Understanding Data Centers from Logs: Leveraging External Knowledge for Distant Supervision

Chad DeLuca, Anna Lisa Gentile(✉), Petar Ristoski, and Steve Welch

IBM Research Almaden, San Jose, CA, USA
{delucac,welchs}@us.ibm.com,
{annalisa.gentile,petar.ristoski}@ibm.com

Abstract. Data centers are a crucial component of modern IT ecosystems. Their size and complexity present challenges in terms of maintaining and understanding knowledge about them. In this work we propose a novel methodology to create a semantic representation of a data center, leveraging graph-based data, external semantic knowledge, as well as continuous input and refinement captured with a human-in-the-loop interaction. Additionally, we specifically demonstrate the advantage of leveraging external knowledge to bootstrap the process. The main motivation behind the work is to support the task of migrating data centers, logically and/or physically, where the subject matter expert needs to identify the function of each node - a server, a virtual machine, a printer, etc - in the data center, which is not necessarily directly available in the data and to be able to plan a safe switch-off and relocation of a cluster of nodes. We test our method against two real-world datasets and show that we are able to correctly identify the function of each node in a data center with high performance.

1 Introduction

Understanding the functions implemented within a data center (which software processes are running, and where) is often an extremely challenging problem, especially in situations where good documentation practices are not in place, and machine re-configurations, software updates, changing software installation, failures, malevolent external attacks, etc. make the ecosystem difficult to understand. When data center migration is offered as a third party service, it is important to enable the practitioners to quickly and precisely characterize the nature, role, and connections - which are often not explicitly declared - of the multitude of nodes in the data center to migrate, in order to offer a reliable service.

Migrating data centers, either physically relocating machines or logically moving applications to the cloud, is a time and resource intensive task. Preparing a migration plan, especially in the absence of well documented information about the inner workings of the datacenter, involves intensive data analysis. Often the practitioners have to rely heavily on logs and network activities of each node in

J. Z. Pan et al. (Eds.): ISWC 2020, LNCS 12507, pp. 601–616, 2020.
https://doi.org/10.1007/978-3-030-62466-8_37

the data center to understand its cartography. Discovering and understanding connections and dependencies can be very laborious and missing any component of a dependency can result in unplanned outages during, or after, a migration. Traditional data analysis tools offer little support during the planning phase, which typically requires a significant amount of labor.

In this work, we propose a data exploration solution that allows the subject matter expert (SME) to interactively augment the collected data with structured knowledge and semantic information which is not initially present in the data. We combine traditional Information Extraction techniques together with human-in-the-loop learning to construct a semantic representation of the functions provided by the data center.

The main contribution of this work is a novel technique to create a semantic representation of a data center. Knowledge extraction is performed with a human-in-the-loop model: we first collect available knowledge about software processes from the Linked Open Data (LOD) cloud and use it in a distant supervision fashion to generate initial tags for each node in the data center; an SME validates (accept/reject/correct) the proposed tags; the validated tags are used to train several learning models and label all the processes from each node in the data center. The SME validates new annotations and the process can be repeated until the coverage is considered satisfying.

The nature of this problem is unique in the sense that, while for many processes in the data center we have useful textual information (long process name strings captured from logs), for many others we only have information about ports, connections, etc, without textual logs. Using all available textual content from those processes with logs, we generate initial tags using knowledge from the LOD. The main advantage of our solution is that we effectively exploit external knowledge to create tags and bootstrap the annotation process, training the model using both the textual information as well as the graph structure. We then apply and refine the models on the entire dataset representing the data center, including nodes where no textual content is available. Finally, all the enriched data is rendered with graph visualization tools. The SMEs have access to combined information about the nodes' logs, together with iteratively added knowledge, which creates an intelligible cartography of the data center.

In the following, we give an overview of related work (Sect. 2), formally define the problem (Sect. 3), introduce our approach (Sect. 4), followed by an evaluation (Sect. 5). We draw conclusions and discuss potential future work in Sect. 7.

2 State of the Art

The literature about data center analysis is quite broad, in the following we will highlight the differences of our proposed work in terms of (i) methods, (ii) scope - the ultimate task they are tackling - and (iii) utilized features.

In terms of methods, the common ground for knowledge discovery from log-like data is the usage of rules - in the form of regular expressions, filters, etc. [16] which can in turn rely on different features - e.g. tokenization, dictionaries or timestamp filters. While we previously investigated the use of regular expressions for this task [1], in this work we successfully explore the use of external knowledge resources and match them against the data with vector space based methods, followed by iteratively assigning validated annotations with a human-in-the-loop model.

In terms of scope, there are different classes of work in this field: some of the efforts have the goal of identifying *user sessions*, e.g., in Web based search [14] or within e-commerce frameworks [2]; some focus on identifying *anomalies and attacks* [13] while others, more similar in scope to this work, attempt to generate a *semantic view of the data* [5,9,17]. The Winnower system [13] is an example of a tool for monitoring large distributed systems, which parses audit records (log files) into provenance graphs to produce a behavioral model for many nodes of the distributed system at once. The main difference with our work is that while Winnower builds signatures of "normal behavior" for nodes - to be able to identify an anomaly - our goal is to semantically characterize the function of each node in the data center, rather than produce a binary classification of normal vs anomalous behavior. The majority of works that focus on generating a semantic view of data centers are substantially ontology-based solutions [5,9], where the logs are aligned to a specific available knowledge representation of the data center. While in this work we also create a semantic representation of the data, we propose a knowledge selection and refinement approach: we use pre-existing target knowledge but we let the final semantic representation emerge from the human interaction. The SME can add, discard, and modify semantic tags to characterize the logs, and we collect and organize them in a growing consolidated domain knowledge set. The work by Mavlyutov et al. [17] is the most similar to ours, in the sense that they analyze logs without using any pre-existing target knowledge, rather letting the semantic representation emerge from the data. They propose Dependency-Driven Analytics (DDA), which infers a compact dependency graph from the logs, and constitutes a high level view of the data that facilitates the exploration. The main difference is that the purpose of DDA is to offer compact access to the log files for tracking provenance information and allowing for different levels of inspection, e.g. to retrieve all the logs related to certain particular job failures or the debug recurring jobs etc. Differently from DDA, we generate high level, semantic views of the data center and the implemented functions, rather than semantically indexing the logs files. The logs are only exploited to generate semantic tags for the nodes in the data center, with the aid of a human-in-the-loop paradigm.

Last, but not least, logs are not the only source of information available in data centers. A plethora of sources of data are available, including network data, configuration management, databases, data from monitoring devices and appliances, etc., all of which can be leveraged to generate a model of the data center [4,7,11]. While we use all this data, this is not the focus of this paper as the major novelty of our work is a methodology to leverage unstructured data (the logs) and make sense of them using external Knowledge Resources and active interaction with the subject matter experts.

Finally, it is worth noting that there exist works in the literature that focus on understanding data centers in terms of work load, memory utilization, optimization of batch services etc. [12,21,23]. These works are also encouraged by benchmarking initiatives, such as the Alibaba Cluster Trace Program[1] that distributes data about some portion of their real production data center data. The purpose of these types of work is entirely different from this work, as we do not look at performance optimization but we aim to produce a detailed "semantic inventory" of a data center, that depicts all the applications running on each node, which we make consumable in a visual interface via (i) graph exploration, (ii) summary bar that offers contextual and global information about the data center and (iii) the knowledge cards that describe each node. The ultimate purpose is that facilitating data center migration plans. Moreover, we tackle the problem of understanding a data center, even in the absence of structured and detailed data, relying on logs collected from each node. It should also be specified that large, homogeneous data centers (i.e., Google data centers) fall outside the scope of this work. These types of data centers tend to treat each node as an anonymous worker unit with management policies that are similar to cluster or hypervisor management systems.

3 Problem Statement

A data center refers to any large, dedicated cluster of computers that is owned and operated by a single organization [3]. We formally define it as follows:

Definition 1. *A data center D is a tuple $<H, P, C>$ where H is a set of hosts, i.e. physical or virtual machines, $P = p_1 \dots p_n$ is a list of processes running on each host $h \in H$, and C is a list of directed links between the processes P. Each host h has a set of features $s \in S$. Each s is a datatype property of the host, such as the operating system, the hardware platform, available network interfaces, etc. Each process p has a set of features $d \in D$, where each d can be either a datatype property of the process, such as port number, IP address, consumer/service status etc. or a relational property of the process, such as parent/child/sibling processes within the host. C contains the links between all processes in P, which express a directional relation among them: each process is either a consumer or a provider for another processes.*

Each data center D can be transformed into a knowledge graph $G = (V, E)$.

Definition 2. *A knowledge graph is a labeled, directed graph $G = (V, E)$ where V is a set of vertices, and E is a set of directed edges, where each vertex $v \in V$ is identified by a unique identifier, and each edge $e \in E$ is labeled with a label from a finite set of link edges.*

To transform a given data center D into a knowledge graph G, we convert each process p into a graph vertex v_p. Then, we generate a list of labels for each

[1] https://github.com/alibaba/clusterdata.

feature of the existing processes. For each feature of each process we generate a triple in the form $<v_p \, e_n \, e_p>$, where e_n is the feature predicate, e_p is the value of the feature for the given process v_p. Then we transform the set of links C between all the processes to triples in the form $<v_p \, c_n \, v_{pi}>$, where each type of link is represented with a set of triples, i.e., one triple for each feature of the link $<v_c \, e_n \, e_c>$.

Given a data center $D = <H, P, C>$, our objective is to use the information provided for each process, and the links to other processes, to assign a set of labels $L_p = l_1, l_2, ..., l_n$ to each process p and - by inheritance - to each host h. In the literature, this task is also known as knowledge graph type prediction.

4 Understanding Data Centers

In the following, we describe the end-to-end methodology to create a semantic representation of a data center starting from scratch, i.e. from collecting raw data from the individual nodes of the data center to creating a representation which can be fully visualized and explored as a graph.

The workflow of the system consists of four subsequent steps. The *Data Collector* process (Sect. 4.1) collects log data from a portion of nodes in the data center. The *Knowledge Matcher* component (Sect. 4.2) obtains relevant knowledge from the LOD cloud and produces candidate labels for each node in the data center, in a distant supervision fashion. The candidate labels are validated by the SME (Sect. 4.3) before being used to train several classification models (Sect. 4.4). All augmented data is made available to the users via a visual tool, the Data Center Explorer (covered in Sect. 6) which allows the SMEs to efficiently visualize data center structure and perform complex queries.

4.1 Collecting the Data

Data is collected by running a script on each machine, virtual or physical, that is deemed important. This means deploying the script to at least 10%–20% of the nodes in a data center. Broader deployments are preferable, but not required, as they reduce the human effort required in the final stages of a migration to ensure success. A typical configuration is to run the script for 2 weeks, taking a data snapshot every 15 min. Upon the first snapshot, the script records attributes considered “static”, including operating system, hardware, and network interface information. Every snapshot includes a list of all active connections at that point in time, along with every running process. An active connection includes a process (either the consumer or provider of a service), a direction indicator, a target port, and the IP address of the other side of the connection. If the collection script is running on the node at the other side of this connection, we’ll be able to combine both sets of collected data to construct a full, end-to-end definition of a particular dependency. In cases where the other side of the dependency is not running the collection script, we’ll only have a partial understanding of the dependency, given the fact that we cannot know the exact process communicating on the other node.

When running processes are captured, we record the process log string (which we refer to as *process name*) along with its Process ID (PID) and its Parent Process ID (PPID). These elements allow us to construct a process tree as it existed when that snapshot was taken. We can combine the process tree from each snapshot into a meta-tree such that, at any point-in-time, the process tree represents a subset of the process meta-tree with all relationships intact. The relationship between processes can be helpful in process identification, for example when a particular communicating process is unknown, but its parent is known. In such cases, cascading attributes from a parent process to its children may be beneficial. All collected data is transformed into a knowledge graph representation, following Definition 2 (Sect. 3).

4.2 Bootstrapping the Labelling Process: The Knowledge Matcher

The goal of this step is twofold: to identify an initial set of labels, i.e. the inventory of possible software that can run on a node, and to create an initial set of annotated processes, i.e. assign some of these labels to a portion of the processes in the data center, that is going to be used as initial training set. We tackle the task as a distant supervision problem [18]: we leverage pre-existing, structured knowledge to identify potential labels and annotate the data center processes. The idea is to use external knowledge to construct dictionaries for target concepts and use them to label the target data. The approach is not bound to any specific ontology or resource: the only assumption is to have a dictionary that contains instances of the target concept. The dictionary does not have to be exhaustive either: as we will show in Sect. 4.3, we allow the SME to expand the knowledge with any missing concept.

For this work, we construct a dictionary for the concept *Software* following the same approach as [10]. Given a SPARQL endpoint,[2], we query the exposed Linked Data to identify the relevant classes. We manually select the most appropriate classes and properties that describe the concept of interest and craft queries to obtain instances of those.

We build a vector space representation of this constructed knowledge, where each software item is represented as a tf-idf vector and the vocabulary of features is built using the software names, labels and, optionally, any of the properties which is of text type (e.g., one can collect features such as the operating system, the type of software, etc.). Similarly, we build a vector space representation of all the processes $p \in P$, and the vocabulary of features is built using all the text in the collected logs strings. We discard any annotation with similarity below a certain threshold[3] and for each $p \in P$ we select the top similar instance from the dictionary. We then use standard cosine similarity to assign one label from the software dictionary to each process $p \in P$. The intuition for using the

[2] For example the DBpoedia endpoint http://dbpedia.org/sparql.

[3] For this work the similarity threshold has been set to 0.6.

vector space model and cosine similarity is that some of the words in a log string running a certain software, will be similar to some of the information that we collected from the knowledge base - for example the path were the software is installed or the name of executable files or some of the parameters can contain the name of the software itself. The annotations produced in this step are not expected and not meant to be exhaustive, and our only purpose is to create a reliable initial training set. For this reason we want the annotations to be as accurate as possible so that they can be used to train machine learning models (Sect. 4.4).

4.3 Human-in-the-Loop: Validation and Knowledge Expansion

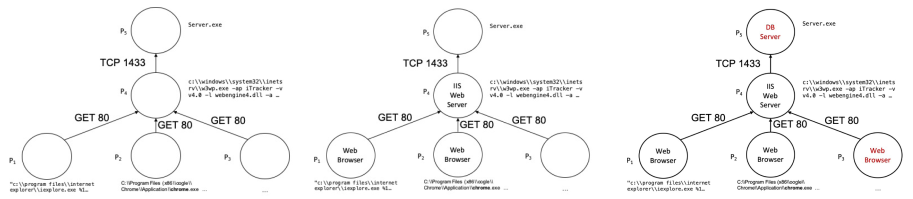
We start by ranking all retained annotations by their similarity score. During the validation process, if the SME does not accept any of the candidate annotations, they can browse the reference dictionary and manually select the appropriate one - if available. We also give the SME the possibility to manually add an entry to the dictionary, if they deem it missing, or delete any dictionary entry that they consider spurious, redundant, or incorrect. All the validated annotations are immediately added as tags for the processes and can be used as exploring dimensions to query and visualize the data (the visual tool for the graph exploration is discussed in Sect. 6).

4.4 Training the Models

The goal of this step is to leverage the validated tags to train a classification model in order to extend the tags to the whole data center. To this end, we experiment with several different neural network models, spanning from simple architectures that only exploit string information about the text in the logs to more complex graph embeddings that capture all the relations among the nodes in the data center.

For String-based approaches we use multi-label Logistic Regression (LR), multi-label Support Vector Machines (SVM) and Convolutional Neural Network (CNN) text classifiers, trained to classify log entries. The architecture of the CNN is inspired by Kim et al. [15], which has shown high performance in many NLP tasks.

In many cases of analyzing data center logs, a significant portion of the processes do not include a name or string description, i.e., either only the process ID is available or the string description is encoded, which cannot be used to infer the label of the process. To be able to correctly label such processes we use multiple graph embedding approaches. Graph embedding approaches transform each node in the graph to a low dimensional feature vector. The feature vector embeds the graph characteristics of the node and it can be used to predict the node's label, i.e., similar nodes in the graph should have the same label. To build graph embedding vectors on a data center D, we transform it to a knowledge graph $G = (V, E)$. In this work we consider 5 state-of-the-art graph embedding approaches: RDF2Vec [20], TransE [6], DistMult [24], ComplEx [22], HolE [19].


P5
Server.exe
TCP 1433
P4
c:\\windows\\system32\\inetsrv\\w3wp.exe -ap iTracker -v v4.0 -l webengine4.dll -a …
GET 80
GET 80
GET 80
P1
"c:\\program files\\internet explorer\\iexplore.exe %1…
P2
C:\\Program Files (x86\\oogle\\Chrome\\Application\\chrome.exe …
P3
…
IIS Web Server
Web Browser
Web Browser
DB Server
Web Browser

a) *Input graph of communicating processes.* **b)** *Inferred process labels after applying the string-based model.* **c)** *Inferred process labels after applying the graph-based model.*

Fig. 1. Example of a sub-graph of communicating processes being labeled using the string-based model then the graph-based model.

The output of each graph embedding approach, which is a numerical vector for each node, is used to build a classification model for node label prediction, i.e., multi-label Logistic Regression, multi-label Support Vector Machines, and CNN network.

The string-based model and the graph-based model are complementary to each other. The string-based model is used to label processes when the process name is available and the model can predict the process label with confidence higher than a specified threshold. In all the other cases we apply the graph-based model, which is able to set a label based on the structure of the graph. Figure 1 shows a simplified example of the labeling steps when using both models. The input is a sub-graph of 5 processes $p1$ to $p5$ (Fig. 1a). The process $p3$ doesn't have a name, while process $p5$ has a very generic name "Server.exe". As such, after applying the string-based model, the system can label the nodes $p1$, $p2$ and $p4$ with high confidence, as the process names are rather informative, e.g., from the processes' names we can easily infer that both $p1$ and $p2$ are web browsers, "Google Chrome" and "Internet Explorer", while $p4$ is "Internet Information Services", which is a "Web Server" (Fig. 1b). However, the string-based model cannot predict the label for process $p3$ as there is no process name provided, and for process $p5$ can only make a prediction with a very low confidence, as the name of the process is very general. In such cases, the graph-based model can help identify the missing labels. As the graph-based model is able to capture the graph structure and the relations between the processes, the model can predict the labels with high confidence when only a couple of labels are missing. In this case, the embedding vector for process $p3$ is going to be quite similar to the one from $p1$ and $p2$ as they all communicate with $p4$ with the same type of relation, which means that the model can label this process as "Web Browser" with high confidence. Regarding process $p5$, we expect that the model should be able to learn that in many other cases a process labeled as "Web Server" communicates with a process "Database Server" over a TCP link (Fig. 1c).

5 Experiments

5.1 Description of the Data

We ran experiments against data collected from 2 different data centers: a large US financial institution (*US-Financial*) and a large Canadian healthcare organization (*Canadian-Healthcare*), both representing industries with the most extreme demands on data center operations, in terms of security, resiliency, and availability. Data is collected following the methodology described in Sect. 5.1 for both data centers. The *US-Financial* dataset contains data about 2,420 discovered hosts and additional 24,127 inferred hosts (5,993 of which are external to the institution's data center). The *Canadian-Healthcare* dataset contains data about 2,139 discovered hosts, as well as 44,169 inferred hosts (526 of which external). While the data collected for *discovered hosts* is richer and contains the strings of all running processes, the information about *inferred hosts* only concerns network connections, ports, etc. For this reason, when it comes to process identification, we focus solely on discovered hosts, since these are the only hosts where we can see the names of running processes and their position within a host's process tree, nonetheless the information about *inferred hosts* is leveraged in the learning steps, especially in the graph embeddings approach (Sect. 4.4). In fact, the graph based methods largely benefit of all the information collected about links and communications among hosts. Specifically, from the 2,420 discovered hosts in the *US-Financial* dataset, we collected 1,375,006 running processes. Of these running processes, 186,861 processes are actually communicating. The *Canadian-Healthcare* dataset derived from 2,139 discovered hosts contains data on 339,555 running processes, 98,490 of which are communicating on the network. Each unique dependency forms a link between two hosts.[4] A unique dependency consists of five elements: Source Host, Source Process, Target Host, Target Process, Target Port. While both hosts and the target port are required to form the link, it is perfectly acceptable for one or both processes to be null. Given this definition of a dependency, the *US-Financial* dataset contains 2,484,347 unique dependencies and the *Canadian-Healthcare* dataset contains 22,016,130 unique dependencies, or links, between nodes.

5.2 Human-in-the-Loop Distant Supervision

As target knowledge for the tags, we use a dictionary of Software constructed from DBpedia, which initially contains 27,482 entries. Each entry in the dictionary has a unique identifier, a name and, when available, a software type. We run the distant supervision module on the 1,375,006 running processes. For each process we select the highest scoring tag, with a similarity score of at least 0.6.[5] These gave us 639,828 tagged processes, annotated with 632 different tags, which are passed to the SME for validation. At the end of this step, we have a

[4] The words host and node are used interchangeably.
[5] The threshold was selected based on empirical observation.

gold standard of 442,883 annotated processes, meaning that more than 69% of the annotations produced with distant supervision where considered correct by the SME. After this validation, 122 unique tags remained in the pool of useful ones. Some of these tags classify processes that are crucial to be identified when planning a migration, such as "Internet Information Services", "Microsoft SQL Server", and "NetBackup". During the validation process, the SME pointed out that some of the proposed tags identified software tools that they wouldn't have thought about looking for, if not otherwise prompted by the system. We provided the ability to remove entries based on softwareType, and our SME decided to remove entries of type "Fighting game", "Shooter game", "Sports video game", and others. One example of a manually added entry in this experiment was Internet Explorer, which was not initially included in the dictionary as the DBpedia type for http://dbpedia.org/page/Internet_Explorer is Television Show (http://dbpedia.org/ontology/TelevisionShow).

5.3 Training the Models

String-Based Approach. We developed two baseline string-based machine learning approaches, i.e., multi-label Logistic Regression (LR) and multi-label Support Vector Machines (SVM). We use standard bag-of-word with tf-idf weights to represent each log entry. We use the scikit-learn multi-label implementation with standard parameters.[6] Furthermore, we use a multi-label Convolutional Neural Network (CNN) text classifier and we train it to classify log entries. We selected the following parameters for the CNN model: an input embedding layer, 4 convolutional layers followed by max-pooling layers, a fully connected sigmoid layer, rectified linear units, filter windows of 2, 3, 4, 5 with 100 feature maps each, dropout rate of 0.2 and mini-batch size of 50. For the embedding layer, we use word2vec embeddings trained on 1.2 million log entries, with size 300 using the skip-gram approach. We train 10 epochs with early stopping. Using a sigmoid activation function in the last dense layer, the neural network models the probability of each class as a Bernoulli distribution, where the probability of each class is independent from the other class probabilities. This way we can automatically assign multiple labels to each log entry.

Graph Embedding Approaches. We experiment with 5 different graph embeddings approaches, i.e. RDF2Vec, TransE, DistMult, ComplEx and HolE. For the RDF2Vec approach, we use the implementation provided in the original paper.[7] For the rest of the approaches, we are using the implementation provided by AmpliGraph [8].[8] For the RDF2vec approach, for each entity in the DCE graph we generate 500 random walks, each of depth 8. We use the generated sequences to build a Skip-Gram model with the following parameters: window size = 5; number of iterations = 10; negative sampling for optimization;

[6] https://scikit-learn.org/.
[7] http://data.dws.informatik.uni-mannheim.de/rdf2vec/code/.
[8] https://github.com/Accenture/AmpliGraph.

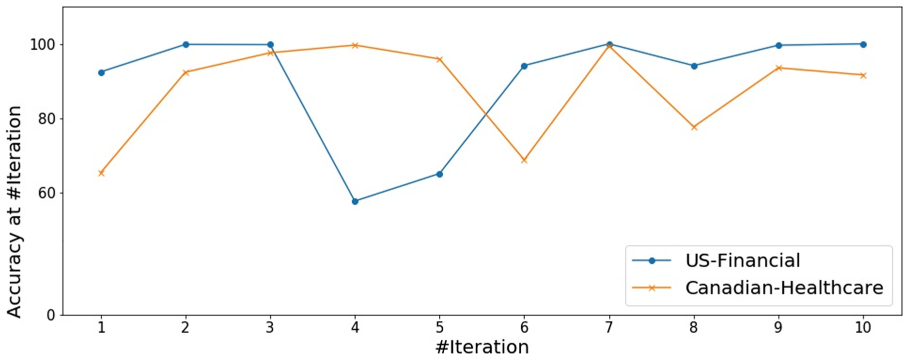


Fig. 2. Accuracy of the string-based CNN per iteration.

negative samples = 25; vector size = 300. For TransE, DistMult, ComplEx and HolE, we learn embeddings with 300 dimensions and maximum of 50 epochs.

The learned embeddings from the 5 approaches are then used to train a machine learning model for type prediction. The output of the embeddings is used to train a multi-label Logistic Regression (LR), multi-label Support Vector Machines (SVM) and a 1-dimensional CNN network. The architecture of the CNN is as follows: two one-dimensional convolutional layers followed by one max-pooling layer and a fully connected sigmoid layer. Each convolutional layer has 100 filters with size 10, using ReLU activation function.

5.4 Human-in-the-Loop Gold Standard Generation

We adapted a human-in-the-loop approach to generate a gold standard in order to evaluate our approaches on the whole data center. We used the dataset from the distant supervision module to bootstrap the labeling process. The labeling process runs in iterations, where in each iteration we (i) train a CNN model using the available labeled data, (ii) apply the model on the unlabeled data, (iii) present the candidates to the SME with confidence above 0.8,[9] and (iv) add the labeled data by the SME in the training set. For the *US-Financial* dataset, we run the process in 10 iterations. In each iteration we present the SME with the top 100, 000 candidates sorted by the model's confidence and calculate the accuracy at each iteration. Note that this step of the pipeline is the equivalent of generating a gold standard dataset, hence the accuracy measure: at each iteration we calculate the percentage of the 100, 000 candidates automatically labeled by the model that are deemed correct by the SME. Figure 2 plots the classification accuracy for both datasets: at each iteration the y-axis shows the percentage of processes correctly labeled by the model and manually validated by the SME. We perform the experiment first on *US-Financial* and then on *Canadian-Healthcare*.

[9] The threshold was selected based on empirical evaluation.

Table 1. Datasets statistics

Dataset	*US-Financial*	*Canadian-Healthcare*
#nodes	2,420	2,139
#processes	1,375,006	339,555
#labeled processes	1,324,759	264,496
#unique labels	153	64

For *US-Financial*, iteration number one represents the model trained on distant supervision data and all subsequent iterations use the feedback from the SME as additional training material. We can notice a relative decrease in performance in iterations 4 and 5. At iteration 4, the SME started validating processes where the classification confidence of the model was lower and identified that the knowledge base did not contain any appropriate tags for them. The SME added 31 additional tags to the knowledge base in iterations 4 and 5, and we can then notice an improvement of performance on the next iterations. At the end of the annotation process, the complete dataset consists of 1,324,759 processes, including those labeled in the bootstrapping step. The SME was unable to validate only 50,247, which is less than 4% of the data, as a result of insufficient information about the processes. Finally, the total number of used tags to label all the processes is 153.

For *Canadian-Healthcare*, the bootstrapping phase is done using the model from *US-Financial*, and then proceed updating the model with the newly labeled (and validated) instances at each subsequent iteration. In the plot (Fig. 2), iteration one of *Canadian-Healthcare* represents results obtained with the previously trained model (on the *US-Financial* dataset): the accuracy is rather low at this first iteration as the overlap with the previous dataset is not very high, but the performance improves with the subsequent interactions with the human. At the end of the annotation process, the complete dataset consists of 264,496 processes. The SME was unable to validate 75,059 as a result of insufficient information about the processes. Tthe total number of used tags to label all the processes is 64. The statistics for both datasets are given in Table 1.

5.5 Results

We evaluate the approaches using the standard measures of precision, recall and F-score. Our datasets are imbalanced: while some of the tags are only used a handful of times, others classify many processes. We therefore weight all the evaluation metrics by the class size. The results are calculated using stratified 20/80 split validation, i.e., we use stratified random 20% of the data to train the model and we test the model on the remaining 80%. We have to note that the results significantly improve when using 10-fold cross validation, however that is not a realistic representation of the problem we are trying to solve, i.e. in our application we are trying to use as little labeled data as possible to automatically label the reset of the dataset. The graph embedding approaches are built on the

Table 2. Results on the *US-Financial* dataset.

Method	Precision	Recall	F-Score
TF-IDF - LR	89.46	73.2	75.85
TF-IDF - SVM	90.11	72.64	76.86
word2vec - CNN	**91.51**	**92.43**	**89.58**
RDF2Vec - LR	47.31	24.58	33.04
RDF2Vec - SVM	43.01	34.95	26.86
RDF2Vec - CNN	72.36	59.17	62.43
TransE - LR	46.98	37.57	28.95
TransE - SVM	41.35	22.86	29.29
TransE - CNN	54.65	32.49	40.64
DistMult - LR	74.97	71.51	71.73
DistMult - SVM	79.39	79.07	79.08
DistMult - CNN	82.44	81.78	82.29
ComplEx - LR	75.86	72.11	72.44
ComplEx - SVM	79.59	79.16	79.19
ComplEx - CNN	84.19	84.97	83.57
HolE - LR	76.51	74.97	75.18
HolE - SVM	78.62	81.36	80.24
HolE - CNN	89.28	88.21	87.93

Table 3. Results on the *Healthcare* dataset.

Method	Precision	Recall	F-Score
TF-IDF LR	94.12	79.06	85.53
TF-IDF SVM	95.30	91.99	92.98
word2vec CNN	**97.38**	**98.17**	**97.63**
RDF2Vec - LR	45.67	21.93	29.51
RDF2Vec - SVM	41.29	32.14	36.04
RDF2Vec - CNN	69.12	57.36	62.57
TransE - LR	12.78	13.01	12.89
TransE - SVM	29.80	17.16	17.62
TransE - CNN	34.48	14.56	4.90
DistMult - LR	43.84	39.59	30.96
DistMult - SVM	41.33	34.92	36.97
DistMult - CNN	47.65	32.49	38.58
ComplEx - LR	75.45	71.48	73.33
ComplEx - SVM	76.35	73.51	74.83
ComplEx - CNN	87.95	79.48	80.17
HolE - LR	76.69	73.31	74.43
HolE - SVM	76.63	72.39	73.39
HolE - CNN	89.46	74.20	81.11

complete graph, using all available information, without the tags of the processes that are part of the test set in the current fold.

Table 2 and Table 3 show a summary of the results for both datasets, and compares figures for each model - Logistic Regression (LR), Support Vector Machine (SVM) or Convolutional Neural Network (CNN) - when using either string-based features (tf-idf/word2vec) or graph embeddings features (RDF2Vec/TransE/DistMult/ComplEx/HolE).

Among the string-based approaches, the CNN with word2vec embedding is the best performer on both datasets. For the graph based approaches, the CNN using HolE embedding outperforms the rest. While DistMult and ComplEx perform comparably to HolE, RDF2vec and TransE show worse performance. On the *Canadian-Healthcare* the TransE model cannot learn a good representation of the graph, which leads to rather bad results. When comparing string-based methods against graph-based approaches, it is important to notice that, despite the fact that word2vec-CNN outperforms HolE-CNN, all the graph-based approaches completely ignore the *process name*, i.e. the text content, and solely use the graph structure to generate the graph embeddings. The result is significant in this particular domain, because much of the data comes without any text content. This means that, while we can successfully use string-based approaches for a portion of the data, we can supplement with graph-based approaches for datasets lacking string descriptions. Finally, one of the major advantages of our

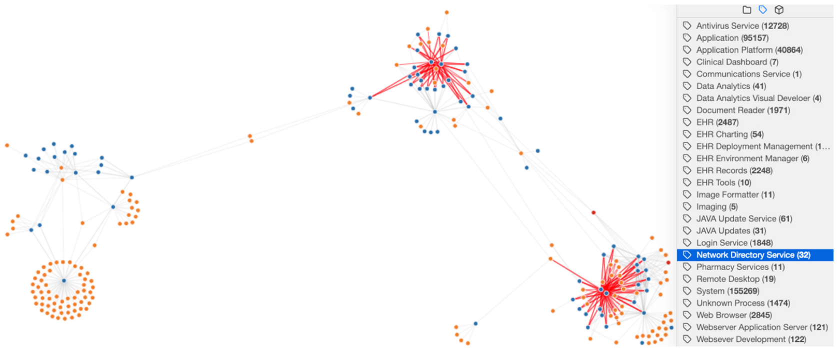


Fig. 3. Data Center Explorer - example view of a portion of the datacenter graph.

methodology was creating the pool of tags and the initial training data in a distant supervision fashion, using LOD as external knowledge. The validation of this initial data took less than a handful of hours for the SME, but enabled a granular understanding of the data center.

6 System Deployment and User Engagements

The machine learning pipeline described in this work is currently deployed on a machine with 100 cores and 1 TB of RAM. The models are continuously updated in the background, and the new results are used to produce data center graphs for any new engagement. To preserve data privacy, for each client engagement we build a separate set of models, trained only on the client's data, without transfer learning from other engagements. Each produced data center graph is used to feed the User Interface used by our SMEs, called *Data Center Explorer* (DCE). The DCE visual tool builds a color-coded representation of all the nodes, as well as their incoming and outgoing connections. We then overlay the produced semantic representation: each node is characterized by all its entities of interest, i.e. the processes running on the machine and their associated tags. An example view produced by DCE is shown in Fig. 3.

We create a knowledge card for each node that summarizes all the information about the node itself, including all the semantic tags associated with all processes running on the node. Moreover, all the knowledge collected in the enriched data center graph can be used to query the data center via GraphQL[10] queries, as well as using the tags as active facets.

DCE is currently used internally by one of the IBM business units working on datacenter migrations engagements with different clients. The team involved in our pilot consists of about 15 practitioners that have the mission of designing and implementing datacenter migration tasks. Since its deployment on our internal cloud, the DCE tool has been used by the team to process 8 large data

[10] https://graphql.org/.

center migrations (over 15000 hosts). The team was previously only relying on manually populated spreadsheets and client interviews, but since the deployment of our pipeline they use our solution to bootstrap any new datacenter migration engagement. From the informal feedback that we gathered from the team, they identified 3 striking benefits of the solution. First, information provided via client interviews is often refuted by DCE reports, with DCE being far more reliable. Second, enormous amounts of time can be saved or redirected to other tasks, as the majority of upfront labor is relegated as unnecessary by DCE's semantic bootstrapping process. Finally, the immediate cost savings provided by DCE (reduced manual labor and fewer migration failures) tends to be between 1 and 2 orders of magnitude higher than the cost to develop, maintain, and train DCE. The major lesson learned for us was the importance of being able to quickly add initial semantic annotations to the nodes of the datacenter, by transparently mining labels from Linked Data, but giving the SMEs full control on which label to add and use in the graph. The combination of automatic semantic bootstrapping with human-in-the-loop achieves the right level of representation to improve the quality and effectiveness of the datacenter migration task.

7 Conclusions and Future Work

Understanding the structure of a data center is crucial for many tasks, including its maintenance, monitoring, migration etc. In this work, we presented a distant supervision approach that bootstraps the annotation of logs using knowledge from the Linked Open Data Cloud. We then train neural models and refine them with a human-in-the-loop methodology. We performed quantitative experiments with two real-world dataset and showed that the approach can classify nodes with high performance.

While in this work we focus on the task of type prediction, in the future we will extend the pipeline to also perform link prediction, i.e. identifying missing links between nodes in the data center, and identifying the type of the relation. Furthermore, using graph embeddings to represent the whole data center and having the whole data into one single feature space opens the possibility for many useful applications, e.g., clustering nodes by type, identifying dependency clusters (clusters of nodes that must be migrated together), ranking nodes and clusters based on importance, and finally building ad-hoc machine learning models for custom data analytic.

References

1. Alba, A., et al.: Task oriented data exploration with human-in-the-loop. A data center migration use case. In: Companion Proceedings of the 2019 World Wide Web Conference, WWW 2019, pp. 610–613. ACM, New York (2019)
2. Awad, M., Menasc, D.A.: Automatic workload characterization using system log analysis. In: Computer Measurement Group Conference (2015)
3. Benson, T., Akella, A., Maltz, D.A.: Network traffic characteristics of data centers in the wild. In: 10th ACM SIGCOMM (2010)

4. Benzadri, Z., Belala, F., Bouanaka, C.: Towards a formal model for cloud computing. In: Lomuscio, A.R., Nepal, S., Patrizi, F., Benatallah, B., Brandić, I. (eds.) ICSOC 2013. LNCS, vol. 8377, pp. 381–393. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-06859-6_34
5. Bernstein, D., Clara, S., Court, N., Bernstein, D.: Using Semantic Web Ontology for Intercloud Directories and Exchanges (2010)
6. Bordes, A., Usunier, N., Garcia-Duran, A., Weston, J., Yakhnenko, O.: Translating embeddings for modeling multi-relational data. In: Advances in Neural Information Processing Systems, pp. 2787–2795 (2013)
7. Bourdeau, R.H., Cheng, B.H.C.: A formal semantics for object model diagrams. IEEE Trans. Softw. Eng. **21**(10), 799–821 (1995)
8. Costabello, L., Pai, S., Van, C.L., McGrath, R., McCarthy, N.: AmpliGraph: a Library for Representation Learning on Knowledge Graphs (2019)
9. Deng, Y., Sarkar, R., Ramasamy, H., Hosn, R., Mahindru, R.: An Ontology-Based Framework for Model-Driven Analysis of Situations in Data Centers (2013)
10. Gentile, A.L., Zhang, Z., Augenstein, I., Ciravegna, F.: Unsupervised wrapper induction using linked data. In: Proceedings of the Seventh International Conference on Knowledge Capture, pp. 41–48 (2013)
11. Grandison, T., Maximilien, E.M., Thorpe, S., Alba, A.: Towards a formal definition of a computing cloud. In: Services. IEEE (2010)
12. Guo, J.: Who limits the resource efficiency of my datacenter: an analysis of Alibaba datacenter traces. In: IWQoS 2019 (2019)
13. Hassan, W.U., Aguse, L., Aguse, N., Bates, A., Moyer, T.: Towards scalable cluster auditing through grammatical inference over provenance graphs. In: Network and Distributed Systems Security Symposium (2018)
14. Jiang, Y., Li, Y., Yang, C., Armstrong, E.M., Huang, T., Moroni, D.: Reconstructing sessions from data discovery and access logs to build a semantic knowledge base for improving data discovery. ISPRS **5**, 54 (2016)
15. Kim, Y.: Convolutional neural networks for sentence classification. In: EMNLP 2014, pp. 1746–1751. ACL, October 2014
16. Lemoudden, M., El Ouahidi, B.: Managing cloud-generated logs using big data technologies. In: WINCOM (2015)
17. Mavlyutov, R., Curino, C., Asipov, B., Cudre-mauroux, P.: Dependency-driven analytics: a compass for uncharted data oceans. In: 8th Biennial Conference on Innovative Data Systems Research (CIDR 2017) (2017)
18. Mintz, M., Bills, S., Snow, R., Jurafsky, D.: Distant supervision for relation extraction without labeled data. In: ACL (2009)
19. Nickel, M., Rosasco, L., Poggio, T.: Holographic embeddings of knowledge graphs. In: Thirtieth AAAI Conference on Artificial Intelligence (2016)
20. Ristoski, P., Rosati, J., Di Noia, T., De Leone, R., Paulheim, H.: RDF2Vec: RDF graph embeddings and their applications. Semant. Web **10**, 1–32 (2018)
21. Shan, Y., Huang, Y., Chen, Y., Zhang, Y.: LegoOS: a disseminated, distributed {OS} for hardware resource disaggregation. In: 13th Symposium on Operating Systems Design and Implementation, pp. 69–87 (2018)
22. Trouillon, T., Welbl, J., Riedel, S., Gaussier, É., Bouchard, G.: Complex embeddings for simple link prediction. In: ICML (2016)
23. Wu, H., et al.: Aladdin: optimized maximum flow management for shared production clusters. In: 2019 IEEE PDPS, pp. 696–707. IEEE (2019)
24. Yang, B., Yih, W., He, X., Gao, J., Deng, L.: Embedding entities and relations for learning and inference in knowledge bases. arXiv preprint arXiv:1412.6575 (2014)