Service-Level Awareness in Cloud Applications

Karl-Erik Årzén

Martina Maggio, Alessandro Papadopoulos, Manfred Dellkrantz, Jonas Dürango
Department of Automatic Control
Lund University, Sweden

+ Cristian Klein
Umeå University, Sweden

Contents

• Motivation
• Resource Management for Embedded Systems
  – The ACTORS Resource Manager
  – Game-Theory Resource Manager
• Cloud Resource Management
  – Brownout
  – Resource Management
  – Load Balancing
Background

- Application complexity and uncertainty increases for embedded system, smartphones, cloud infrastructures etc.
- Strict demands on both resource efficiency (e.g. power) and service performance.

Cloud infrastructures

- Energy consumption for ICT sector increases with 6.6% every year
- 2012 the total energy consumption was 930 TWh (= 100 nuclear power plants)
- In the future the majority of this will reside in the cloud

Resource optimization and control a crucial challenge!
Feedback Computing

• Computing as a technique to manage uncertainty, achieve performance and robustness and/or control power and temperature in computer and communication systems.

• Autonomic Computing

Data Centers

Desktop Systems

The Cloud

Embedded Systems

MPSoCs

Cloud Control

• Framework grant from Swedish Research Council in collaboration with the Cloud Research group at Umeå University.

• A control approach to a range of cloud management problems.
Service Level-Aware Applications

- Application knob
  - Decides the QoS achieved and the amount resources required
  - High SL → high QoS & high resource usage
  - Low SL → low QoS & low resource usage
- Discrete
  - “application modes”
- Continuous
  - e.g., sampling rate in a controller

Service level example

- SL: Resolution and/or frame rate of a video stream
- QoS and required CPU for encoding and decoding depends on the SL

SL1: 640x480
CPU: 30%

SL2: 800x600
CPU: 60%

SL3: 1024x768
CPU: 90%
Contents

• Motivation

• Resource Management for Embedded Systems
  – The ACTORS Resource Manager
  – Game-Theory Resource Manager

• Cloud Resource Management
  – Brownout
  – Resource Management
  – Load Balancing

ACTORS

• Adaptivity and Control of Resources in Embedded Systems

• EU FP7 STREP project
  – 2008-2011
  – Coordinated by Ericsson (Johan Eker)
  – Lund University, TU Kaiserslautern, Scuola Superiore Sant'Anna di Pisa, EPFL, AKAtech, Evidence

• Media applications (soft real-time) for smart phones
• Control applications
Overview

Applications
- Service-level aware
- Implemented with CAL dataflow language or legacy

Resource Manager
- C++ framework
- DBus IPC to applications
- Control groups API to scheduler

SCHED_EDF hierarchical scheduler
- Partitioned multi-core scheduler
- Hard CBS Reservations - one or several threads

Static Information

From applications to RM at registration:
- Service Level Table

<table>
<thead>
<tr>
<th>Service Level</th>
<th>QoS</th>
<th>BW Requirement</th>
<th>BW distribution</th>
<th>Timing Granularity</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>100</td>
<td>240</td>
<td>60-60-60-60</td>
<td>20 ms</td>
</tr>
<tr>
<td>1</td>
<td>75</td>
<td>180</td>
<td>45-45-45-45</td>
<td>20 ms</td>
</tr>
<tr>
<td>2</td>
<td>40</td>
<td>120</td>
<td>30-30-30-30</td>
<td>20 ms</td>
</tr>
</tbody>
</table>

- Thread IDs and how they should be grouped

From system administrator to RM at startup:
- Application Importance (weight)

<table>
<thead>
<tr>
<th>Appl</th>
<th>Importance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Appl 1</td>
<td>10</td>
</tr>
<tr>
<td>Appl 2</td>
<td>20</td>
</tr>
<tr>
<td>Appl 3</td>
<td>100</td>
</tr>
<tr>
<td>Default</td>
<td>10</td>
</tr>
</tbody>
</table>
Dynamic Inputs

- Happiness: boolean indicator of whether the QoS obtained corresponds to what could be expected at the current service level.
- Used Budget (Bandwidth): average used budget.
- Exhaustion Percentage: percentage of server periods in which the budget was exhausted.

Outputs

- Service Levels
- Reservation Parameters:
  - Budget
  - Period
  - Affinity
  - RM may migrate VPs
Resource Manager Functionality

- Assign service levels
  - When applications register or unregister
  - Formulated as a ILP problem
    - Importance as weight
    - glpk solver
- Mapping & bandwidth distribution
  - Map reservations to cores
  - Distribute the total BW to the reservations
    - Two Approaches:
      - Balance the load
      - Pack the VPs in as few cores as possible
    - Bin packing

\[
\begin{align*}
\text{max} & \sum_{i=1}^{m} w_i q_i x_i \\
\sum_{i=1}^{m} \alpha_i x_i & \leq C \\
\forall i, \sum x_i & = 1
\end{align*}
\]

Resource Manager Tasks

- Bandwidth adaptation
  - Adjust the servers bandwidth dynamically based on measured resource usage and obtained happiness

Changes what is meant by sufficiently close based on EP:

Changes the AB so that the UB lies sufficiently close:
MPEG-4 Video Decoding Example

- MPEG-4 SP decoder implemented in CAL
- Connected to an Axis network camera
- Service level changes results in commands from the decoder to the camera to reduce the frame rate and/or resolution

MPEG-4 Video Decoding Demonstrator

Application unhappy

More important application started. SL decreased

Application terminated. SL increased again
Drawbacks

Worked well, but
• ILP does not scale well
• Requires a lot of information from the applications
• More natural if the applications select their service levels and the resource manager adjust the VP size

→ The game-theory inspired approach

Contents

• Motivation
• Resource Management for Embedded Systems
  – The ACTORS Resource Manager
  – Game-Theory Resource Manager
• Cloud Resource Management
  – Brownout
  – Resource Management
  – Load Balancing
Towards decentralization

- The resource manager allocates resources and applications choose their service levels based on current performance.

Matching function

Measures how well the resources assigned to the application (vp size) matches the resource requirements (SL) of the applications.

- The matching is abundant
- The matching is perfect
- The matching is scarce
The matching is scarce
- increase vp or
- decrease SL

The matching is abundant
- decrease vp or
- increase SL

The matching is perfect

Application weights

- $\lambda_i \in [0..1]$
- $\lambda_i = 0$ means that only the application should adjust
- $\lambda_i = 1$ means that only the resource manager should adjust
Matching Function

- Application executes a series of jobs
  - Execution time $C_i = a_i s_i$
  - Response time for each job $R_i = \frac{c_i}{v_i} = \frac{a_i s_i}{v_i}$
  - Want this to be equal to the deadline $D_i$
  - Hence
    \[ f_i = \frac{D_i}{R_i} - 1 = \frac{D_i v_i}{a_i s_i} - 1 = \beta_i \frac{v_i}{s_i} - 1 \]
  - Can be measured from job start and stop times
  - Depends on service level and virtual processor speed

Resource Manager

At each step:
- Measure $f_i$ for all the applications
  - The applications report the start and stop of each job by writing to shared memory
  - RM reads from shared memory and calculates the response time
- Updates the virtual processors:
  \[ v_i(k + 1) = v_i + \varepsilon_{RM}(k) \left( -\lambda_i f_i(k) + \sum_{j=1}^{n} \lambda_j f_j(k) v_j(k) \right) \]
  \[ \varepsilon_{RM}(k) = \frac{1}{k + 1} \]
Service Level Adjustment

- Should set $s_i$ so that $f_i$ becomes 0
- Naive approach: $s_i(k + 1) = \beta_i v_i(k + 1)$
  - Assumes knowledge of $\beta_i$
- Instead estimate $\beta_i$
  \[ \beta_i = (1 + f_i(k)) \frac{s_i(k)}{v_i(k)} \]
- Gives
  \[ s_i(k + 1) = (1 + f_i(k)) \frac{v_i(k + 1)}{v_i(k)} s_i(k) \]
  - Continuous service levels
  - Robustified

Game Theory

- Provides convergence results and results about stationary values
- For example
  - A stationary point satisfies the following condition
    - The matching function is zero
    OR
    - The service level is the smallest possible AND the matching function is negative
Implementation

• Using SCHED_DEADLINE
  – New EDF+CBS scheduling policy in Linux
  – Developed by Evidence within ACTORS

Contents

• Motivation
• Resource Management for Embedded Systems
  – The ACTORS Resource Manager
  – Game-Theory Resource Manager
• Cloud Resource Management
  – Brownout
  – Resource Management
  – Load Balancing
From Embedded to the Cloud

- Apply the game-theoretic resource manager to cloud applications
- Focus on state-free, request-based applications

Problem: Unexpected Events

- 25% of end-users leave if load time > 4s*
- 1% reduced sale per 100ms load time*
- 20% reduced income if 0.5s longer load time**

* Amazon  ** Google
Standard Practice

- Overprovisioning
  - Economically impractical

Brownout

- We borrow from the concept of brownouts in power grids
- A brownout-compliant application can degrade user performance when needed to face unexpected conditions
- We assume that the reply to a request consists of a mandatory and an optional part
- During overload and/or lack of resources the percentage of requests that only receive the mandatory part can be increased
Brownout Examples

- E-commerce systems with recommendations
- Content adaptation in web server applications
- ......

Always Service Optional Part
Never Service Optional Part

With Brownout
Brownout Control Loop

- Measured variable: Response time (average or 95% percentile)
- Control signal: Dimmer value (percentage of requests for which also the optional parts are calculated)
- Setpoint: 2 seconds

Brownout Controllers

- Adaptive PI and/or PID controllers
- Adaptive deadbeat controller
- Feedforward + feedback controller
  - Assumes knowledge of average service time for mandatory and optional parts
Contents

• Motivation
• Resource Management for Embedded Systems
  – The ACTORS Resource Manager
  – Game-Theory Resource Manager
• Cloud Resource Management
  – Brownout
  – Resource Management
  – Load Balancing

Brownout Resource Management
Resource Management Details

- Applications sends value of matching function
  \[ f_i^k = 1 - t_i^k / \bar{r}_i \]
- Resource manager computes size of virtual machine
  \[ c_i^{k+1} = c_i^k - \epsilon_{rm} \left( f_i^k - c_i^k \cdot \sum_p f_p^k \right) \]
- Proven to be fair and converge using game theory

Evaluation

- Applications
  - RUBiS: eBay-like prototype auction website
    - Added a recommender
  - RUBBoS: Slashdot-like bulletin board website
    - Added a recommender
    - Marked comments as optional
  - Effort in lines of code:

<table>
<thead>
<tr>
<th>Modification</th>
<th>RUBiS</th>
<th>RUBBoS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recommender</td>
<td>37</td>
<td>22</td>
</tr>
<tr>
<td>Dimmer</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>Reporting response time to controller</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Controller</td>
<td>120</td>
<td>120</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>165</strong></td>
<td><strong>153</strong></td>
</tr>
</tbody>
</table>
Evaluation

- Hardware
  - AMD Opteron 6272 processor
  - 2 processors, 8 cores
- Hypervisor
  - Xen
- Client load generator
  - Custom built – httpmon
  - Closed loop model
    - Clients with think time

Experimental Results

- RUBiS flash crowds

![Graphs showing experimental results](image-url)
Experimental Results

Contents

• Motivation
• Resource Management for Embedded Systems
  – The ACTORS Resource Manager
  – Game-Theory Resource Manager
• Cloud Resource Management
  – Brownout
  – Resource Management
  – Load Balancing
Load Balancing

• Multiple replicas
  – Support scaling
  – Robust towards faults
• Load balancing

Existing Load Balancers

• **Dynamic** load balancers often measure response times and decide based on that
  – FRF, FRF-EWMA, 2RC, Predictive ...
• Does not work well in the presence of brownout control
  – the replica controllers keep the response times close to the setpoint
Existing Load Balancers

- **Static** load balancers do not reflect the current situation
  - Random, RR, Weighted-RR ...
- However,
  - Shortest Queue First (**SQF**) works quite good in the brownout case
  - **SQF** monitors the queues of the replicas and send requests to the replica that has the least amount of requests to serve

Can we do better?

- Ideally we would like to serve as much **optional content** as possible, without sacrificing response times

\[
\max_{w_i} J = \sum_i w_i \theta_i
\]

- **Replica weights**
- **Dimmer values**
Piggyback Dimmer Values

- Piggyback the dimmer values to the replies going back to the load balancer

Solution #1: VPBH

- Variational principle-based heuristic
- Based on the difference between the current dimmers and the past dimmers
- If the percentage of optional content served is increasing, the replica is assumed to be less loaded, and more traffic can be sent to it.
- When the optional content decreases, the replica will receive less traffic, to decrease its load and allow it to increase its dimmer value

\[
\tilde{w}_i(k+1) = w_i(k) \cdot [1 + \gamma_p \Delta \theta_i(k) + \gamma_i \theta_i(k)],
\]

\[
w_i(k) = \frac{\tilde{w}_i(k)}{\sum_i \tilde{w}_i(k)}.
\]
Solution #2: EPBH

- Equality principle-based heuristic
- Based on the fact that when the load is correctly balanced the dimmers should be equal

\[ \hat{w}_i(k+1) = w_i(k) + \gamma e_i(k) \]

\[ e_i(k) = \left[ \theta_i(k) - \frac{\sum_j \theta_j(k)}{n} \right] \]

\[ w_i(k) = \frac{\hat{w}_i(k)}{\sum_i \hat{w}_i(k)}. \]

Solution #3: OBLB

- Optimization based load-balancing
- Based on knowledge about the system - the service rate for requests with and without optional content
- The replica is an M/G/1 queue with PS discipline, probability density function for service times and effective service rate:

\[ f_{S_i}(t) = (1 - \theta_i) \cdot \mu_i \cdot e^{-\mu_i \cdot t} + \theta_i \cdot M_i \cdot e^{-M_i \cdot t}, \]

\[ \mu_i^* = \left[ E[S_i] \right]^{-1} = \left[ \frac{1 - \theta_i}{\mu_i} + \frac{\theta_i}{M_i} \right]^{-1}. \]
Solution #3: OBLB

- We can compute where the dimmers are supposed to converge

\[
\theta_i^* = \frac{M_i \cdot (\mu_i t_i^* - 1 - \lambda w_i t_i^*)}{(1 + \lambda w_i t_i^*) \cdot (\mu_i - M_i)} = \frac{A_i - B_i w_i}{C_i + D_i w_i}.
\]

Setpoint for average response time

Solution #3: OBLB

- We can solve the constrained optimization problem (concave cost function and concave constraints)

\[
\begin{align*}
\max_{w_i} & \quad J = \sum_i w_i \theta_i = \sum_i w_i \frac{A_i - B_i w_i}{C_i + D_i w_i} \\
\text{s.t.} & \quad \sum_i w_i = 1 \\
& \quad \max(0, \frac{C_i - A_i}{B_i + D_i}) \leq w_i \leq \frac{A_i}{B_i}
\end{align*}
\]
Evaluation

- We evaluate:
  - The percentage of optional content served $\%_{oc}$
  - Response times: e.g., average response time
  - User perceived stability (standard deviation of response times)

Experiments: client behavior

- Client behavior:
  - Four replicas (one fast, one medium, two slow)
  - 50 clients at time 0, 10 disconnected at time 200, 25 more at time 400, 25 more at time 600, 40 disconnected at time 800
  - Closed loop client model, with think time of 1s
  - Load balancer executed every 1s, replica controllers executed every 0.2s
Experiments: client behavior

\[ w \quad \theta \]

- **EPPH**: 81.9%
- **VPBH**: 78.9%
- **OBLB**: 76.2%
- **SOF**: 67.0%

---

Experiments: client behavior

<table>
<thead>
<tr>
<th>Method</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>EPPH</td>
<td>81.9%</td>
</tr>
<tr>
<td>VPBH</td>
<td>78.9%</td>
</tr>
<tr>
<td>OBLB</td>
<td>76.2%</td>
</tr>
<tr>
<td>SOF</td>
<td>67.0%</td>
</tr>
<tr>
<td>RFR</td>
<td>87.9%</td>
</tr>
<tr>
<td>RR</td>
<td>41.2%</td>
</tr>
<tr>
<td>Predictive</td>
<td>36.9%</td>
</tr>
<tr>
<td>Weighted RR</td>
<td>21.9%</td>
</tr>
</tbody>
</table>
Experiment - Infrastructure

Experiments: infrastructure

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>$%_{OC}$</th>
<th>$\mu$</th>
<th>$\sigma$</th>
</tr>
</thead>
<tbody>
<tr>
<td>OBLB</td>
<td>91.4%</td>
<td>0.71</td>
<td>0.89</td>
</tr>
<tr>
<td>EPBH</td>
<td>89.5%</td>
<td>1.06</td>
<td>1.95</td>
</tr>
<tr>
<td>VPBH</td>
<td>87.7%</td>
<td>1.02</td>
<td>1.90</td>
</tr>
<tr>
<td>SQF</td>
<td>83.3%</td>
<td><strong>0.55</strong></td>
<td><strong>0.40</strong></td>
</tr>
<tr>
<td>RR</td>
<td>75.5%</td>
<td>1.11</td>
<td>2.42</td>
</tr>
<tr>
<td>Random</td>
<td>72.9%</td>
<td>0.86</td>
<td>2.23</td>
</tr>
<tr>
<td>2RC</td>
<td>72.2%</td>
<td>0.74</td>
<td>1.64</td>
</tr>
<tr>
<td>FRF</td>
<td>70.4%</td>
<td>1.27</td>
<td>2.03</td>
</tr>
<tr>
<td>Weighted-RR</td>
<td>60.7%</td>
<td>0.90</td>
<td>2.97</td>
</tr>
<tr>
<td>FRF-EWMA</td>
<td>51.4%</td>
<td>1.44</td>
<td>3.41</td>
</tr>
<tr>
<td>Predictive</td>
<td>47.4%</td>
<td>1.66</td>
<td>3.48</td>
</tr>
</tbody>
</table>
More to gain

- SQF event-based
- The brownout aware policies are periodical
- Better performance with event-based implementation

Conclusions

- We introduced service level awareness to the cloud, through the concept of brownouts
- We developed a brownout-compliant resource manager, based on a decentralized resource management scheme proposed for embedded systems
- Existing load-balancing algorithms are not suited for brownout-compliant applications, except for SQF
- We developed some brownout-aware load-balancers
Future work

- Develop better (less heuristic based) load balancers for brownout compliant cloud applications
- Integrate resource management, autoscaling and load balancing
- Take into account the network latency with differentiated setpoints for clients with higher/lower latency

Further Information

- ACTORS Resource Manager
- Game-Theoretical Resource Manager
Further Information

Cloud Brownout

- Cristian Klein, Martina Maggio, Karl-Erik Årzén, Francisco Hernández-Rodríguez: "Brownout: Building more Robust Cloud Applications", 36th International Conference on Software Engineering (ICSE), Hyderabad, India, 2014
- Martina Maggio, Cristian Klein, Karl-Erik Årzén "Control strategies for predictable brownouts in cloud computing", IFAC World Congress, Cape Town, South Africa, August 2014
- Jonas Dürango, Manfred Dellkrantz, Martina Maggio, Cristian Klein, Alessandro Vittorio Papadopoulos, Francisco Hernández-Rodríguez, Erik Elmroth, Karl-Erik Årzén, “Control-theoretical load-balancing for cloud applications with brownout”, In submission to CDC 2014

Test #1

- Convergence of virtual processors
- Four applications
  \[ \lambda_1 = 0.1, \lambda_2 = 0.3, \lambda_3 = 0.2 \text{ and } \lambda_4 = 0.5 \]
Test #2