Development and validation of an optimal self-adaptive SLA-oriented resource allocation algorithm for cloud computing

Universidade Federal do Rio de Janeiro
Escola Politécnica

Jonathan FERREIRA PASSONI

jonathan.ferreirapassoni@gmail.com

March, 2021

Outline

Introduction

Related work

The solution conceived by DELL-EMC

Adaptive Control Strategies for a single container

Adaptive Control Strategies

Testing

Optimal Adaptive Control Strategy for multiple containers

System Structural Architecture

The OACS design

Testing

Conclusion and Perspectives

Introduction

Current trends in Cloud Computing

Main idea: computational resources seen as an *on-demand* service

- Quality of Service ensured by Service Level Agreements (SLA), based on performance metrics;
- Need of a Resource Management System to deal with multiple applications sharing the same infrastructure.

Challenges:

- Power consumption in cloud data centers;
- Software development : widespread implementation of *micro services* - the use of **containers**;
- Diversity of workload profiles;
- Software systems usually not described by mathematical models.

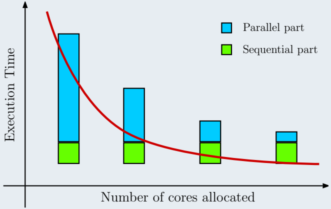
Introduction

The Resource Allocation Problem

Main goal

Determine the optimal amount of resources to satisfy a SLA defined as an upper bound in execution time.

Software Systems



Iterative Workload

- identical small jobs that run in succession;
- Examples: ML training and video encoding processes.

Control Theory

- model based technique;
- provides formal guarantees for an adequate performance.

The solution of this work

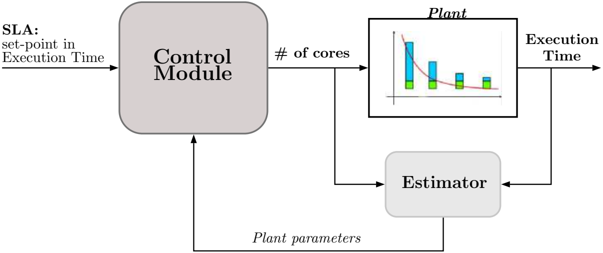
Develop and validate an algorithm applying **control techniques** to provide the **optimal** resource allocation in a **self-adaptive** approach, specifically for iterative workloads.

Introduction

The Resource Allocation Problem

The approach considered in this work

- Every application is seen as a plant to be controlled;

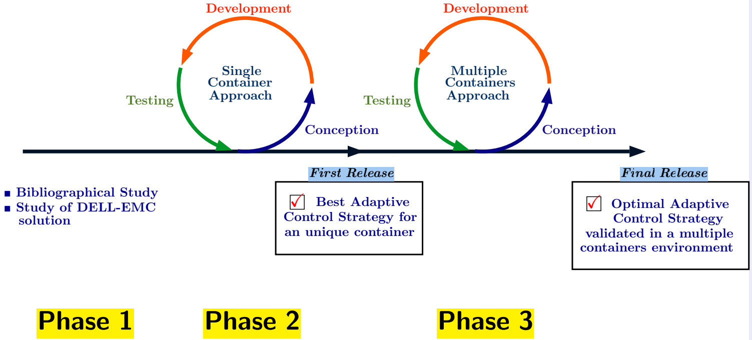


- **Multiple applications** seen as different plants (sub-systems) to be managed by a centralized control module.

Introduction
Organization of the work in this project

- Define a model for the workload characteristics
- Design Adaptive Control Strategies
- Study the performance of all strategies under different operating conditions

- Conceive a new Structural Architecture
- Design the global OACS
- Validate the deployment of multiple containers using the OACS under different scenarios



Related work

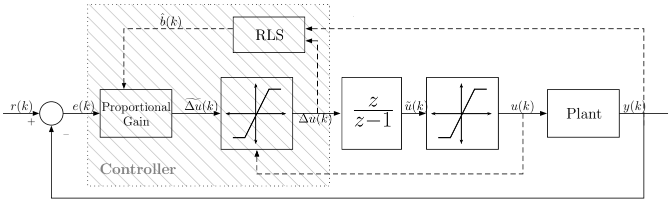
The self-adaptive concept: a parametric adaptation *(in most cases)*

Current solutions developed:

- In Research Community:
  - Model: gray-box, discrete non-linear time-invariant;
  - Parameters Identification: using an Initial Phase dedicated to it. Most of them using a Kalman Filter;
  - Controller Types: PID, MPC, LLC;
  - Presence of *change point detection mechanisms* to deal with some model inconsistencies.
- In Industry: **No self-adaptation implemented**

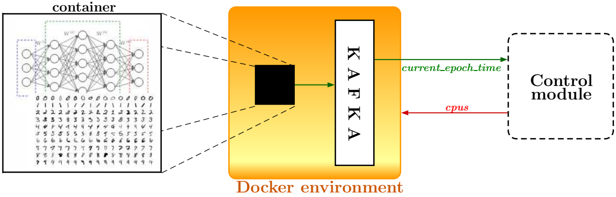
- Use of control techniques, but the client must specify details related to the workload;
- Client defines boundary values for each resource type;

The solution conceived by DELL-EMC
Control System Block Diagram



- Plant: relationship between **# of cores**$(u(t))$ and **Execution Time**$(y(t))$
- Model: $x(k+1) = x(k) + \beta\Delta u(k)$
  $y(k) = x(k)$
- Control input: $\Delta u(k)$
- Online estimation using **R**ecursive **L**east **S**quares
  - Regression model: $x(k+1) - x(k) = \hat{\beta}\Delta u(k)$
  - Estimated parameter is applied only after Identification Phase.
- Control law: $\Delta cpus(k+1) = \frac{1}{\hat{\beta}(k)} e(k)$

The solution conceived by DELL-EMC
System Structural Architecture and Performance Analysis



Testing Campaign
(Workload profile: ML training process - MNIST data-base)

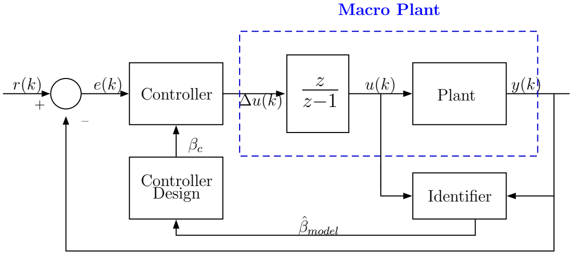
- SLA: constant set-point in all epochs.

Performance in steady state	**Set-point values**			
	12.5s	**25.0s**	**50.0s**	**100.s**
Set-point tracking error achieved?	YES	YES	NO	NO
Mean value [s]	12.42	25.06	51.39	106.41
Variance [s^2]	0.04	1.32	48.65	351.09
$\hat{\beta}$	constant	constant	variant	variant
RLS estimation error	converges to zero	converges to zero	variant	variant

ACS for a single container

Adaptive Control Strategies

Control System Block Diagram



- **Plant model:** 2 candidate functions
  - Hyperbolic function: $TTF(k) = \alpha + \beta \frac{1}{cpus(k)+\gamma}$
  - Exponential function: $TTF(k) = \alpha e^{\beta cpus(k)} + \gamma$, $\beta < 0$

  where $TTF(k)$ is the time execution.
- **Estimation:** linear parameterization; γ considered as a constant : γ_0
  - **Without** an initial Identification Phase;
  - Additional mechanism to better manage high estimation errors.
- **Controller Type:** Dead-beat

ACS for a single container

Adaptive Control Strategies

Workload characteristics as a hyperbolic function

Control Strategy 1 (using Taylor expansion)

- Dynamics: $TTF(k+1) = TTF(k) - \frac{\beta_c \Delta cpus}{[cpus(k) + \gamma_0]^2}$
- Dead-beat control law: $\Delta cpus = -\frac{[cpus(k) + \gamma_0]^2}{\hat{\beta}_c}(sp\_error)^{i}$
- Relationship between $\hat{\beta}_c$ and $\hat{\beta}$: $\hat{\beta}_c = \frac{[cpus(k) + \gamma_0]}{[\hat{cpus}(k+1) + \gamma_0]}\hat{\beta}$

Control Strategy 2 (using the exact hyperbolic function)

$$TTF(k) = \alpha + \frac{\beta}{cpus(k) + \gamma_0} \Rightarrow TTF(k+1) = \alpha + \frac{\beta}{(cpus(k) + \Delta cpus) + \gamma_0}$$

- Dynamics: $TTF(k+1) = TTF(k) - \beta \frac{\Delta cpus(k)}{(cpus(k) + \gamma_0)^2 + \Delta cpus(k)[cpus(k) + \gamma_0]}$
- Dead-beat control law: $\Delta cpus(k) = -\frac{(cpus(k) + \gamma_0)^2 (sp\_error)^{i}}{(cpus(k) + \gamma_0)(sp\_error)^{i} + \hat{\beta}}$

[i] $sp\_error = set\_point - TTF(k)$

ACS for a single container

Adaptive Control Strategies

Workload characteristics as an exponential function

Control Strategy 3 (using Taylor expansion)

- Dynamics: $TTF(k+1) = TTF(k) - \beta_c \alpha e^{-\beta cpus(k)} \Delta cpus$
- Dead-beat control law: $\Delta cpus = -\frac{1}{\hat{\beta}_c} \frac{set\_point - TTF(k)}{TTF(k) - \gamma_0}$
- Relationship between $\hat{\beta}_c$ and $\hat{\beta}$: $\beta_c = \frac{e^{\hat{\beta}(cpus(k) - \hat{cpus}(k+1))} - 1}{cpus(k) - \hat{cpus}(k+1)}$

Control Strategy 4 (using the exact exponential function)

$$ln[TTF(k) - \gamma_0] = ln(\alpha) - \beta cpus(k)$$
$$\ln[TTF(k+1) - \gamma_0] = ln(\alpha) - \beta cpus(k) - \beta \Delta cpus$$

- Dynamics: $ln[TTF(k+1) - \gamma_0] = ln[TTF(k) - \gamma_0] - \beta \Delta cpus$
- Dead-beat control law: $\Delta cpus = -\frac{1}{\hat{\beta}} ln \left[\frac{set\_point - \gamma_0}{TTF(k) - \gamma_0} \right]$

ACS for a single container
Testing — Performance Analysis

(Workload profile: ML training process - MNIST data-base)

Main goals:

- Analyze the importance of γ_0 for the control input evaluation;
- Verify how the system adapts itself to new contexts :
  - change in set-point values;
  - resource usage limitation.

Common aspects in all test case phases

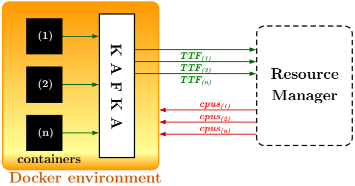
Candidate Function	**Hyperbolic**		**Exponential**	
Control Strategy	**ACS1**	**ACS2**	**ACS3**	**ACS4**
Set-point tracking	YES		NO	
Transient response	best	ok	ok	
$\hat{\beta}$ behavior [2]	constant		constant	
System response without considering γ_0	best	well-managed	ok	

Comparing all control strategies: ACS1 presents the best operation

Approach	Set-point values			
	12.5s	25.0s	50.0s	100.0s
SLA-oriented	**ACS 1**	**ACS 3**	**ACS 1**	**ACS 1**
resource usage-oriented	**ACS 1**	**ACS 1**	**ACS 1**	**ACS 1**

[2] in steady state

OACS for multiple containers
System Structural Architecture

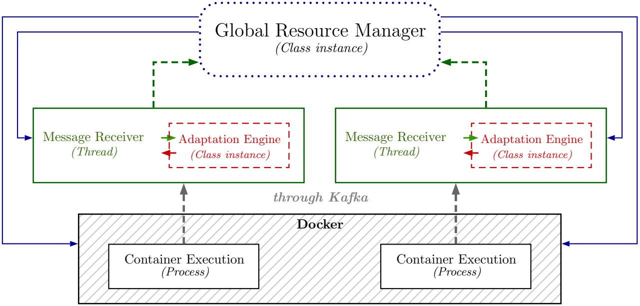


New functionalities required to handle multiple containers:

- Autonomous assignment of variables for container identification;
- Autonomous initial allocation of resources, based only on the current total amount available;
- Synchronization of events coming from different containers - use of Semaphores.

OACS for multiple containers

System Structural Architecture



OACS for multiple containers
The OACS design

Update on the Problem formulation

- SLA: an upper bound in execution time of a container;
- No *a priori* knowledge about the parameters of workload characteristics;

Constraint 1 *The amount of resources to be allocated to each container must be a positive value;*

Constraint 2 *The sum of all allocated resources in a given time cannot exceed the total number of resources provided by the infrastructure.*

Reformulating the dynamics of ACS 1

$$\begin{aligned} TTF(k+1) &= TTF(k) - \beta \tfrac{cpus(k)}{cpus(k+1)} \tfrac{\Delta cpus}{[cpus(k)]^2} \\ &= TTF(k) - \beta u(k) + \beta u(k+1) \quad , u(k) = \frac{1}{cpus(k)}. \end{aligned}$$

OACS for multiple containers

The OACS design - Control law design using MPC
The unconstrained control problem

Planta defined as a container

$$x(k+1) = x(k) - \beta u(k) + \beta u(k+1)$$
$$y(k) = x(k)$$

Control law design

MPC: control law considers a prediction horizon N_P

In a matrix form: $\mathbf{y} = \mathbf{f}x(k_i) - \beta \mathbf{f}u(k_i) + \beta \mathbf{I}_{N_P}\mathbf{u}$,

$$\begin{aligned}
\mathbf{y} &= [y(k_i+1|k_i) \quad y(k_i+2|k_i) \quad \dots \quad y(k_i+N_P|k_i)]^T, \\
\mathbf{f} &= [1 \quad 1 \quad \dots \quad 1]^T, \\
\mathbf{u} &= [u(k_i+1|k_i) \quad u(k_i+2|k_i) \quad \dots \quad u(k_i+N_P|k_i)]^T
\end{aligned}$$

Optimization

Cost function: $J = (\mathbf{r_s} - \mathbf{y})^T(\mathbf{r_s} - \mathbf{y}) + \mathbf{u}^T\bar{\mathbf{R}}\mathbf{u}$,
with $\bar{\mathbf{R}} = r_w \mathbf{I}_{N_P}$ and $\mathbf{r_s}^T = [1 \dots 1]r(k_i)$

$$\frac{\partial J}{\partial U} = 0 \implies \mathbf{u}^* = \beta(\beta^2 \mathbf{I}_{N_P} + \bar{\mathbf{R}})^{-1}(\mathbf{r_s} - \mathbf{f}x(k_i) + \beta \mathbf{f}u(k_i))$$

OACS for multiple containers

The OACS design - Control law design using MPC
The unconstrained control problem

Control law design - Implementation

Only the first predicted value is implemented:

$$u(k+1)^* = \underbrace{[1 \quad 0 \quad \dots \quad 0]}_{N_P}\mathbf{u}^* = \frac{\beta}{\beta^2 + r_w}(r_s - x(k_i) + \beta u(k_i))$$

The prediction horizon has **no** influence in the optimal control law.

Tuning parameter r_w:

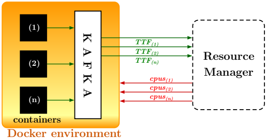
$$x(k+1) = x(k) - \beta u(k) + \beta \frac{\hat{\beta}}{\hat{\beta}^2 + r_w}(r_s - x(k_i) + \hat{\beta} u(k_i))$$

Two conditions: $\frac{\beta\hat{\beta}}{\hat{\beta}^2+r_w} = 1$ and $\frac{\beta\hat{\beta}^2}{\hat{\beta}^2+r_w} = \beta \implies r_w = 0$ and $\hat{\beta} = \beta$.

OACS for multiple containers

The OACS design - Control law design using MPC
The unconstrained control problem

- Every single container is considered as different SISO plant without coupling variables



The optimal control solution - the unconstrained case

For a single container: $u(k+1)^* = \frac{1}{\hat{\beta}}(r_s - x(k_i)) + u(k_i)$

Extending to multiple containers:

$$\mathbf{u}(k_i+1)^* = (\mathbf{I}_j \mathbf{b}(k_i))^{-1}(\mathbf{r_s} - \mathbf{x}(k_i)) + \mathbf{u}(k_i) \quad (1)$$

where

$$\begin{aligned} \mathbf{b}(k_i) =& [\hat{\beta}_1(k_i) \quad \hat{\beta}_2(k_i) \quad \dots \quad \hat{\beta}_j(k_i)]^T, \\ \mathbf{r_s} =& [r_{s1} \quad r_{s2} \quad \dots \quad r_{sj}]^T, \\ \mathbf{x}(k_i) =& [x_1(k_i) \quad x_2(k_i) \quad \dots \quad x_j(k_i)]^T, \\ \mathbf{u}(k_i) =& [u_1(k_i) \quad u_2(k_i) \quad \dots \quad u_j(k_i)]^T, \end{aligned} \quad (2)$$

OACS for multiple containers

The OACS design - Control law design using MPC
The constrained control problem

The optimal control solution - the constrained case

$$
\begin{aligned}
\min_{cpus_n} \quad & \sum_{n=0}^{j} (r_{sn} - x_n(k_i+1))^2 \\
\text{s.t.} \quad & x_n(k+1) = x_n(k) - \hat{\beta}_n u_n(k) + \hat{\beta}_n u_n(k+1) \\
& u_n(k_i) = \frac{1}{cpus_n} \\
& cpus_n \geq min\_cpus \\
& \sum_{n=0}^{j} cpus_n < total\_cpus
\end{aligned} \tag{3}
$$

OACS for multiple containers

The OACS design - Control law design using MPC
The constrained control problem

The optimal control solution - the constrained case

$$
\begin{aligned}
\min_{cpus_n} \quad & \sum_{n=0}^{j} \hat{\beta}_n^2 u_n^2(k_i+1) - 2\hat{\beta}_n u_n(k_i+1)(r_{sn} - x_n(k) + \hat{\beta}_n u_n(k)) \\
\text{s.t.} \quad & u_n(k_i) = \frac{1}{cpus_n} \\
& cpus_n \geq min\_cpus \\
& \sum_{n=0}^{j} cpus_n < total\_cpus
\end{aligned}
\tag{3}
$$

OACS for multiple containers

The OACS design - The global solution

Algorithm 1: The optimal adaptive control input evaluation algorithm

Input : The **container_id** whose resource allocation is about to be updated;
The total amount of resources available **total_cpus**;
The data $(cpus(k-1), ttf(k-1))$ from **container_id** ;
For the other running containers, their current resource allocation **cpus(k)** ;
For each running container: the estimated value $\hat{\beta}$ and the set-point value *sp*.
Output: The amount of resources **cpus(k+1)** to be allocated to **container_id**

```
Estimate TTF(k) for the running containers, using TTF(k) = α̂ + β̂/cpus(k)
Form vectors b(k_i) , r_s, x(k_i) and u(k_i) from Equation (2)
Perform unconstrained optimal control law (Equation (1))
if u(k_i + 1)* < 0 then
    Perform additional mechanism
if ∑ 1/u(k_i + 1)* > total_cpus then
    Perform constrained optimization given by (3)
cpus(k+1) ← 1/u(k_i + 1)*
```

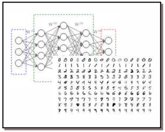
Implementing the global optimal solution

- Only a request for resource allocation is treated each time;
- Only part of the solution due to the container which requested the resource allocation is implemented.

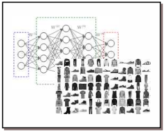
OACS for multiple containers

Testing Campaigns

- Workload profiles:



ML Training Process
MNIST data-base



ML Training Process
Fashion-MNIST data-base

Video encoding
Youtube video

OACS validation for the deployment of multiple containers

- Verify the management of all SLAs, including the introduction of new contexts at run-time:
  - The variation of the total amount of resources available;
  - The change in set-point values.
- Includes a monitoring of the CPU Clock Speed.

OACS for multiple containers

Tests — Real-time operation

Varying set-point with changes in environment

OACS for multiple containers
Testing - Performance Analysis

- **Changes in set-point values**:
  - Solution without important overshoots / undershoots;
  - When not all SLAs can be satisfied, the optimal solution minimizes the set-point tracking error.
- **Changes in the total available amount of cores**:
  - The first container to face this constraint is the most penalized in its performance;
  - The other containers reach quickly their new operating points.
- **Important events:**
  - Introduction of new containers / Ending of a container execution;
  - Change of the level of resource availability / parallel capacity;
  - Presence of oscillations in the CPU Clock Speed.

Change in the workload characteristic - change in the operating point

Adaptation mechanisms were vital to handle these unexpected events.

Conclusion and Perspectives

Conclusion

- With a **new model defining an iterative workload**, the OACS provides a solution that optimizes SLAs, respecting the available resources in a computational infrastructure.
- The resource allocation evaluation implemented in the OACS **only** uses data obtained at run-time.
- The use of **control techniques** associated to **decision-making mechanisms** guarantee the **self-adaptive** behavior of the system.

Perspectives

- Scale up the OACS to a larger number of applications;
- Develop a new multivariable control strategy at the level of each application;
- The extension of this work to other types of workloads.

Thank you!
Questions?