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

Samuel Kounev (skounев@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 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

- Grid QoS-Aware Resource Manager
  - QoS Broker
  - QoS Predictor
  - Client Registry
  - Service Registry

- Client Session Negotiation
- Service Request

- Grid Server 1
  - Services
  - CPUs

- Grid Server $N$
  - Services
  - CPUs
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](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
QPME Screenshot
QoS Predictor

QoS-Aware Resource Manager

Client $t_1$ Service Queue $t_2$ Grid Server $t_3$

Server 1
Thread Pool

Server $N$
Thread Pool

Grid Server $N$
Resource Allocation Algorithm

- New session request \((v, \lambda, \rho)\) 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

\[ V = \{ \nu_1, \nu_2, \ldots, \nu_n \} \] Services offered

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

\[ C = \{ c_1, c_2, \ldots, c_l \} \] Active client sessions where \( c_i = (\nu, \lambda, \rho) \)

For \( s \in S \):

- \( P(s) \) server capacity (e.g. \# CPUs)
- \( \bar{U}(s) \) maximum utilization constraint

\[ T \in [C \times S \rightarrow \mathbb{N}_0 \cup \{\infty\}] \] Thread allocation function
Resource Allocation Algorithm (2)

Define the following predicates

\[
P_X^T(c) \text{ for } c \in C \text{ as } X^T(c) = c[\lambda]
\]
\[
P_R^T(c) \text{ for } c \in C \text{ as } R^T(c) \leq c[\rho]
\]
\[
P_U^T(s) \text{ for } s \in S \text{ as } U^T(s) \leq \overline{U}(s)
\]

Configuration T is acceptable if and only if

\[
\forall c \in C : P_X^T(c) \land P_R^T(c) \land (\forall s \in S : P_U^T(s))
\]

Define also

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

\[
I^T(\nu, \epsilon) = \{ s \in S : \nu \in F(s) \land A^T(s) \geq \epsilon \}\]
Resource Allocation Algorithm (3)

1. \( C := C \cup \{\tilde{c}\} \)
2. for each \( s \in I^T(v, \epsilon) \) do \( T(\tilde{c}, s) := \infty \)
3. if \( (\exists c \in C : \neg P^T_X(c)) \) then reject \( \tilde{c} \)
4. while \( (\exists \hat{s} \in S : \neg P^T_U(\hat{s})) \) do
5. begin
6. \( T(\tilde{c}, \hat{s}) := 1 \)
7. while \( P^T_U(\hat{s}) \) do \( T(\tilde{c}, \hat{s}) := T(\tilde{c}, \hat{s}) + 1 \)
8. \( T(\tilde{c}, \hat{s}) := T(\tilde{c}, \hat{s}) - 1 \)
9. end
10. if \( (\exists c \in C \setminus \{\tilde{c}\} : \neg P^T_X(c) \lor \neg P^T_R(c)) \) then reject \( \tilde{c} \)
11. if \( (\neg P^T_X(\tilde{c}) \lor \neg P^T_R(\tilde{c})) \) then
12. send counter offer \( o = (v, X^T(\tilde{c}), R^T(\tilde{c})) \)
13. 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

2-way Pentium Xeon 2.4GHz
2 GB RAM

Client Emulator
Service Request Dispatcher

1Gb Ethernet Switch

2-way Pentium Xeon 2.4GHz
2 GB RAM
Globus Toolkit 4.0.3

4-way Pentium Xeon 1.4GHz
4 GB RAM
Globus Toolkit 4.0.3
Workload Used

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

<table>
<thead>
<tr>
<th>Service</th>
<th>Service 1</th>
<th>Service 2</th>
<th>Service 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU resource demand on 2-way server</td>
<td>6.89</td>
<td>4.79</td>
<td>5.84</td>
</tr>
<tr>
<td>CPU resource demand on 4-way server</td>
<td>7.72</td>
<td>5.68</td>
<td>6.49</td>
</tr>
<tr>
<td>External Service Provider Time</td>
<td>2.00</td>
<td>3.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>

- 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

<table>
<thead>
<tr>
<th></th>
<th>Service 1</th>
<th>Service 2</th>
<th>Service 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU service time on 1-way server</td>
<td>7.48</td>
<td>5.28</td>
<td>6.05</td>
</tr>
<tr>
<td>(8-way machine)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CPU service time on 1-way server</td>
<td>7.17</td>
<td>5.19</td>
<td>6.22</td>
</tr>
<tr>
<td>(4-way machine)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CPU service time on 2-way server</td>
<td>7.04</td>
<td>5.07</td>
<td>6.04</td>
</tr>
<tr>
<td>(4-way machine)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>External service provider time (sec)</td>
<td>2.00</td>
<td>3.00</td>
<td>na</td>
</tr>
</tbody>
</table>
Grid Server Model

Grid Server

Input \( t_1 \) \( \Rightarrow \) Server CPUs \( t_2 \) \( \Rightarrow \) Output

G/M/m/PS \( \Rightarrow \) G/M/\( \infty \)/IS

Service Providers

S. Kounev, R. Nou, F. Julia, J. Torres
Model Validation & Calibration

<table>
<thead>
<tr>
<th>Services</th>
<th>No of threads allocated</th>
<th>Request interarrival time (sec)</th>
<th>Request response time (sec)</th>
<th>Error (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>measured</td>
<td>predicted</td>
</tr>
<tr>
<td>2</td>
<td>unlimited</td>
<td>4</td>
<td>11.43</td>
<td>10.47±0.033</td>
</tr>
<tr>
<td>1—3</td>
<td>unlimited</td>
<td>8 / 8</td>
<td>13.66 / 12.91</td>
<td>12.21±0.019 / 11.17±0.031</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>2.5</td>
<td>10.93</td>
<td>8.14±0.030</td>
</tr>
<tr>
<td>1—3</td>
<td>2/2</td>
<td>8 / 8</td>
<td>18.15 / 9.79</td>
<td>15.58±0.23 / 7.8±0.05</td>
</tr>
</tbody>
</table>

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

- Calibrated model by adding the 1 sec delay
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

![Graph showing CPU utilization over time with server 1 and server 2 lines, session begin and end markers, and server usage limit.]

S. Kounen, R. Nou, F. Julia, J. Torres
Average Session Response Times

SLAs

Response Time SLA
Response Time without QoS Control (95% c.i.)
Response Time with QoS Control (95% c.i.)
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

Response Time SLA
Response Time with CPU limitation
Response Time with QoS Control
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)
Scenario 3 Results

Response Time SLA
- Response Time with Config 1 - Overload control
- Response Time with Config 2 - QoS control, workload model available in advance
- Response Time with Config 3 - QoS control, workload characterization on-the-fly with basic algorithm
- Response Time with Config 4 - QoS control, workload characterization on-the-fly with enhanced algorithm
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

<table>
<thead>
<tr>
<th>Configuration</th>
<th>SLA fulfilled</th>
<th>SLA violated</th>
<th>Sessions rejected</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>19</td>
<td>63</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>46</td>
<td>2</td>
<td>37</td>
</tr>
<tr>
<td>3</td>
<td>22</td>
<td>0</td>
<td>63</td>
</tr>
<tr>
<td>4</td>
<td>34</td>
<td>0</td>
<td>51</td>
</tr>
</tbody>
</table>
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

Response Time with Config 1 - Overload Control with dedicated servers
Response Time with Config 2 - QoS Control with dedicated servers
Response Time with Config 3 - Overload Control with servers added on demand
Response Time with Config 4 - QoS Control with 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

<table>
<thead>
<tr>
<th>Configuration</th>
<th>SLA fulfilled</th>
<th>SLA violated</th>
<th>Sessions rejected</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>19</td>
<td>63</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>46</td>
<td>2</td>
<td>37</td>
</tr>
<tr>
<td>3</td>
<td>15</td>
<td>61</td>
<td>9</td>
</tr>
<tr>
<td>4</td>
<td>45</td>
<td>7</td>
<td>33</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Failures emulated</th>
<th>Without QoS Control</th>
<th></th>
<th>With QoS Control</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SLA fulfilled</td>
<td>SLA violated</td>
<td>Sessions rejected</td>
<td>SLA fulfilled</td>
</tr>
<tr>
<td>1</td>
<td>14</td>
<td>62</td>
<td>9</td>
<td>37</td>
</tr>
<tr>
<td>2</td>
<td>16</td>
<td>57</td>
<td>12</td>
<td>39</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>58</td>
<td>17</td>
<td>40</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>56</td>
<td>26</td>
<td>38</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>45</td>
<td>36</td>
<td>31</td>
</tr>
</tbody>
</table>
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


Thanks