AI4DL: Mining Behaviors of Deep Learning Workloads for Resource Management

12th USENIX Workshop on Hot Topics in Cloud Computing, July 2020

Josep L. Berral, Chen Wang, Alaa Youssef

Barcelona Supercomputing Center
IBM – Container Cloud Platform

Presented by:

Josep Lluís Berral
josep.berral@bsc.es

Context (Background & Motivation)

- **Background**
  - Concepts:
    - *Cloud-native DL workloads*
    - *Efficient resource usage*
  - Problem to tackle: Better workload management/provisioning
  - Our Environment: Containers for Deep Learning training applications
- **Motivation**
  - Increasing use of DL services on the Cloud
    - *Not just inference but training!*
  - DL platforms over Cloud
    - *Different providers*
    - *Resources changing/increasing over time…*
  - Containers allow higher usage/sharing of machines
    - *Must manage better to avoid competition/underprivison*

Learn about the workload → Make better decisions

- Resource management: "How many resources should I allocate for that job?"
- Auto-Scaling: "Increase/decrease container provisioning?"

Introduction

- **In this work:**
  - <u>Discover behavior phases</u> from resource usage metrics → CRBM for multi-dimensional time-series
  - <u>Estimate resource demand</u> from phase information → Statistical information
  - <u>Devise container auto-scaling</u> policies for DL workloads → Based on phase identification + stats

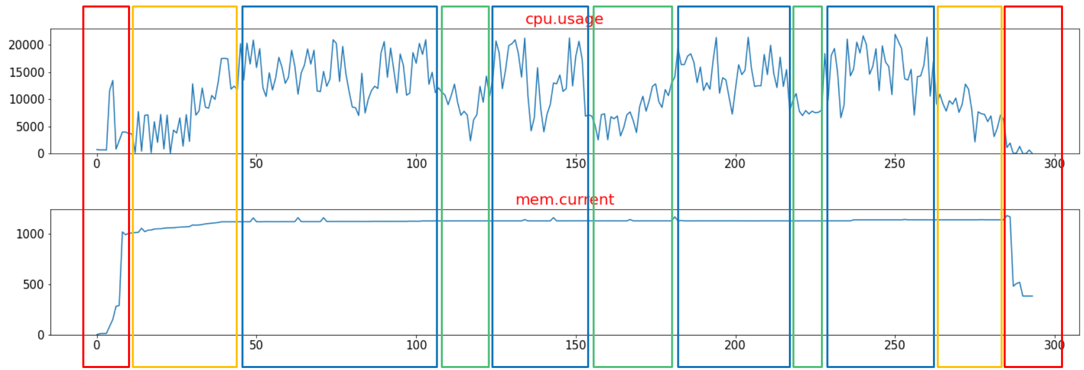
- **Basic Questions:**

"Can we identify common behaviors in workloads?"

(Characterization / Phase discovery)

"Can we exploit that to properly provision?"

(Learning phase characteristics)



Previous Work

- **Workload characterization and learning**
  - Previous work:
    - *Use of data mining techniques to model workloads (ALOJA project) **
    - *Characterization / Detection of Phases (Hi-EST project) ***

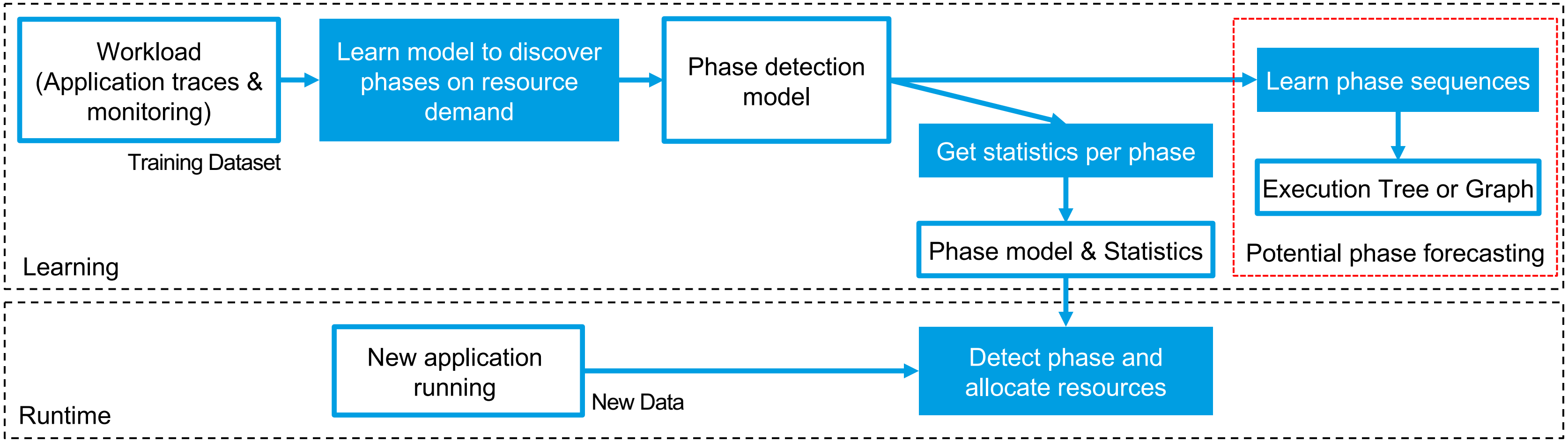
Modeling towards Optimal Configuration for Hadoop/Spark

  - Related work:
    - *Focus on direct resource prediction / continuous modeling*
      - *Problems with burstiness / variability and sudden behaviors*
      - *Phase-modeling to detect "shapes" rather than punctual values*
    - *Use of Time-Series techniques*
      - *Systems with high variability are better modelled by "periods" (here with phases)*
      - *Adaptive modeling may require constant learning. Here we try to reduce model update to extremely novel workloads*
    - *Reactive methods*
      - *Constant adaption of resources. Here we leverage anticipation or recognition of current trend*

* "ALOJA: A Framework for Benchmarking and Predictive Analytics in Big Data Deployments" http://dx.doi.org/10.1109/TETC.2015.2496504
** "Automatic Generation of Workload Profiles using Unsupervised Learning Pipelines" http://dx.doi.org/10.1109/TNSM.2017.2786047

Methodology

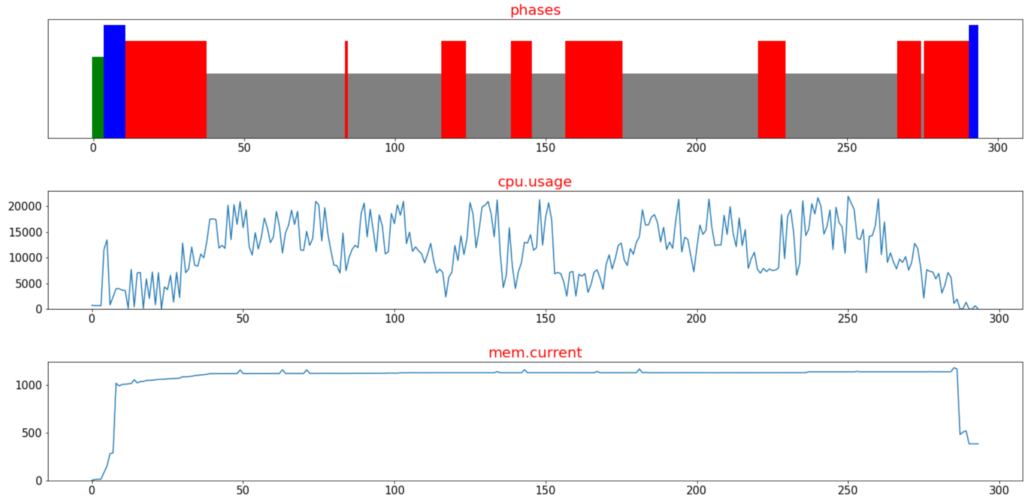
- **Characterization to DL containerized workloads**
  - Training and Inference process:



Phase Discovery and Detection

1. **Phase Discovery and Detection**
   - *Discover different behaviors on resource demand*
   - *Build a model capable to identify those on-line*
   - *Keep the behavior statistics for next provisioning*

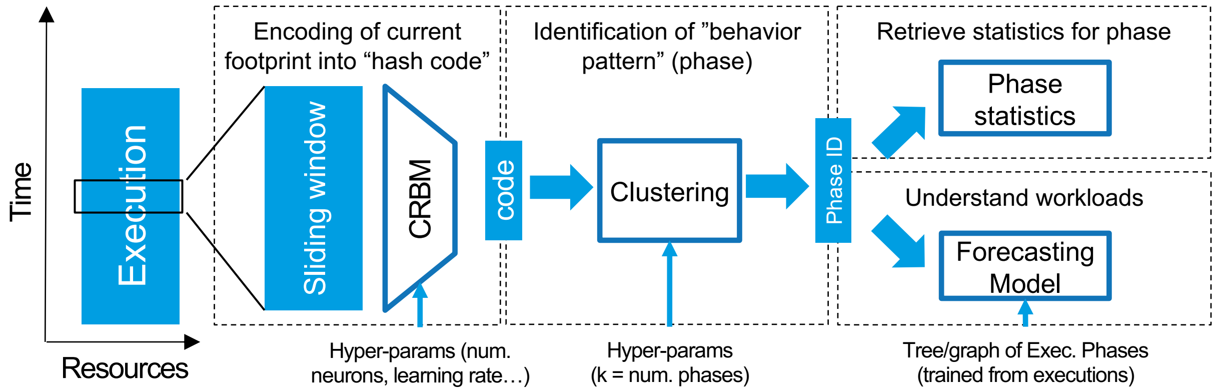
– Example:



Collected Information per phase:

- Green Phase
  - Warm up / Low resource demand
- Blue phase
  - Low memory / Low CPU
- Red Phase
  - CPU w. variation / High Memory
- Gray Phase
  - High CPU & Mem demand

Phase Discovery and Detection

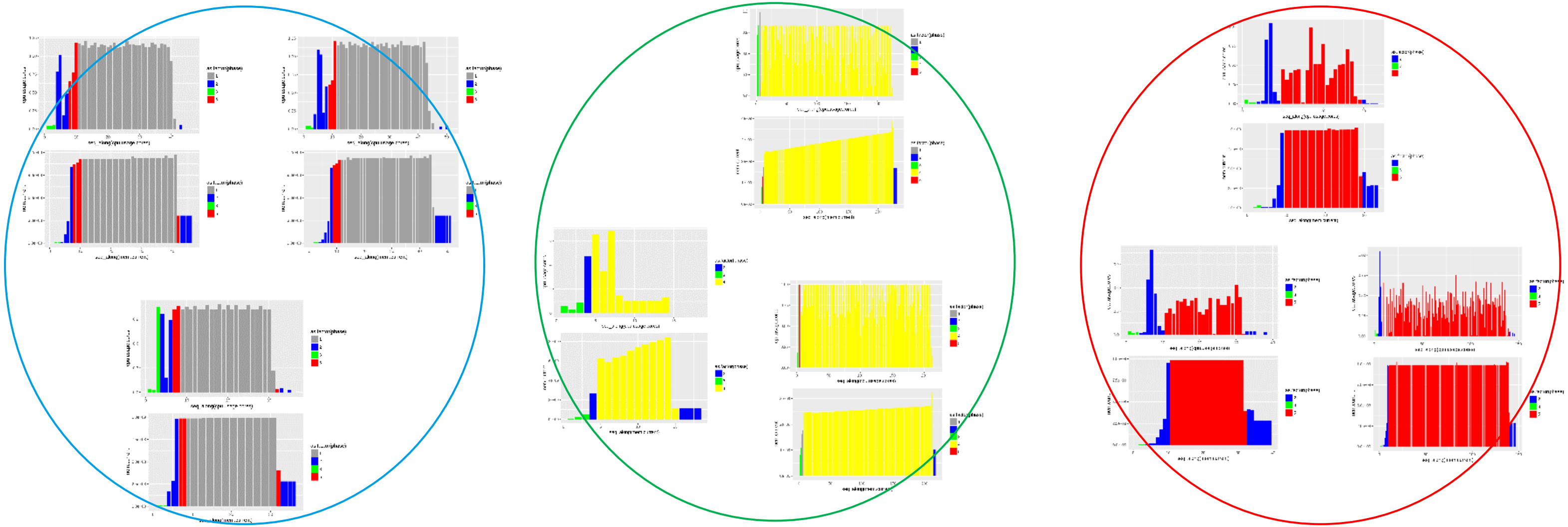


- Conditional Restricted Boltzmann Machines (CRBM)
  - *Multi-dimensional Time-series "encoder"*
  - *"Code" shares similarity among similar inputs*
- Clustering methods
  - *Find similar "codes" → similar "behaviors along time"*
  - *E.g. k-means method ("k" with best cluster cohesion, SSW)*
- Characterization through Phases
  - *Each phase has characteristics "mean", "st.dev", "min/max", …*
  - *Each workload is represented by a sequence of phases*

Modeling Executions as Phase-series

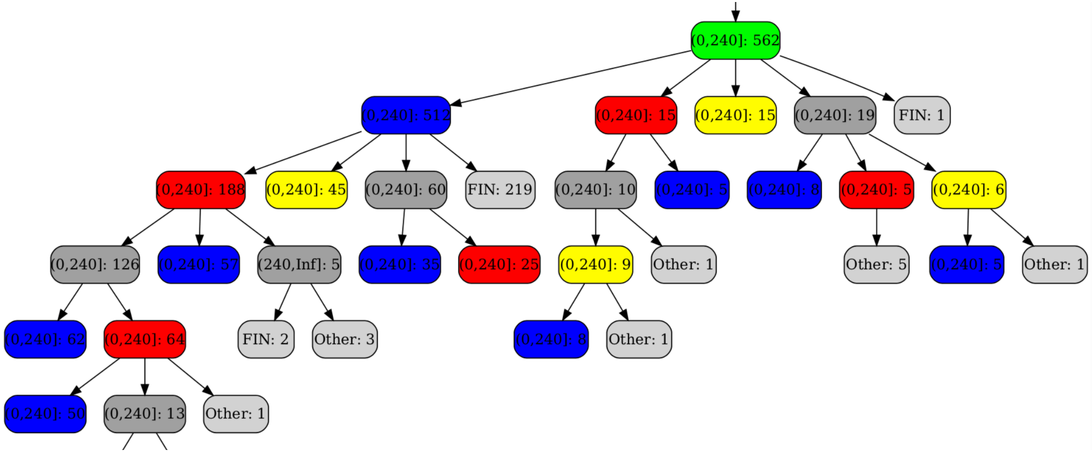
2. **Modeling Executions as Phase-series**
   - *Prototypes (or common workloads by phase-sequence)*
   - *Tree/Graph probabilistic representation*

– Executions by similarity. Example:



Modeling Executions as Phase-series

- **Prototype representation**
  - Probabilistic tree form
    - *Jump from phase to phase*
    - *Considering phase lengths in “bins”*



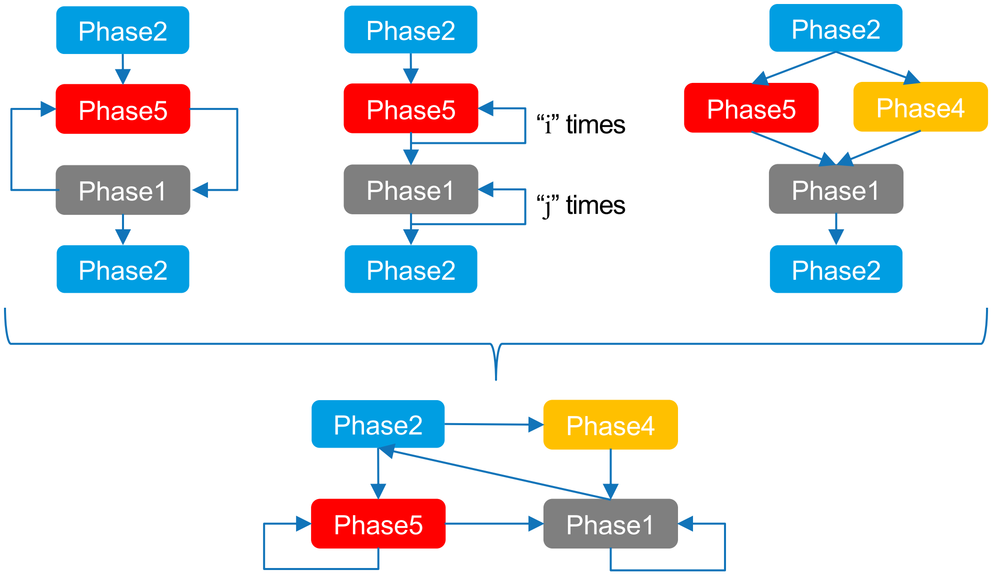
Describe sequences of phases by “tree”.

Observations

- Spawn of tree
  - Variability in our workload
  - Reduced set of standard executions
- Branches with high probability
  - Consistency on executions
  - Our prototypes

Modeling Executions as Phase-series

- **Prototype representation**
  - Graph representation
    - *E.g. State-graph or Markov Chain*
  - Solve problems found in trees:
    - *Alternate sequences of phases*
    - *Different lengths in different executions*
    - *“Convergence” in variations in middle-execution*



Resource Provisioning Policies

3. Resource Provisioning Policies

- *Dynamic vs. Adaptive vs. Phase-based policies*

– Types of Policies:

- *Dynamic Policies: "We know a priori the load for next time-window"*
- *Adaptive Policies: "We observe what happened last time-window, use that same information"*
- *Phase-based Policies: "From last time-window, we detect the current phase and its expected stats"*

– Statistic Values

- *Using "mean + 2 standard deviation": Provide the container the expected 95th percentile ceiling, to avoid outliers*
- *Using "maximum observed": Provide the container the maximum observed*
  - *Not in phase-based policy, to avoid carrying the "global maximum observed per phase"*
- *Here we can consider a tolerance margin between 0-10% for any policy*

Experiments

- **Evaluation benchmark:**
  - IBM DLaaS services, with +5500 containers

- **Set-Up**
  - Traces for **DLaaS** (Deep Learning as a Service) Kubernetes containers from IBM Watson ML services
  - Telemetry: recording of CPU & Memory demands and usage each 15 seconds.

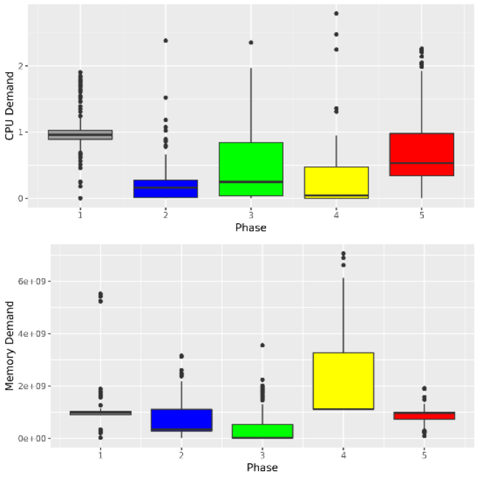
  - Dataset division
    - *Training dataset: Create and validate models, CRBMs, clustering, … → Handy set for experimentation (5000 execs)*
    - *Testing dataset: Test the method with new data → Large data set (550 execs)*

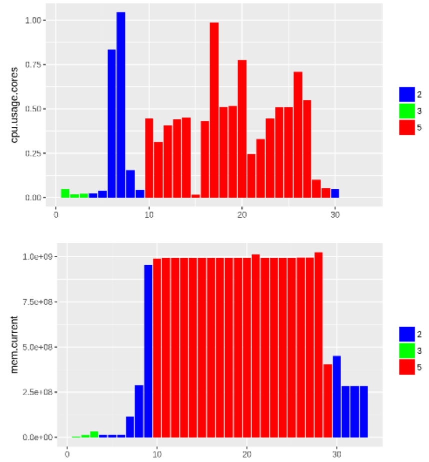
- **Training Environment**
  - CRBM package (R-cran + C + OpenBlas + GSL) + k-Means from R-base
  - Code also available in Python, open source in https://github.com/HiEST/AI4DL

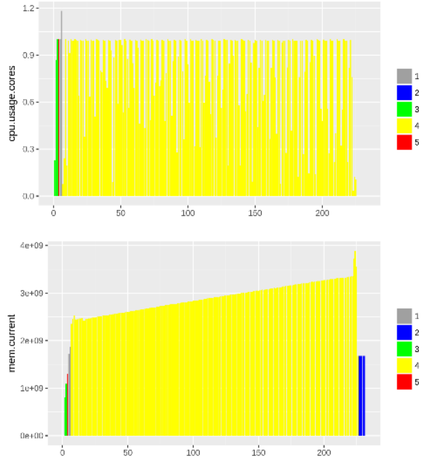
Experiments: Phase Behavior

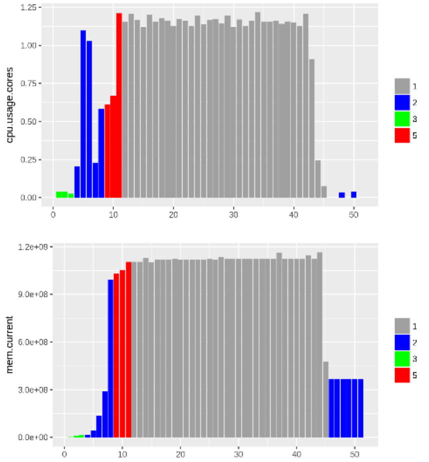
- **Identification of behaviors for each phase**
  - Phase discovery + prototype discovery (CPU and Memory)
    - *Variability and behavior for each detected phase*
    - *Discovered 6 major prototypes (here the 3 principal ones)*

X-axis: TIME (15 sec. PHASES)
Y-axis: CPU & MEM USAGE







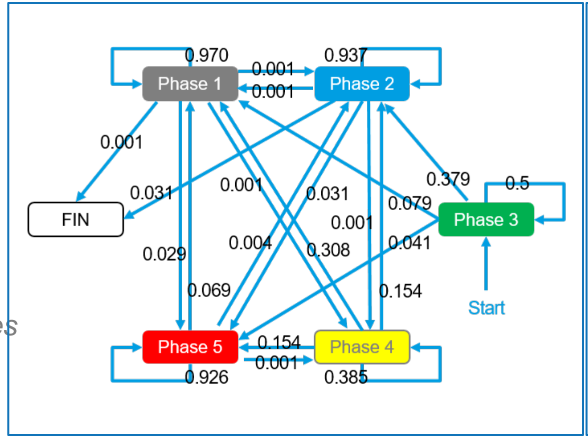


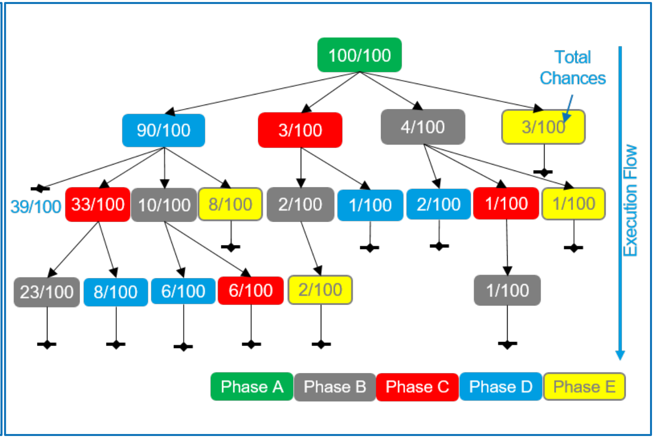
Experiments

- **Representation of Phase sequences**

  – Graph & Tree
    - *Tree: Observed 6 prototypes (branches) concentrating ~90% of executions*
    - *Also we observe their variations*

  – Phase changes
    - *Some phases are stable (easy to follow)*
    - *Others are clearly "temporary" phases (constant switch between phases)*

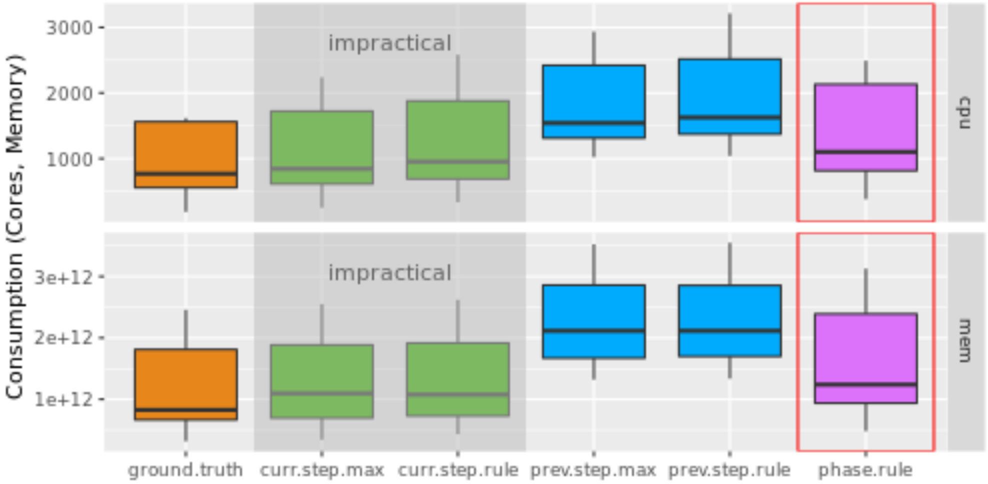




IBM Cloud

Experiments

- **Phase-based resource provisioning**
  - Re-scheduling window
    - *10 minutes (here by system policy)*
  - Evaluation dataset:
    - *Consumption tested over the full dataset of +550 executions longer than 10 minutes*
  - Consumption close to "a-priori" policies
    - *Improvement over adaptive policies*
    - *Saving up to 30% on CPU/Memory consumption in total*
  - Quality of service
    - *Fulfillment of 95% of CPU/Memory demand*
      - *Allowing over/under-provisioning margin between a -10% / +10%*
    - *No degradation compared to "prev.step" policy:*
      - *Same amount of OOM/CPU Throttling scenarios*
      - *2 out of this 5% unfulfilled are bursts or "outliers"*



Conclusions

- **Approach & Contribution**
  - Discover behavior phases from resource usage metrics
    - *Use of CRBM encoding + Clustering method*
  - Estimate resource demand from phase information
    - *Study diversity of behaviors on resources demand*
  - Devise container auto-scaling policies for DL workloads
    - *Resource allocation strategy according to specific statistics*

- Codification of DLaaS applications into "behaviors" (i.e. phases)
- Finding prototypes and phase-sequences (graph/tree representations)

- Knowledge from applications
- Specific resource demands in determined execution moments

- Leverage a-priori information from identified phases
- Better heuristic to know in advance resource demands

Conclusions

- **Discussion:**
  - Different bottlenecks in Workloads
    - *E.g. network and storage*
  - Sophistication of policies
    - *How to leverage phase information, or add new info*
  - Forecasting Phases
    - *Additional information for graph transitions*
    - *Time in the current phase towards observing a change?*
    - *MX with N-memory to avoid Loss of prototype information?*
  - Updatability of models!
    - *Do we choose models easy to adapt?*

- **Future Work**
  - Phase forecasting in workloads
    - *Refine prediction of future phases*
  - Refine resource allocations strategy per phase
    - *E.g. advance resource scheduling from a-priori phase changes*
    - *E.g. slow reduction of provisioning to prevent hysteresis and reduce re-provision rounds*
  - Containerized services for ML inference
    - *Also other kinds of workload!*

Thank you for your attention

Any questions?

Presented by:

Josep Lluís Berral
josep.berral@bsc.es