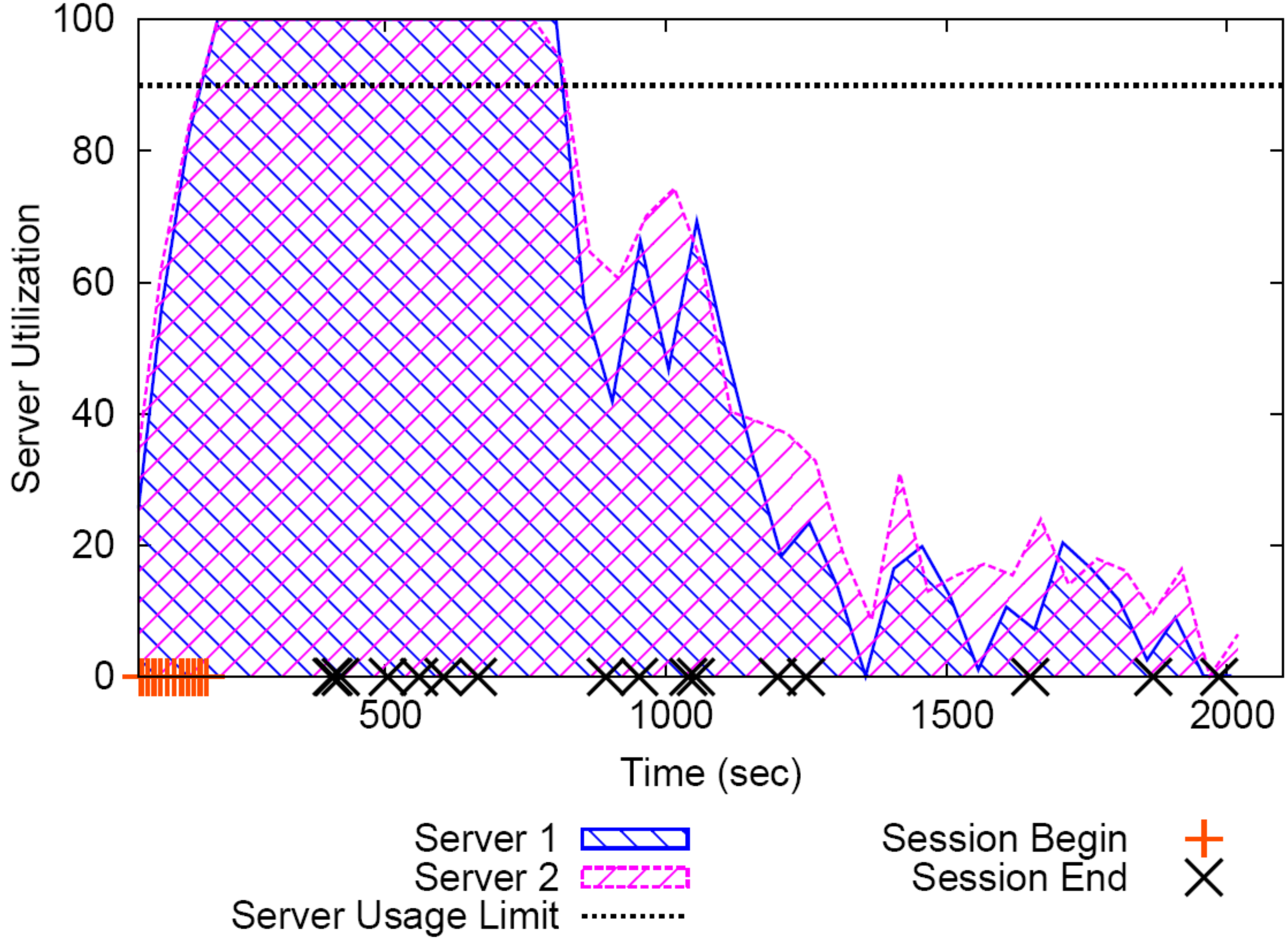
- Calibrated model by adding the 1 sec delay

S. Kounev, R. Nou, F. Julia, J. Torres

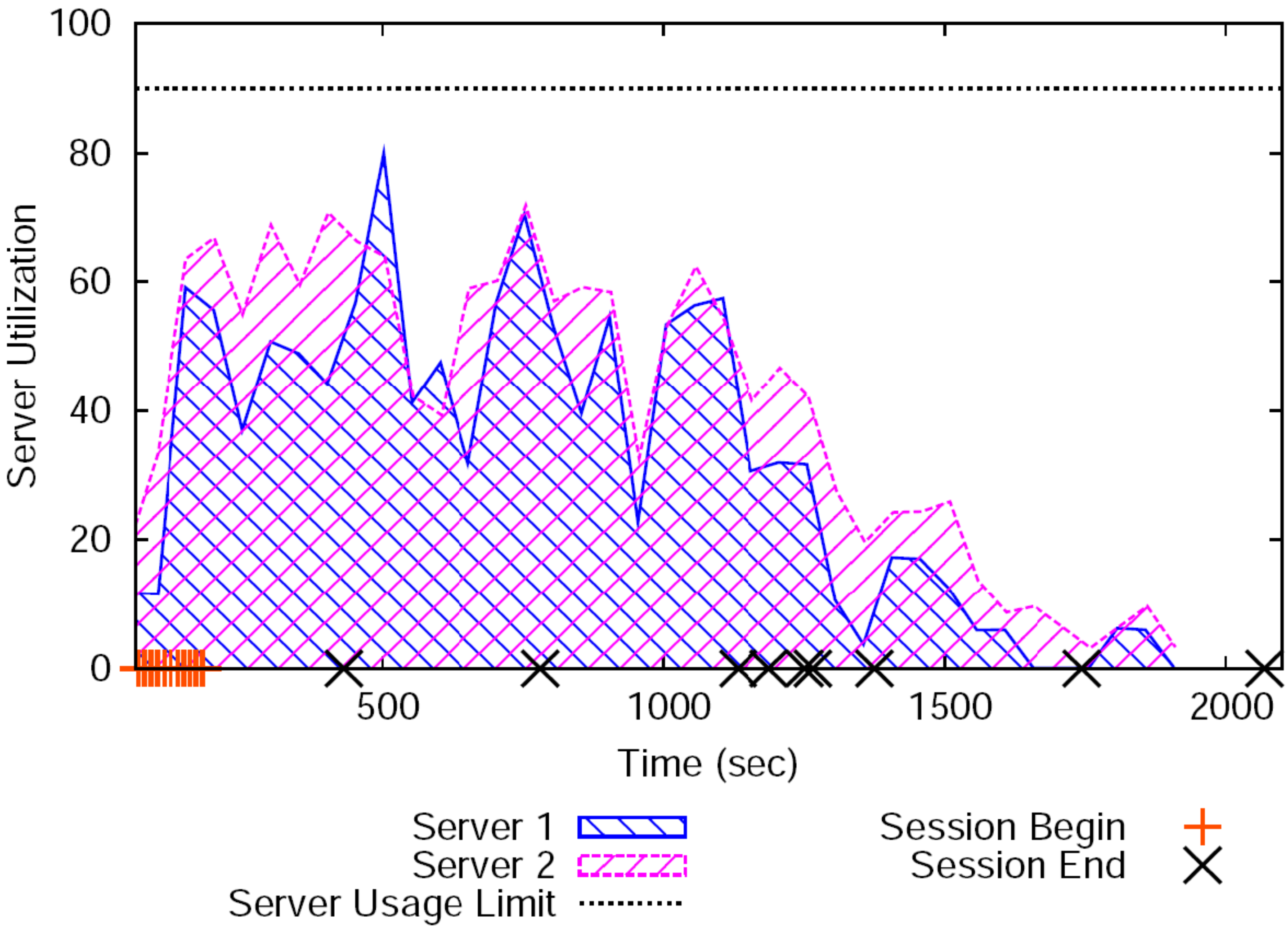
Scenario 1

- Used experimental setup 1
- 16 session requests
- Run until all sessions complete
- Each session has 20-120 service requests (avg. 65)
- SLAs between 16 and 30 sec
- 90% maximum server utilization constraint
- Will compare two configurations
  - Without QoS Control
    - Incoming requests simply load-balanced
  - With QoS Control
    - QoS-aware resource manager used

CPU Utilization – Without QoS Control

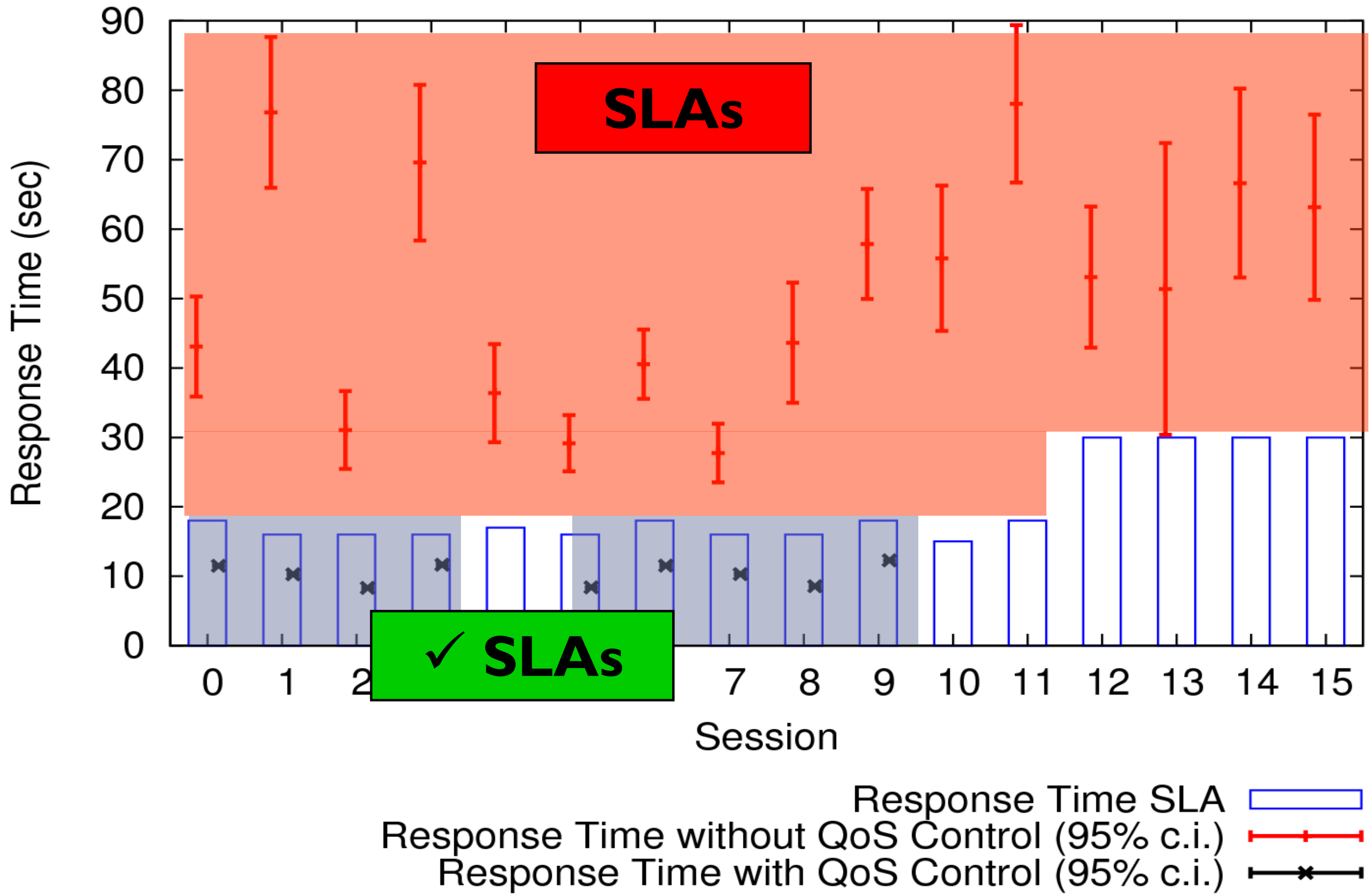


CPU Utilization – With QoS Control



S. Kounev, R. Nou, F. Julia, J. Torres

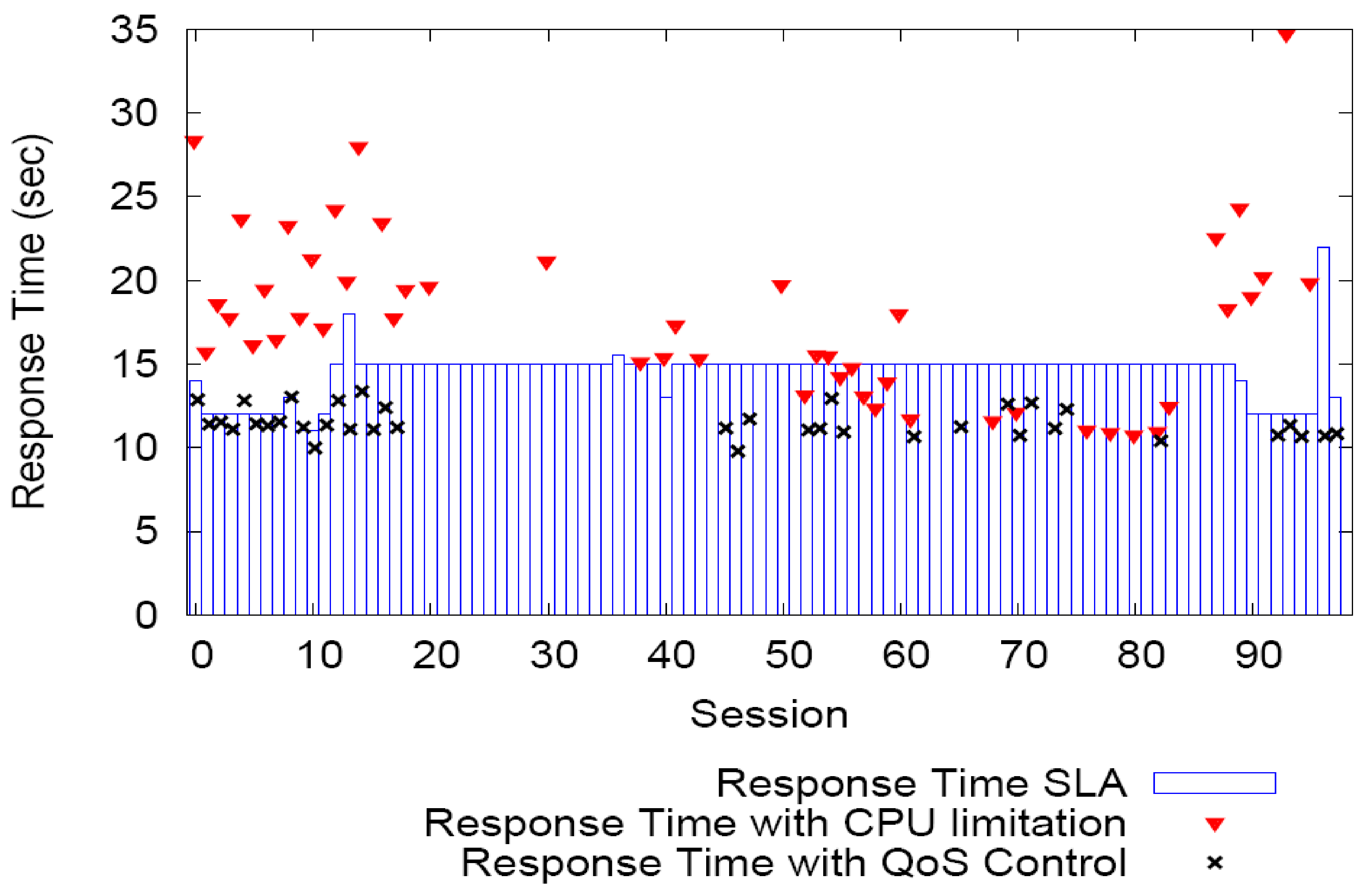
Average Session Response Times



Scenario 2

- Used experimental setup 1
- 99 session requests executed over period of 2 hours
- Run until all sessions complete
- Average session duration 18 minutes (92 requests)
- 90% maximum server utilization constraint
- Will compare two configurations
  - Without QoS Control
    - Incoming requests simply load-balanced
    - **Reject session requests when servers saturated**
  - With QoS Control
    - QoS-aware admission control enforced

Average Session Response Times

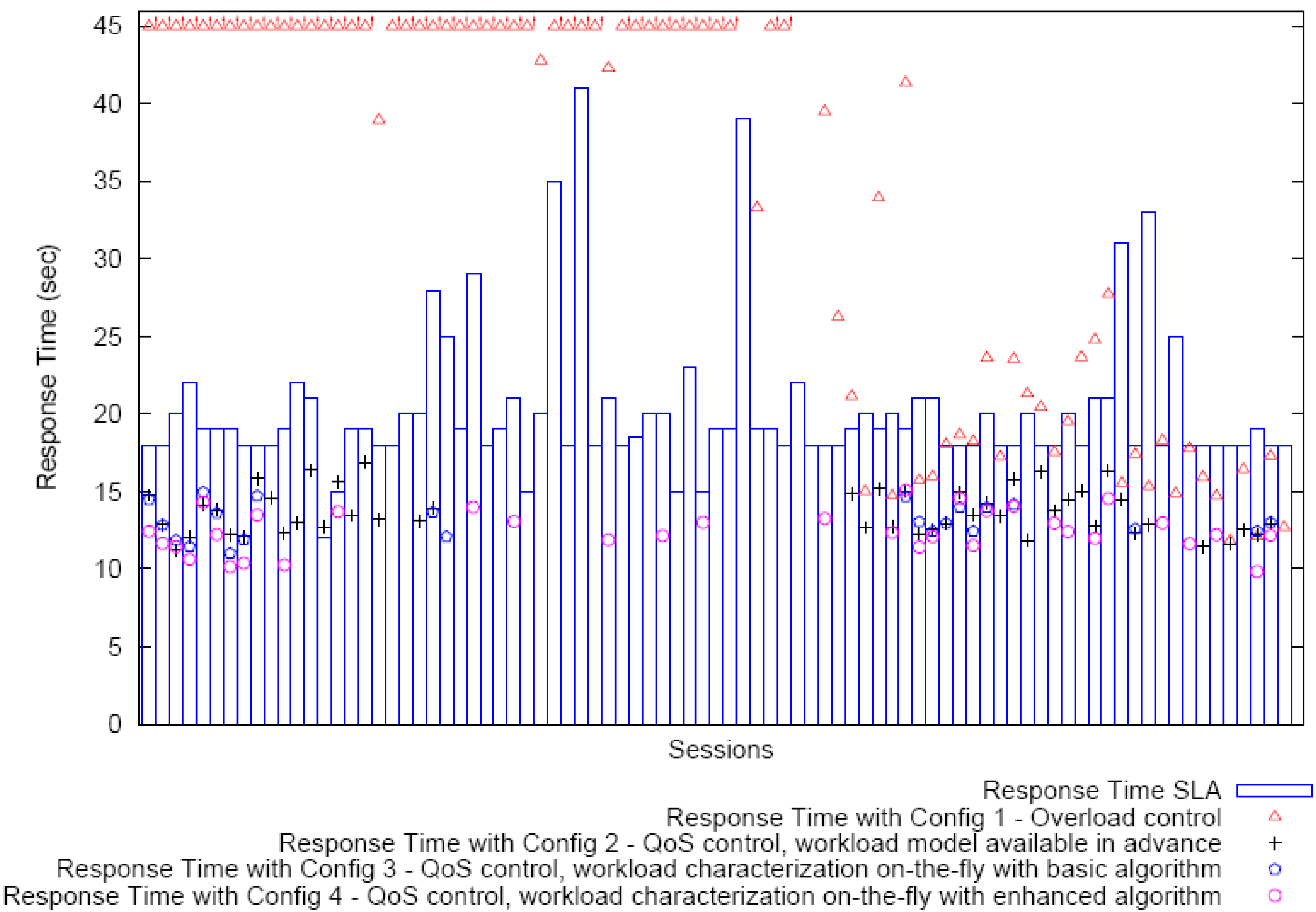


Scenario 3: Workload Characterization On-The-Fly

- Used experimental setup 2
- 85 sessions run over 2 hours
- Repeated for four configurations
  1. Overload control: reject new session requests when server utilization exceeds a specified threshold (70%)
  2. QoS control with workload model available in advance
  3. QoS control with workload char. on-the-fly (basic algorithm)
  4. QoS control with workload char. on-the-fly (enhanced algorithm)

S. Kounev, R. Nou, F. Julia, J. Torres

Scenario 3 Results



Scenario 3: Summary of SLA Compliance

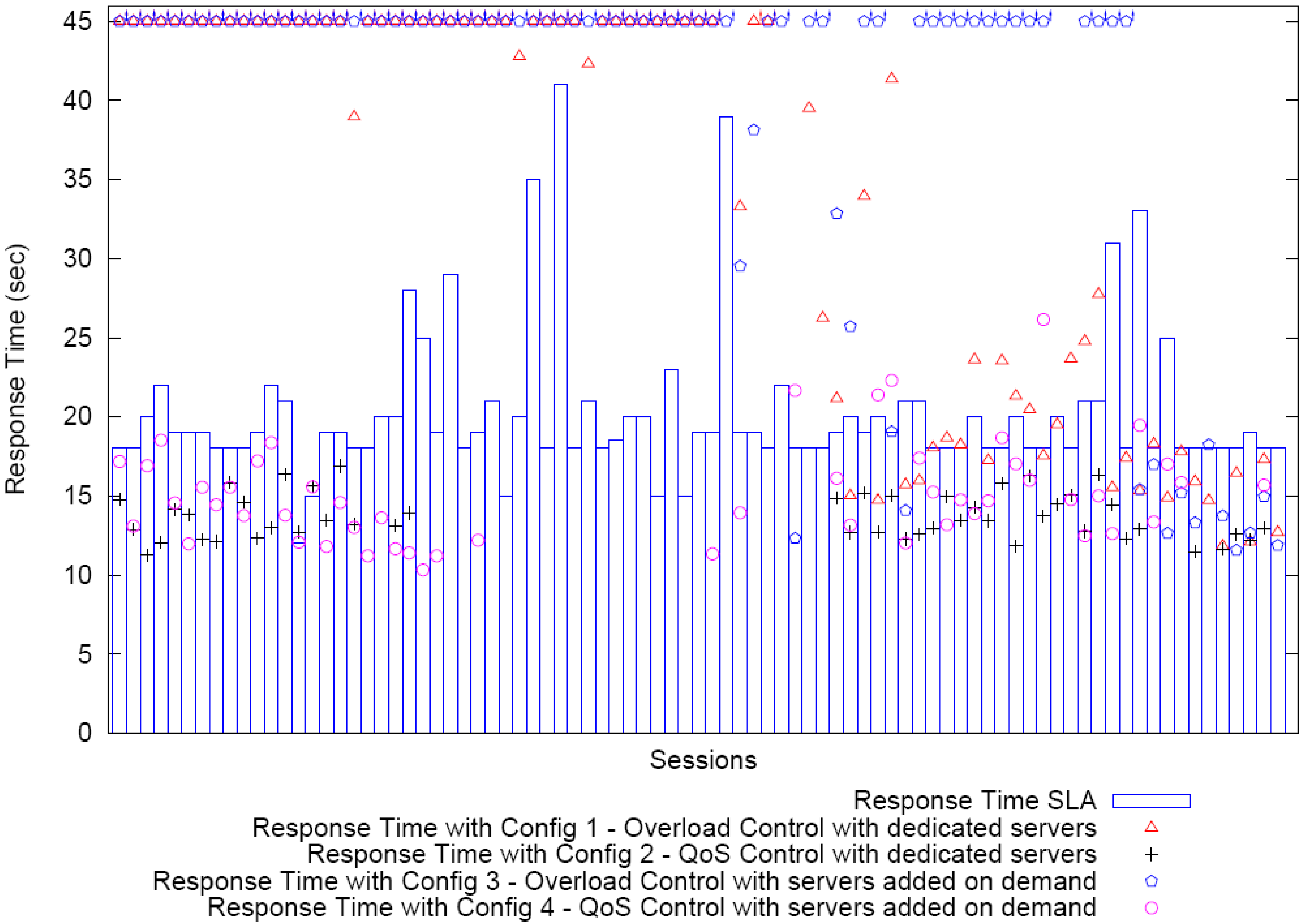
- Config 1: Basic overload control
  - 96% of sessions admitted, SLAs observed by only 22% of them
- Config 2: Workload characterization off-line
  - 54% of sessions accepted
- Config 3: Workload characterization on-the-fly (basic alg)
  - 26 % of sessions accepted
- Config 4: Workload characterization on-the-fly (enhanced alg)
  - Rejects only 14 sessions (16%) more compared to config 2

Configuration	SLA fulfilled	SLA violated	Sessions rejected
1	19	63	3
2	46	2	37
3	22	0	63
4	34	0	51

Scenario 4: Servers Added on Demand

- Used experimental setup 2
- 85 sessions run over 2 hours
- Repeated for four configurations
  1. Overload control with all nine servers available from the beginning.
  2. QoS control with all nine servers available from the beginning
  3. Overload control with one server available in the beginning and servers added on demand (when utilization exceeds 70%)
  4. QoS control with one server available in the beginning and servers added on demand

Scenario 4: Servers Added on Demand



Scenario 4: Summary of SLA Compliance

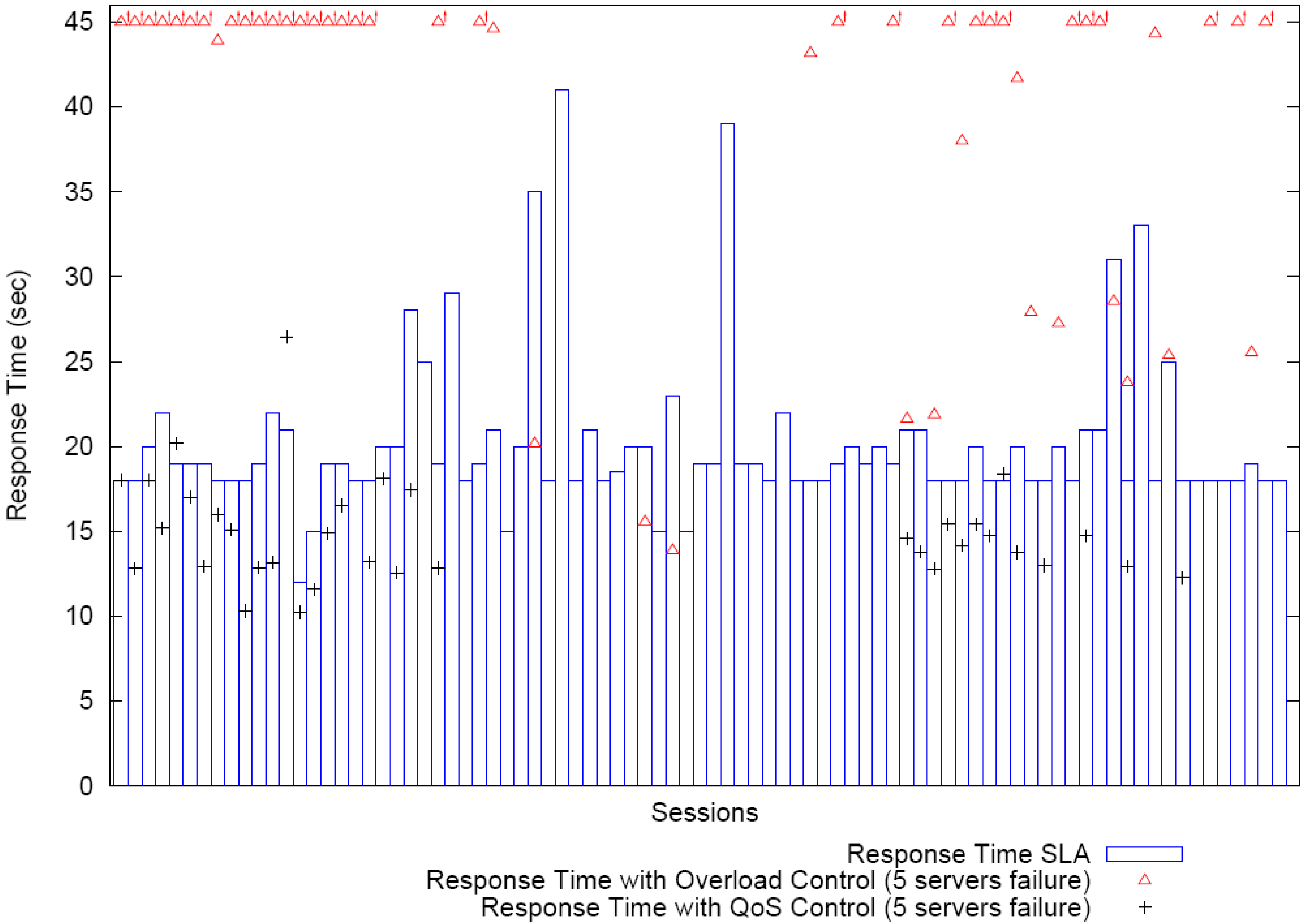
- Config 1: Overload control with all servers available
- Config 2: QoS control with all servers available
- Config 3: Overload control with one server available in the beginning and servers added on demand
- Config 4: QoS control with one server available in the beginning and servers added on demand

Configuration	SLA fulfilled	SLA violated	Sessions rejected
1	19	63	3
2	46	2	37
3	15	61	9
4	45	7	33

Scenario 5: Dynamic Reconfiguration

- Used experimental setup 2
- 85 sessions run over 2 hours
- Up to five server failures emulated during the run
- Points of server failures chosen randomly during the 2 hours
- Sessions reconfigured after each server failure

Scenario 5: Dynamic Reconfiguration



Scenario 5: Summary of SLA Compliance

Failures emulated	Without QoS Control			With QoS Control		
	SLA fulfilled	SLA violated	Sessions rejected	SLA fulfilled	SLA violated	Sessions rejected
1	14	62	9	37	1	47 (0)
2	16	57	12	39	3	43 (1)
3	10	58	17	40	3	42 (2)
4	3	56	26	38	1	46 (6)
5	4	45	36	31	4	50 (13)

Architecture Pros & Cons

- PROS
  - Service users decoupled from service providers
  - Fine-grained load-balancing
  - Possible to load-balance across heterogeneous servers
  - Without platform-specific load-balancing mechanisms
  - Dynamic reconfiguration possible
- CONS
  - Extra level of indirection
  - QoS manager overhead

QoS Predictor Overhead

- Ran simulation for fixed amount of time
- In scenarios 3, 4 and 5, the average time to reach decision was 15 sec with a max of 37 sec
- Several approaches to boost performance
  - Speed up model analysis
    - Parallelize simulation to utilize multi-core CPUs
    - Use alternative model types and solution techniques
  - Optimize resource allocation algorithm
    - Allocate resources bottom up instead of top down
    - Aggregate sessions of the same type
    - Cache analyzed configurations
    - Simulate proactively

CONCLUSIONS & FUTURE WORK

Conclusions & Future Work

- First to combine QoS Control with fine-grained load-balancing
- Balancing accuracy and speed is a major challenge
- Approach can be used in SOA environments
- On-going and future work
  - Optimize model analysis and resource allocation algorithm
  - Exploit multiple model types and analysis techniques
  - Integrate with design-oriented performance models (e.g., PCM)
  - Enhance to support hard QoS requirements
  - Integrate resource usage costs into the model

Further Reading

- Ramon Nou, Samuel Kounev, Ferran Julia and Jordi Torres. *Autonomic QoS control in enterprise Grid environments using online simulation.* To appear in Journal of Systems and Software, 2008.
- Samuel Kounev, Ramon Nou and Jordi Torres. *Autonomic QoS-Aware Resource Management in Grid Computing using Online Performance Models*. In Proceedings of the Second International Conference on Performance Evaluation Methodologies and Tools (VALUETOOLS-2007), Nantes, France, October 23-25, 2007.
- Ramon Nou, Samuel Kounev and Jordi Torres. *Building Online Performance Models of Grid Middleware with Fine-Grained Load-Balancing: A Globus Toolkit Case Study*. In Formal Methods and Stochastic Models for Performance Evaluation, Springer LNCS 4748/2007, Proceedings of the 4th European Performance Engineering Workshop (EPEW-2007), Berlin, Germany, September 27-28, 2007.
- Samuel Kounev. *Performance Modeling and Evaluation of Distributed Component-Based Systems using Queueing Petri Nets*. IEEE Transactions on Software Engineering, Vol. 32, No. 7, pp. 486-502, doi:10.1109/TSE.2006.69, July 2006.

Thanks