Datalog with External Machine Learning Functions for Automated Cloud Resource Configuration

Zhuoxun Zheng[1,2], Ognjen Savkovic[3], Nikolay Nikolov[4,2], Luu Huu Phuc[1], Ahmet Soylu[4,2], Evgeny Kharlamov[4,2] and Baifan Zhou[4,2]

[1]*Bosch Center for AI, Germany*
[2]*Department of Informatics, University of Oslo, Norway*
[3]*Free University of Bozen-Bolzano*
[4]*Department of Computer Science, Oslo Metropolitan University, Norway*

Abstract
Industry 4.0 and Internet of Things (IoT) technologies unlock unprecedented amount of data from factory production, posing big data challenges. In that context, distributed computing solutions such as cloud systems are leveraged to parallelise the data processing and reduce computation time. As the cloud systems become increasingly popular, there is increased demand that more users that were originally not cloud experts (such as data scientists, domain experts) deploy their solutions on the cloud systems. To this end, we propose SemCloud, a semantics-enhanced cloud system, for tackling the challenges of data volume and more users. The system has been evaluated in industrial use case with millions of data, thousands of repeated runs, and domain users, showing promising results. This poster paper accompanies our full paper and focuses on Datalog rules with external machine learning functions for automated resource configuration, and provides additional discussion on formalism and implementation techniques.

Keywords
Datalog, knowledge graph, cloud configuration, machine learning

1. Introduction

Background and Challenges. Industry 4.0 focuses on smart factories that rely on IoT technology for automation. This produces massive amounts of production data, increasing the demand for data-driven solutions and cloud technology. Yet, users of these solutions and cloud technology are often not cloud experts, such as domain experts and data scientists. In a standard setting of a data science project, the team requires extensive assistance from cloud experts, whenever they want to deploy solutions or make small changes to their solutions deployed on the cloud. To facilitate the adoption of cloud systems for more projects and users, one can equip all projects with some cloud experts, or launch training programs about cloud technology. Both require careful planing to balance time, cost, and benefits.

Our Approach. We notice that the existing work on this topic addressed the cloud deployment issues only to a limited extent [1], whereby they either only focus on the formal description of cloud, or on the limited adaptability of cloud systems. To address scalability challenges in data volume and democratising cloud systems for more users, we propose SemCloud, a semantics-enhanced cloud system, that scales semantic ETL pipeline on the cloud, and allows non-cloud experts to deploy their solutions. A rough description of workflow of SemCloud is as follows: (0) non-cloud experts create knowledge graphs (KG) that represent ETL-Pipelines on a cloud system, where attributes of cloud resource configuration is under-specified; then Datalog rules execute in three steps: (i) graph extraction rules write populate rule predicates by extracting information from the ETL-pipeline KGs; (ii) resource estimation rules estimate the resource consumption for the given pipeline assuming that there is only one computing node (assuming infinite large node); (iii) resource configuration rules that find the optimal resource allocation in distributed computing given the pipeline. This poster paper accompanies our full In-Use paper [2]. The paper is inspired by a industrial use case of manufacturing quality monitoring. The paper provides additional insights and more details regarding the implementation.

2. Approach

Use Case: Distributed Semantic ETL. In our welding use case, large amount of data collected from different factories, customers, software versions are integrated and analysed in parallel to optimise the following welding production. To enable distributed ETL, we need to find a strategy that makes the ETL parallelisable. SemCloud achieves this by breaking down the ETL into pipelines of four steps: *retrieve*, *slice*, *prepare*, and *store* (Fig. 1), where data is first retrieved from databases or online streams, and then split into subsets (e.g. each belong to one welding machine) by *slice* to achieve parallel processing and storage in the following two steps. Here the cloud configuration play an important role.

KG Construction for ETL Pipelines. SemCloud provides the users GUI to construct semantic ETL pipelines and encode them into knowledge graphs, based on a SemCloud ontology (Figure 1a). The ontology *SemCloud* [3] is written in OWL 2, and consists of 20 classes and 165 axioms. It has three main classes: *DataEntity*, *Task*, *Requirement*. *DataEntity* refers to any dataset to be processed; *Task* has sub-classes that represents the four types of tasks in the data preparation: *retrieve*, *slice*, *prepare*, and *store*; and *Requirement* that describes the requirements for computing, storage and networking resources. We illustrate the generation of ETL pipelines in KGs with the example in Figure 1b. For these data, the users construct an ETL pipeline `p1` with four layers (via GUI). Firstly, data are "retrieved" from the welding factories. Thus, the layer `l1` is of type *RetrieveLayer*, and has the task `t1` of type *Retrieve*. The task `t1` has an IO handler `io`, which has an output `d1` of type *DataEntity*. Then the data are read in by a task `t2` of type *Slice*, and "sliced" into smaller pieces `d2`, `d3`. These slices are input to different computing nodes to do tasks `t3` and `t4` of type *Prepare*. Finally, all prepared data entities are stored by `t5` of type *Store*.

Datalog ML Rules. Obtaining an optimised cloud configuration is not a trivial task. Cloud experts typically try different configurations by testing the system with various settings and use heuristics to manually decide on the configurations. To this end, SemCloud uses adaptive rules in *Datalog* with aggregation and calls to external predicates learned by ML (they are adaptive because the function parameters are learned). In particular, we consider non-recursive rules of the form $B \leftarrow B_1, \ldots, B_n$, where B is a head of rule (the consequence of the rule application) and $B_1, \ldots, B_n$ are either predicates that apply join, aggregate function that filters out the

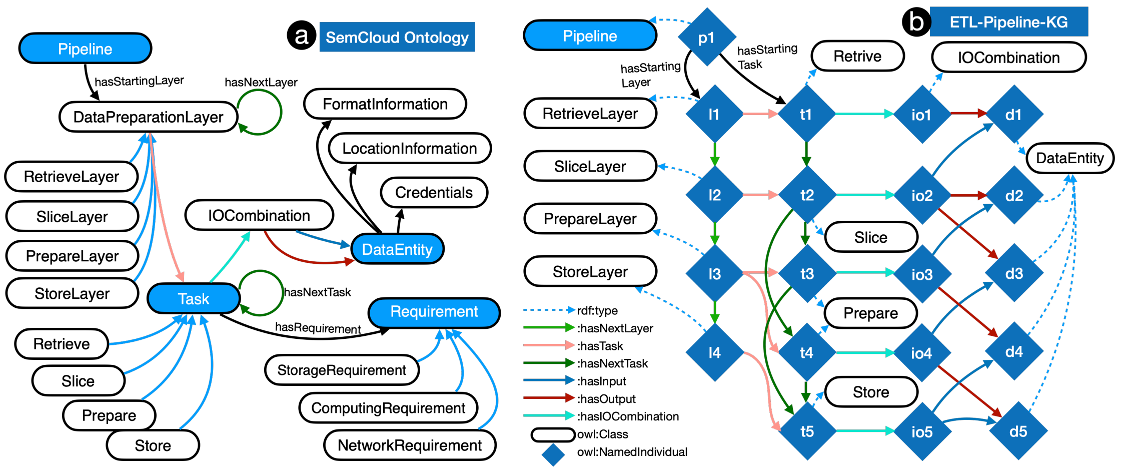

Figure 1: (a) Schematic illustration of the SemCloud ontology and (b) a KG for `ETL-Pipeline`.

results or the expression of the form $Var = @FUNCT(Vars)$. For the theory of Datalog we refer to [4]. We have six set of independent Datalog rules that are divided into three steps.

Graph Extraction Rules. These rules populate the predicates so that these predicates will be used for the resource estimation and configuration. The $rule_0$ exemplifies populating the predicate `subgraph1` that is related to the ETL pipeline `p`. Similarly, $rule_1$ creates `subgraph2(p,n,v,ms,mp ,ts,tp,nc,ns,mrs,mrp,mode)`.

```
subgraph1(p,n,v,ms,mp,ssl,spr,sst)  ←  ETLPipeline(p),
   hasInputData(p,d), hasVolume(d,v), hasNoRecords(d,n)
   hasEstSliceMemory(p,ms), hasEstPrepareMemory(p,mp)
   hasEstSliceStorage(p,ssl), hasEstPrepareStorage(p,spr)
   hasEstStoreStorage(p,sst)                                (rule_0)
```

Resource Estimation Rules. These rules are used to estimated required resource assuming one computing node. For example, $rule_2$ estimates the required slice memory (`ms`), prepare memory (`mp`), slice storage (`ssl`), prepare storage (`spr`), and the store storage (`sst`). It then stores these estimation in the predicate `estimated_resource`.

```
estimated_resource(p,ms,mp,ssl,spr,sst) ←
    subgraph1(p,n,v,ms,mp,ssl,spr,sst),
    ms=@func_ms(n,v), mp=#avg{@func_mp(n,v,ms,i):range(i)},
    ssl=@func_ssl(n,v), spr=#avg{@func_spr(n,v,ssl,i):range(i)},
    sst=@func_sst(n,v,ssl,spr)                               (rule_2)
```

where `@func_ms`, `@func_ssl`, `@func_sst`, etc. are parameterised ML functions whose parameters are learnt in the *rule parameter learning*. In the implementation, those are defined as external functions that are called in the grounding phase of the program and are replaced by concrete values [5]. We also have other estimation rules for other resources, such as CPU consumption.

Resource Configuration Rules. These rules find the optimal cloud configurations based on the estimated cloud resource. $rule_3$ is an example for deciding the slicing strategy and the storage strategy, and finding the optimal resource configuration such as the chuck size (`nc`), slice size (ns), memory reservation for *slice* (mrs) and for *prepare* (mrp). In essence, $rule_3$ stipulates that if the maximum of estimated slice memory (ms) and prepare memory (mp) is greater than a given threshold (c1*nm), and the maximum of estimated slice storage (ssl), prepare storage (spr), and store storage (sst) is smaller than (or equal to) another threshold (c2*ns), then the chosen strategy for the given pipeline is *slicing* (thus nc and ns are computed), and *fast storage* (fs), where the thresholds are calculated from cloud attributes.

```
configured_resource(p,nc,ns,fs,mrs,mrp) ←
    subgraph2(p,n,v,ms,mp,ts,tp,nc,ns,mrs,mrp,mode),
    estimated_resource(p,ms,mp,ssl,spr,sst),
    CloudAttributes(c,c1,c2,c3,nm,ns,fs,cs),
    #max{ms,mp} > (c1 * nm), #max{ssl,spr,sst} <= (c2 * ns),
    nc = @func_fs_1(n,v,ts,tp), ns = @func_fs_2(n,v,ts,tp),
    mrs = #min{ms, #max{@func_ss(n,v,nc,ns), c3*ms}},
    mrp = #min{mp, #max{@func_pn(n,v,nc,ns), c3*mp}}                    (rule3)
```

Rule Parameter Learning with ML. The functions in the adaptive rules are in the form of ML models. The *resource estimation rules* are selected from the best model resulting from training three ML methods and the pilot running statistics. These three ML methods are *Polynomial Regression (PolyR)*, *Multilayer Percetron (MLP)*, and *K-Nearest Neighbours (KNN).* We selected these three methods because they are representative classic ML methods suitable for the scale of the pilot running statistics. The *resource configuration rules* are trained with the three ML methods and with optimisation techniques, such as Bayesian optimisation or grid search. For example, the functions @func_fs_1 and @func_fs_2 that find the optimal chuck size (nc) and slice size (ns) are trained by finding the arguments of (nc,ns) for the minimal total computing time ($t_{\texttt{total}}$):

$$\mathrm{nc}, \mathrm{ns} = \underset{\mathrm{nc,ns}}{\arg\min}\ t_{\texttt{total}} = \underset{\mathrm{nc,ns}}{\arg\min}\ f(\mathrm{v}, \mathrm{n}, \mathrm{nc}, \mathrm{ns}, t_{\texttt{slice}}, t_{\texttt{prepare}})$$

3. Implementation and Evaluation

Implementation. We implement the Datalog rules with DLV and external functions as Python plugins [6]. In particular, we first write and compile the external functions written in Python, then we define the interface for the functions in the Datalog program, and then link the Datalog program with the compiled Python code when running the DLV. For some complicated rules with reusable part, we introduce a auxiliary predicate that stands for the reusable to improve efficiency. A rule in the form of $B \leftarrow B_1, ..., B_n$ becomes two parts: (PartI): $B_{aux} \leftarrow B_1, ..., B_m$, (PartII): $B \leftarrow B_{aux}, B_{m+1}, ..., B_n$, where PartI is reused also in other rules. For instance, $rule_3$ applies for one of the four cases of #max{ms,mp}>(c1*nm), #max{ssl,spr,sst}<=(c2*ns), while there exist other three cases for the comparison (<=,<=), (<=,>), (>,>). We introduce the auxiliary predicate configured_resource_aux to replace the first three lines (subgraph2 to CloudAttributes), which will be reused in the inference of the three other cases.

Evaluation and Discussion. To verify the time efficiency and accuracy of the rule parameter learning and inference, we run SemCloud repeatedly 3562 times and gather pilot running statistics. These statistics are split to 80% for training and 20% for testing and inference. Three ML models are trained and tested. After a grid search, the selected hyper-parameters are, *PolyR*: 4 degree; *MLP*: 2 hidden layers with neurons 10 and 9; *KNN*: 2 neighbours. We use these performance metrics: *normalised mean absolute error* (*nmae*), minimal training data amount (Min. $|\mathcal{D}_{train}|$ for yielding satisfactory results, optimisation time (Opt. time), learning time (for ML training) and inference time (including the inference time of ML and Datalog). The results (Table 1) show that PolyR has the best prediction accuracy, requires the least training data, and consumes the least time. Therefore, PolyR generates the best results and is selected for the use case. We presume the reason is that PolyR works better with small amounts of and not very complex data (3562 repeated running statistics). The results show that our approach exhibit promising inference accuracy and time efficiency for automated cloud resource configurations.

4. Conclusion and Outlook

Table 1: Parameter learning and reasoning on Intel Core i7-10710U.

Metric	PolyR	MLP	KNN
nmae	<u>0.0671</u>	0.0947	0.0818
Min. $\|\mathcal{D}_{train}\|$	<u>7.42%</u>	50.97%	10.00%
Opt. time	<u>1.12s</u>	174.32s	7.25s
Learning time	<u>20.82ms</u>	120.31ms	27.52ms
Inference time	<u>$\leq$1ms</u>	<u>$\leq$1ms</u>	$<$5ms

This poster paper accompanies our full paper [2] with a focus on Datalog rules with external machine learning functions and provides additional discussions on formalism and implementation techniques. The research is under the under the umbrella of Neuro-Symbolic AI for Industry 4.0 at Bosch. We aim at enhancing manufacturing technology with both symbolic AI [7] for improving transparency [8], and ML for prediction power. We will further improve the performance of the KG embedding method and develop other complementary technologies, such as ontologies [9], ontology-based data access, etc.

Acknowledgements. The work was partially supported by the European Commission funded projects DataCloud (101016835), enRichMyData (101070284), Graph-Massivizer (101093202), Dome 4.0 (953163), OntoCommons (958371), and the Norwegian Research Council funded projects (237898, 323325, 309691, 309834, and 308817).

References

[1] L. Youseff, M. Butrico, D. Da Silva, Toward a unified ontology of cloud computing, in: 2008 Grid Computing Environments Workshop, IEEE, 2008, pp. 1–10.
[2] B. Zhou, N. Nikolov, Z. Zheng, X. Luo, O. Savkovic, D. Roman, A. Soylu, , E. Kharlamov, Scaling data science solutions with semantics and ML, in: ISWC, 2023.
[3] The SemCloud Ontology, 2023. Open source under: https://github.com/nsai-uio/SemCloud.
[4] S. Paramonov, et al., An asp approach to query completeness reasoning, TPLP 13 (2013).
[5] N. Leone, et al., The dlv system, in: JELIA, Springer, 2002, pp. 537–540.
[6] Dlvhex python plugin manual, http://www.kr.tuwien.ac.at/research/systems/dlvhex/doc2x/group__pythonpluginframework.html, 2016.
[7] D. Rincon-Yanez, et al., Addressing the scalability bottleneck of semantic technologies at bosch, ESWC Industry (2023).
[8] Z. Zheng, et al., Executable knowledge graph for transparent machine learning in welding monitoring at bosch, in: CIKM, 2022, pp. 5102–5103.
[9] B. Zhou, Z. Zheng, D. Zhou, Z. Tan, O. Savković, H. Yang, Y. Zhang, E. Kharlamov, Knowledge graph-based semantic system for visual analytics in automatic manufacturing, ISWC, 2022.