Self-Adapting Service Level in Java Enterprise Edition

Jeremy Philippe   Noel DePalma   Fabienne Boyer   Olivier Gruber

INRIA

December 3, 2009

Dynamic service-level adaptation

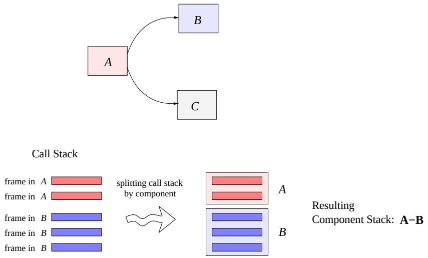
- Goal : improve quality of service in terms of better latency and peak throughput
- Alternative service levels (normal mode, degraded mode)
- Component-based system
  - Some components can dynamically updrade or degrade their level of service
  - Trading a lower service level for an improved quality of service of the system as a whole

Approach

- Online approach
  - Performance profile
    - Resource usage tracking
    - Gain estimation
  - Adaptation controler
    - Closed-control loop
    - Two thresholds for each resource
    - Regulation mode/Calibration mode
- Challenges
  - Stability of the regulation
  - Workload independent

Performance profile

- Identify the application's hot spots
  - Statistical sampling
- Account resource usage depending on the components involved in the threads
  - Abstract component stack
  - Account resource usage per component stack



Performance profile and adaptation gain

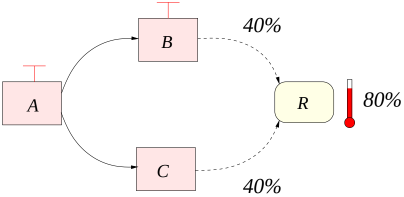
- Usage of resource R per component stacks
  - $U_R(S) = H_R(S).U_R$
- Gain of an adaptation δ on a resource R for a component stack S
  - First approximation
  - $G_R^{\delta}(S) = \frac{U_R^{+}(S)}{U_R^{-}(S)}$
- The gain is still sensitive to the actual execution rate of each component stack

Performance profile and adaptation gain

- $W_R(S)$: Execution rate of a component stack
- $C_R(S)$: Cost of a component stack
  - $C_R(S) = \frac{U_R(S)}{W_R(S)}$
  - More fine grain behavior
- Gain of an adaptation δ on a resource R for a component stack S
  - $G_R^{\delta}(S) = \frac{C_R^{+}(S)}{C_R^{-}(S)}$

Example of a performance profile

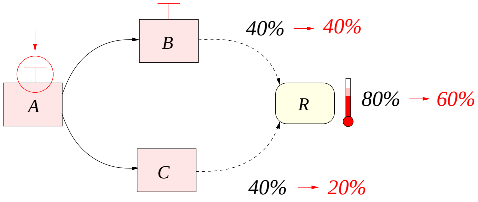
- $U_R = 80\%$
- $H_R(A-B) = H_R(A-C) = 50\%$



S	$H_R(S)$	$U_R(S)$	$W_R(S)$	$C_R(S)$	$G_A(S)$	$G_B(S)$
A-B	50%	40%	10 Hz	40 ms	? ?	? ?
A-C	50%	40%	5 Hz	80 ms	? ?	? ?

Adapting component A

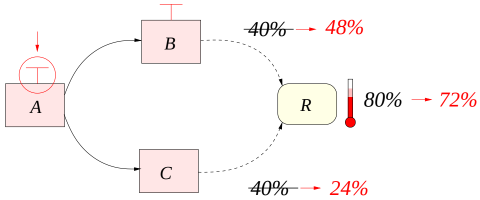
- Lowers the cost of the stack A-C on R
- No effect on A-B



S	$U_R(S)$	$W_R(S)$	$C_R(S)$	$G_A(S)$	$G_B(S)$
$A - B$	40% → 40%	10 Hz	40 ms	?? → 1.0	??
$A - C$	40% → 20%	5 Hz	80 ms → 40 ms	?? → 0.5	??

Adapting A, unstable workload

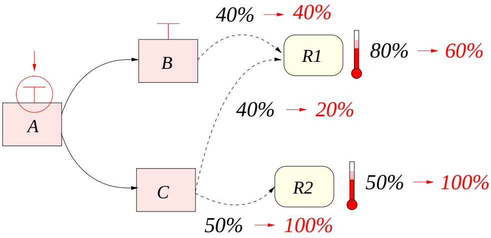
- Lowers $U_R(A-C)$, increases $U_R(A-B)$? ? ?
- if $G_R^A(S) = \frac{U_R^+(S)}{U_R^-(S)}$: wrong estimation
- if $G_R^A(S) = \frac{C_R^+(S)}{C_R^-(S)}$: more workload independent



S	$U_R(S)$	$W_R(S)$	$C_R(S)$	$G_A(S)$	$G_B(S)$
$A-B$	40% → 48%	10 Hz → 12 Hz	40 ms	1.0	? ?
$A-C$	40% → 24%	5 Hz → 6 Hz	80 ms → 40 ms	0.5	? ?

Multi-resource case

- Inhibiting the adaption on A
  - Lower the cost of stack A-C on R1
  - No effect on A-B
  - Stack A-C will overload R2



S	$U_R(S)$	$W_R(S)$	$C_R(S)$	$G_A(S)$	$G_B(S)$
$(A-B)_{R1}$	40% $\longrightarrow$ 40%	10 Hz	40 ms	1	0.75
$(A-C)_{R1}$	40% $\longrightarrow$ 20%	5 Hz	80 ms $\longrightarrow$ 40 ms	0.5	1.0
$(A-C)_{R2}$	50% $\longrightarrow$ 100%	5 Hz	80 ms $\longrightarrow$ 160 ms	2.0	1.0

Experimental set-up

- Standard Java EE
  - JOnAS v4.8, Java v1.5, MySQL v5.0, Fedora Core 6
- RUBiS Benchmark
  - Online auction application
  - Capacity ≈ 700 clients
  - The database is the bottleneck
- TPC-W Benchmark
  - Online bookstore application
  - Capacity ≈ 500 clients
  - The database is the bottleneck
- Component model
  - Limited to Servlets-EJBs
  - Database is a resource

Profiling

- Sampling Frequency = 10Hz
  - No significant overhead
- RUBiS (Read-write mix—write 20% matrix)

n°	S	$W(S)$	$U(S)$	$C(S)$
1	SearchItemsByCategory.Category.MySQL	6,60 Hz	36,9 %	55,8 ms
2	SearchItemsByRegion.Category.MySQL	2,22 Hz	17,5 %	78,4 ms
3	AboutMe.User.MySQL	11,4 Hz	0,33 %	0,49 ms
4	SearchItemsByRegion.Item.MySQL	16,7 Hz	0,48 %	0,29 ms

- TPC-W (Shopping mix—write 20% matrix)

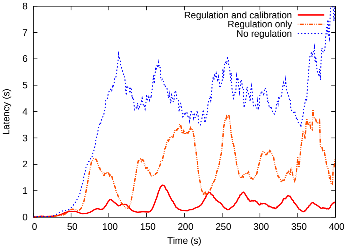
n°	S	$W(S)$	$U(S)$	$C(S)$
1	execute_search.author_search.MySQL	1,98 Hz	23,7 %	119,2 ms
2	best_sellers.MySQL	1,73 Hz	17,0 %	97,5 ms
3	execute_search.title_search.MySQL	2,03 Hz	11,2 %	54,9 ms
4	buy_confirm.MySQL	2,25 Hz	1,52 %	6,75 ms

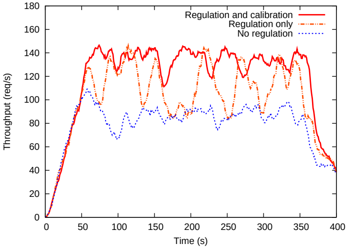
RUBiS performance

Middleware 2010

Philippe, DePalma, Boyer, Gruber

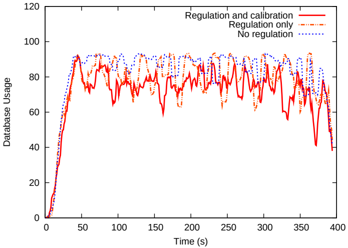
- Challenging target of resource usage (80%,90%)
- 0-400 sec : 1024 RUBiS clients (60 sec ramping up)
  - Poor latency and throughput without regulation
  - Unstable latency and throughput with regulation only



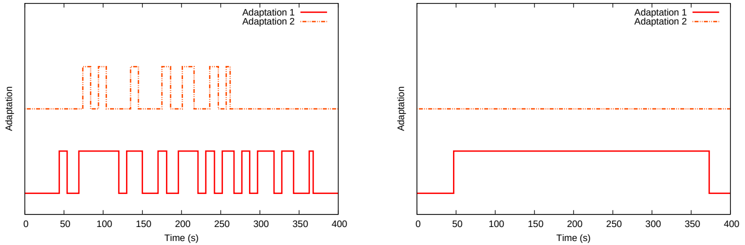


RUBiS performance

- The DB is thrashing without regulation
- The DB usage around 80% with regulation and calibration

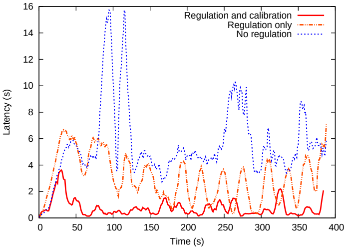


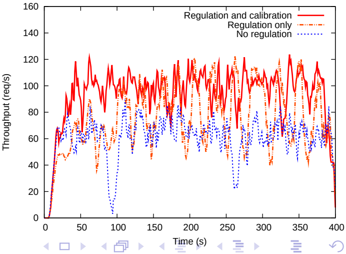
Stability (RUBiS)



TPC-W performance

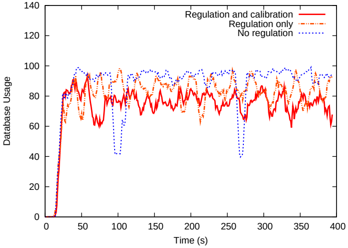
- Same thresholds
- 0-400 sec : 768 TPC-W clients (60 sec ramping up)
  - Peaks in latency and drops in throughput without regulation
  - Unstable latency and throughput with regulation only



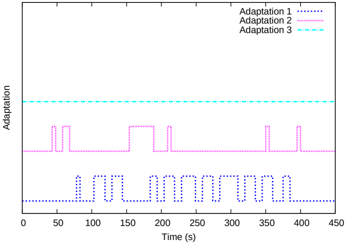


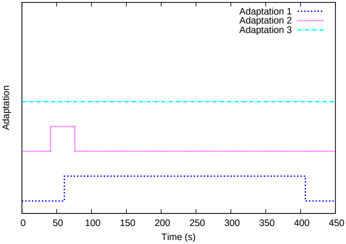
TPC-W performance

- The DB is thrashing without regulation
- The DB usage around 80% and drops avoided with regulation and calibration



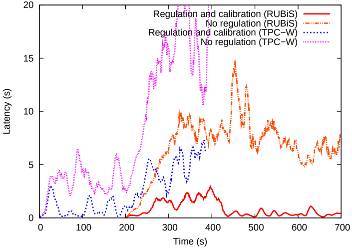
Stability (TPC-W)

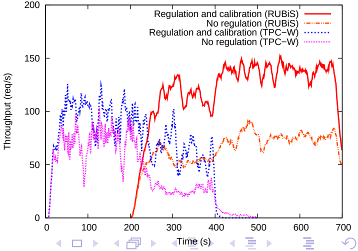




TPC-W/RUBiS mixed workload

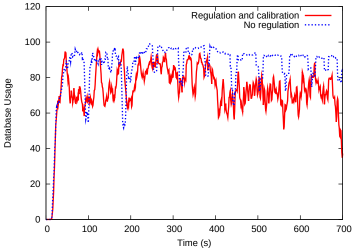
- 0-400 sec : 768 clients TPC-W (capacity ≈ 500 clients)
- 200-700 sec : 1024 clients RUBiS (capacity ≈ 700 clients)
  - Latency and throughput decline sharply without regulation
  - Preserve as much QoS as possible with regulation and calibration

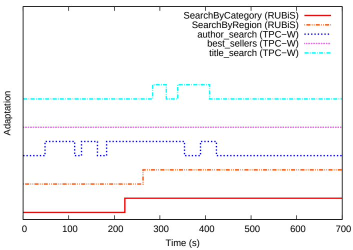




TPC-W/RUBiS mixed workload

- The DB is thrashing without regulation
- The DB usage around 80% with regulation and calibration





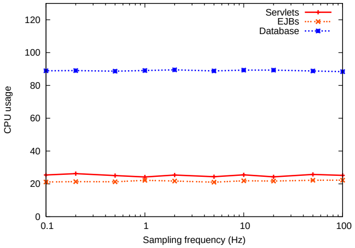
Conclusion

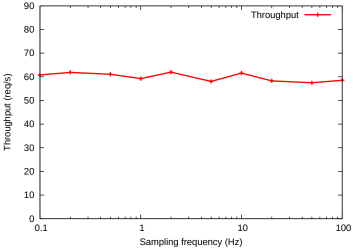
- Dynamic service level adaptation
  - Components can upgrade or degrade their level of service
- Approach
  - Closed control loop
  - Performance profile
- Advantages
  - Adaptations does not require to be characterized in advance
  - Low overhead
  - Good level of stability and accuracy

Future Work

- Experiments on other use cases
- Asynchronous execution
- Architecture reconfiguration
- Learning techniques

Overhead


Servlets
EJBs
Database
CPU usage
Sampling frequency (Hz)
0
20
40
60
80
100
120
0.1
1
10
100


Throughput
Throughput (req/s)
Sampling frequency (Hz)
0
10
20
30
40
50
60
70
80
90
0.1
1
10
100

Related Work

- Service-level adaptation
  - Multimedia[2], security[9]
  - Adaptations are well-known and can be characterized in advance
- Performance profiles
  - Resource containers [14]
  - Statistical regression techniques[16,17]
    - Non-stationary workload