Transferable Knowledge for Low-cost Decision Making in Cloud Environments

Faiza Samreen, Gordon S Blair, Yehia Elkhatib

Abstract—Users of Infrastructure as a Service (IaaS) are increasingly overwhelmed with the wide range of providers and services offered by each provider. As such, many users select services based on description alone. An emerging alternative is to use a decision support system (DSS), which typically relies on gaining insights from observational data in order to assist a customer in making decisions regarding optimal deployment of cloud applications. The primary activity of such systems is the generation of a prediction model (*e.g.* using machine learning), which requires a significantly large amount of training data. However, considering the varying architectures of applications, cloud providers, and cloud offerings, this activity is not sustainable as it incurs additional time and cost to collect data to train the models. We overcome this through developing a Transfer Learning (TL) approach where knowledge (in the form of a prediction model and associated data set) gained from running an application on a particular IaaS is transferred in order to substantially reduce the overhead of building new models for the performance of new applications and/or cloud infrastructures. In this paper, we present our approach and evaluate it through extensive experimentation involving three real world applications over two major public cloud providers, namely Amazon and Google. Our evaluation shows that our novel two-mode TL scheme increases overall efficiency with a factor of 60% reduction in the time and cost of generating a new prediction model. We test this under a number of cross-application and cross-cloud scenarios.

Index Terms—Cloud computing, Decision support, Machine learning, Transfer learning.

1 Introduction

The cloud computing market has a proliferation of service offerings, pricing models, and technology standards [1], [2]. This complicates decision making regarding service selection. Although such challenges apply to all levels of cloud services, the Infrastructure as a Service (IaaS) level is particularly difficult given the fact that IaaS provides more choices and control for developers. In the IaaS domain, there is a wide range of virtual machines (VM) on offer – see Figure 1 – but no straightforward method to compare their performance and, more generally, cost/performance trade-offs, neither *within* nor *across* cloud providers. A wrong or suboptimal decision can result in financial loss as well as reduced application performance [3], [4], a common concern of end users [5], [6].

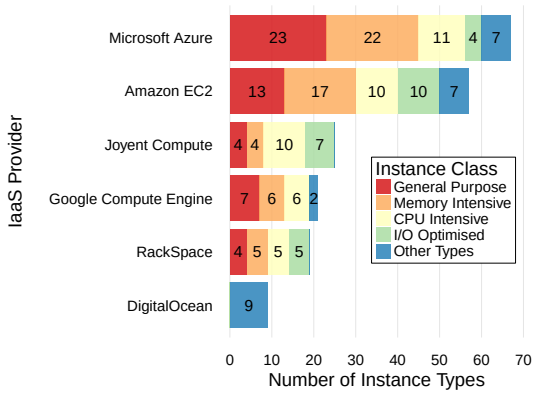

Fig. 1. On-demand instance types (Linux) offered by major IaaS vendors.

The box plots in Figure 2 represent the distributional spread of execution times of 3 different applications across a variety of instance types of Amazon EC2 and Google GCE. Throughout this paper we have used the same set of applications; their architectural details are provided in Section 4.2.

Figure 2(a) illustrates the performance variability in executing a memory-intensive application (VARD) on different instance types of variable computational capacity. Contrary to both Figure 2(b) and Figure 2(c), the T2 series is shown as the most predictable with less variability in terms of performance. In comparison with GCE, `n1s1` shows less variance in performance and offers the best execution time compared to all GCE nodes as well as a most expensive EC2 node (`c4.xlarge`). On top of that, `n1S1` ($0.036/h) and T2 instances ($0.026-$0.052/h) are the cheapest of all. A striking observation is the poor performance of `m3.medium`, as shown across all plots in Figure 2.

For a CPU-intensive application (smallpt), `c4.xlarge` ($0.232/h), `c3.xlarge` ($0.239/h) and `m3.xlarge` ($0.280/h) with variable prices are performing at a very similar level – Figure 2(b). Moreover, GCE's `n1cpu4` and `n1s4` show similar performance to equivalent EC2 instances, and `n1cpu8` ($0.215/h) is performing better than the expensive and predictable performing EC2 `m3.xlarge` ($0.280/h).

Figure 2(c), of a CPU-intensive application (ItemRecommender), reflects the least performance variability and seems more predictable for GCE instances. Moreover, reflects the same trend for the T2 series as seen in Figure 2(b). A consistent performance with a negligible difference in median values is observed among all GCE instances. The same pattern is observed across EC2 instances except the T2 series showing the highest execution time across all other nodes. In contrast to the results in Figure 2(b), `c4.large` ($0.116/h)is performing better than `c3.xlarge` at a much lower cost. In other words, the selection of a more expensive option does not lead to improved performance in terms of execution time.

The following observations can be drawn from the above discussion. The selection of an instance type based on descriptive information cannot guarantee the best fit in terms of cost and performance trade-offs. The distributional spread of results highlights the intrinsic uncertainty related to instance performance, contributing to the complexity regarding the selection of suitable instance. There are some external factors that can influence this performance such as the scheduling policy, machine age,

- *This work was supported by the Adaptive Brokerage for the Cloud (ABC) project, UK EPSRC grant EP/R010889/1.* *All authors are with the School of Computing and Communications, Lancaster University, United Kingdom. Samreen is also with Sheffield Hallam University, United Kingdom. E-mail: {i.lastname}@lancaster.ac.uk*

Manuscript received November 28, 2018 and revised December 16 , 2019

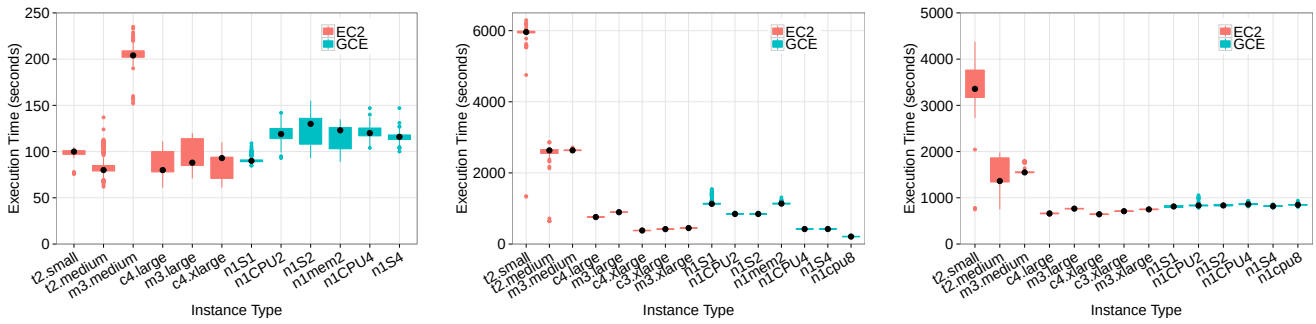

Fig. 2. The execution times on various VM instances of AWS and Google of: (a) a memory-intensive application (VARD); (b) a CPU-intensive application (smallpt); and (c) another CPU-intensive application (ItemRecommender). Instances from each provider are sorted from left to right in increasing instance hourly cost.

provider-specific incentives etc. Therefore, nodes with similar capacity/specifications are difficult to compare.

Classified categories of instance types do not reflect the real performance behaviour of any application due to different architectures and the unique workload patterns of that application. Similarly, superficial knowledge about the application cannot guarantee the expected performance over different instance types.

In such decision-making situations, machine learning (ML) has potential to underpin intelligent and data-grounded Decision Support Systems (DSS). Indeed, ML-aided DSS have been demonstrated to provide behavioural and performance insights about application and deployment setup necessary to make optimal decisions, *e.g.* [4], [7], [8], [9]. A traditional ML approach follows the general steps of: collecting data, generating a learning model, fitting the model on training data, and assessing its accuracy on test data. This is a well established, albeit lengthy, process. However, a common assumption in this traditional learning setting is that the test and training data sets are drawn from the same distribution (say, performance of a certain cloud provider's VMs using a particular application). If the distribution changes, such assumption no longer holds and, consequently, there needs to be a lengthy process of rebuilding the model. In our context, this means commissioning a new set of benchmarks that costs time and money.

This presents a significant challenge for ML-based decision support in multi-cloud environments, where different architectures and varying deployment setups across different cloud providers lead to significant changes in data distribution as well as feature space. Therefore, in a real world scenario, conducting such experiments is time-consuming and requires considerable cost and time for data collection and model generation. This paper focuses on enhancing the learning efficiency of ML-aided DSS to assist application-specific decisions across different cloud providers. In this paper, we propose a novel Transfer Learning (TL) assisted scheme leading to substantial reduction in the training overhead by making use of existing knowledge. Quantitatively, we observed a reduction of approximately 60% in learning time and cost by transferring existing knowledge about an application and/or a cloud platform in order to learn a new prediction model for another application or cloud provider. The proposed scheme has the potential to deal with the challenge of model generation when data distribution or feature spaces differ between source and target domains.

Our main contributions are as follows:

We design and implement `Tamakkon`, a two-mode TL scheme that identifies the type of knowledge to be transferred across domains. We utilize a subset of well-established machine learning approaches, namely Multiple Polynomial Regression (MPR), Support Vector Regression (SVR) and Random Forests (RF). We believe this to be a novel application of TL for decision support in cloud computing.

We present a methodology to measure similarity among different data sets in order to identify the source application and its learned data that can be used as a source of knowledge for generating a prediction model for a target application, and, additionally, to avoid negative knowledge transfer.

Through extensive experimentation, we apply `Tamakkon` on different applications *and* cloud providers. We efficiently generate a learning model between varying application and IaaS domains, reducing the learning cost (of cloud resources and time) by 60%.

`Tamakkon` has many unique features making it different from other similar research [9], [10], [11]. First of all, `Tamakkon` is based on a transfer learning approach and makes use of existing models and data to reduce the cost and time involved for model generation and large-scale data collection. Secondly, `Tamakkon` works across different applications as well as cloud providers. Furthermore, a range of models are offered, resulting in reduced prediction errors. Lastly, the `Tamakkon` framework is extensible: new models representing different variations of workloads can be added as base-learners, if needed.

2 Background on Transfer Learning (TL)

The general goal of TL is to use previously learned knowledge to solve a problem in an unseen environment or in a faster or better way. TL extracts knowledge from one or more source domains and source tasks, and applies that knowledge to achieve a task in a target domain. Given a source domain D_S and source task T_S, a target domain D_T and target task T_T, TL aims to improve the prediction function of D_T using the knowledge in D_S and T_S considering that D_S has some similarity with D_T and $T_S = T_T$ or $T_S \neq T_T$. This process of achieving the target task by learning from the source domain and source task is illustrated in Figure 3.

TL is categorized into two sub-types: *inductive TL* aims to improve the learning of a target predictive function with the help of a source task, considering that the source and target domains are the same but the tasks differ; *transductive TL* aims to improve the learning of a target predictive function with the help of knowledge from a source domain, where source and target domains may or may not be different but the tasks are the same.

The literature offers different approaches of transferring knowledge between domains [12]. The instance knowledge transfer approach is applied by re-weighting some portion of source data to be used in the target domain and iteratively measuring model fitness for target task learning [13], [14]. The

Fig. 3. An overview of the TL process.

feature representation knowledge transfer approach requires identification of good features that can reduce the difference between source and target domains in order to minimize model error and domain divergence [15], [16]. The parameter knowledge transfer approach is applied with an assumption that the source and target tasks share some parameters or prior distributions of the model hyper-parameters [17], [18]. The relational knowledge transfer approach deals with TL problems in the relational domain. TL has been applied to many applications such as classification of images, verbal sentiment, and software defects.

TL for regression tasks is a less researched area, however, some notable efforts have been made, e.g.[19], [20], [21].

3 Tamakkon : A TL-ASSISTED DSS

Our goal is to enhance the efficiency of intelligent DSSs in order to make data-driven and application-specific decisions in cross-cloud environments, i.e. those spanning more than a single provider. We believe that efficiency can be achieved by reducing the training overhead through making use of existing knowledge in the form of experiential data sets, significant predictive features, and prediction models and their parameters. We devise a TL-based approach to assist in efficient model generation by transferring learned knowledge from a related domain. The proposed approach is designed specifically for deployment and migration decisions at the IaaS level.

Our approach is a semi-supervised transductive TL technique that allows the contribution of auxiliary target data for model generation. Semi-supervised learning is used due to its ability to learn with a little amount of labeled data, reducing the required amount of training data for the target domain, which is one of the key concerns of model generation efficiency.

We now present the design and implementation of our solution (x3.1), and the algorithms underpinning the TL scheme (x3.2–3.3).

3.1 System architecture

The main components of the overall system architecture, shown in Figure 4, are a Knowledgebase and three phases: Analysis , Learning and Planning . The Knowledgebase is comprised of learning methods, prediction models, model parameters along with the data sets the models are derived from. The prediction models present in the Knowledgebase are generated by following the traditional ML process (i.e. train and test using cross-validation) (x3.1.2). The key idea of Tamakkon is to efficiently generate a prediction model for a given application and the process starts with a collection of auxiliary data (x3.1.3). The Analysis Phase is responsible for providing auxiliary data to be used by the Learning phase. The auxiliary data is collected by running the application on a representative set of virtual machines and profiling different matrices related to the deployment and performance of the application. The Learning Phase is the core of the architecture and is composed of four sub-modules, each performing their own unique tasks to support model generation for a target domain (i.e. application and/or provider). The model, however, is not generated from scratch by following the long steps of model fitting, but rather it is generated using Tamakkon which makes use of existing learned knowledge fetched from the Knowledgebase. Similarity Measure helps in identifying the similarity of a new domain (target) with existing ones (source). The auxiliary data is then used by the similarity measure (x3.2) to look for a similar application in the Knowledgebase. Based on this similarity, Tamakkon transfers existing knowledge using two-mode TL scheme , which is designed to satisfy the goal of enhancing learning efficiency and generating a prediction model for a new application. The `two mode' part of the naming refers to the two possibilities of transferring learned knowledge across domains: Transfer-All and Transfer-Model These modes present different means of transferring significant knowledge through the transfer of instance knowledge, feature space knowledge, and parameter knowledge (x3.3).

Fig. 4. The architecture of Tamakkon , a novel TL-assisted cloud DSS.

Together with the similarity outcome and a selected base-learner (x3.1.1), one of the knowledge transfer modes gets activated. The activated mode feeds in the auxiliary data as well as learned data to the base-learner which then outputs a prediction model. Model Training uses training data sets which may be composed of source and target domain data depending on the activated mode. The test data set for model assessment is comprised of the target domain's data only. Model assessment is performed using 10-fold cross-validation and, if the output is not satisfactory, the process re-starts by fetching the next most similar application from the Knowledgebase. Finally, the accepted model is saved in the Function Repository to be used by the Planning Phase to predict future application performance. Deployment costs are generated from these predictions to find the best deployment match in accordance with customer-specified constraints.

The remainder of this section details the functionality of the Tamakkon modules.

3.1.1 Base learners

Base learners are the ML methods used to derive the initial prediction models for a source task. For Tamakkon , Multivariate Polynomial Regression (MPR), Support Vector Regression (SVR) and Random Forests (RF) are used as base-learners. MPR has the ability to statistically infer and understand the relationship of response and predictor variables related to a data set. Hence, a behavioural relationship for an application can be generated with

varying deployment setups. SVR attempts to obtain a function as flat as possible having an -deviation from the training output. SVR relies on a set of parameters such as: the sensitivity to the deviation, the cost function and range of kernels, allowing it to optimize the objective function as well as the parameters of the regressive function. This allows Tamakkon to optimally adjust the hyper-parameter when fed with the different distributional training sets. RF [22] is based on an ensemble learning approach making use of Bagging sampling and the random selection of features. The ensemble nature of RF enhances its ability to produce better prediction results. In RF, multiple decision trees (ensemble models) are generated using the randomly drawn samples. For unseen data, each decision tree takes part in the prediction process and the final prediction result is the label selected by a majority of decision trees. RF is good in accuracy and gives a better understanding of the variable importance and their correlations. Moreover, this method is considered robust to outliers as compared to other methods.

3.1.2 Base models

Base models are generated by following what is now deemed a traditional ML process of learning a model starting from data collection as detailed in Figure 5. This model generation is supported by Daleel's framework [4], designed to investigate the role of ML to support application-driven decision making around the selection of instance types in a given cloud provider.

Fig. 5. The model generation process.

The collected data passes through the pre-processing stage to fulfill initial assumptions about the quality of the data. Pre-processed data is then split into training and test sets and fed into an iterative process of fitting a model. The model fitting process is responsible for exploring different distributions to investigate the true relationship of a response variable with the predictors (features). The fitted model is then trained on the training data set using 10-fold cross-validation to capture the maximum variability of data and later passed on to the model assessment process to be assessed for the prediction accuracy. Further to that, the models offering satisfactory prediction capability are then stored in the function repository and, in case of unsatisfactory results, starts updating the model.

The collected data traces are comprised of application-specific, cloud-deployment and execution-related features. The application-specific features are: Application-Type (X1), Multithreading (X2), External-File-Requirement (X3), Load-in-Memory (X4), Parallel-Execution (X5) and File-Size (X6). The cloud-deployment and execution-related features are: VM-Type (X7), RAM (X8), Compute-Units-ECU/GCEU (X9), Virtual-CPU (X10), Application-Submission-Day-of-Week (X11), Application-Submission-Time (X12) and Application-Execution-Time (X13). The complete set of features are used for similarity measurement and the base models are composed of different subsets of these features.

3.1.3 Auxiliary data

Semi-supervised transductive TL, which Tamakkon is based on, requires the presence of auxiliary data for model generation. In this context, the term, “sufficient” amount of data is introduced to represent the auxiliary data requirement for the target domain to be used in Tamakkon . This is an important parameter for the approach, derived from extensive experimentation. A “sufficient amount” is identified by observing model convergence according to the change in percentage contribution of training data.

3.2 Similarity measurement

A similarity measure is required to identify which of the source domains (source applications) will give the best performance on the target domains (target applications) to learn the target task (prediction model). The proposed approach leverages the Kolmogorov-Smirnov (KS) test, a non-parametric test to compare the cumulative distributions of two data sets in order to identify if the two data samples come from the same distribution.

The KS test is considered effective in comparing samples when there is no available knowledge about the common distribution of the source and target domain data. In addition, this method is sensitive to distribution and there is no restriction on the sample size which makes it useful to capture change in performance even with few samples. These properties make Tamakkon useful for different applications and machine configurations considering that often it is difficult to compare deployment settings and resource utilization between the two applications, especially when the VMs belong to different cloud providers and vary in underlying configurations.

Algorithm 1 compares the probability distribution of the target application's profiling data (S_{aux}) with the existing applications (S_{kbi}) and identifies similar distributional application(s) (S_{kb_i} :tagged) from the knowledgebase. The KS/similarity test is applied on vector inputs ($A_{j\,1};:::;A_{jk}$ & $B_{l\,1};::;B_{lk}$) from both source and target domain, and each vector input represents a single feature. This test outputs a p-value (D_p) and a D-value (D_d). Here p-value quantifies the probability of two samples populated from the same or a different distribution and D-value represents the difference of empirical distribution functions of two samples. Table 1 presents the outcome of a similarity analysis based on three representative applications, as detailed in Section 4.2. The first 6 features in the table represent application architecture and the remaining features explain cloud and deployment related information, as explained in Section 3.1.2. The three applications are represented as A (VARD), B (smallpt) and C (ItemRecommender). The D-value and p-value is calculated for each feature vector taken from the source and target applications as explained in the SimilarityMeasure function. The similar features based upon p-value are marked as “**” and the closeness of distribution is marked as “*”. If the p-value rejects the similarity hypothesis, then the D value is evaluated to get an idea about the probability of similarity. A D-value in the range of 0.0–0.5 is considered as a measure of corresponding sample similarity from 100% to 50%. The aggregated p-values and D-values assist in deciding the similarity. An application with a high number of similar features will get a higher ranking in terms of similarity. The resulting similarity is tagged as one of three categories: 1) Similar (all the features are same), 2) Partly-Similar (> = half of application-specific AND cloud, deployment-specific features are same), or 3) Not-Similar (> = half of application-specific OR cloud, deployment-specific features are different).

The aggregated similar features for A and C is higher than A and B so, for application A being a target application, C would be the first choice to be used as a source of knowledge. Moreover, applications A and C have similar architectures and are tagged as Partly-Similar to each other, however application B is tagged as Not-Similar. For application B both the applications A and C are ranked the same to be used as the source, however both

Algorithm 1: Identification of source domain similarity

Input: (i) Auxiliary data = S_{aux}
(ii) Knowledgebase data = S_{kb_i},
where $i = 1 :: :$total number of applications
Output: (i) D_d, D_p, D_f : a similarity estimate for S_{kb_i}
(ii) Tagged knowledgebase datasets = S_{kb_i} :tagged
Initialization;
  Let $A_{j1};:::;A_{jq};:::;A_{jk}$ be the value
  in S_{aux}, where $A_{j1} :: A_{jq}$ represents application architecture,
    and $A_{jq+1} :: A_{jk}$ represents deployment details
  Let $B_{l1};:::;B_{lq};:::;B_{lk}$ be the value
  in S_{kb_i}, where $B_{l1} :: B_{lq}$ represents application architecture,
    and $B_{lq+1} :: B_{lk}$ represents deployment details
Function SimilarityMeasure (S_{aux}, S_{kb_i})
foreach $S_{kb_i} 2 f S_{kb_1};:::;S_{kb_n} g$ do
  foreach $x 2 f A_{j1};:::;A_{jk} g[f B_{l1};:::;B_{lk} g$ do
    for Two:Sample:KS:Test(x_a;x_b) do
      Compute p-value ! D_p
      Compute D^0-value ! D_d
      if p-value (D_p) > 0.05 then
        x:mark = \ SAME "
        x_p:mark = \ SAME "
        D_f = \ NO"
      else
        if $D^0(D_d) < 0.5$ then
          x:mark = \ SAME "
          D_f = \ NO"
        else
          x:mark = \ DIFFERENT "
          D_f = \ YES"
  aggregate:x:mark for $f A_{j1} :: A_{jq} g$ and $f A_{jq+1} :: A_{jk} g$
  aggregate:x_p:mark for $f A_{j1} :: A_{jq} g$ and $f A_{jq+!} :: A_{jk} g$
  if value of
  aggregate:x_p:mark is "SAME" for all $f A_{j1} :: A_{jk} g$ then
    S_{kb_i}:tagged = \ SIMILAR "
  else
    if value of aggregate:x_p:mark is "SAME" for >=
    1/2 of $f A_{j1} :: A_{jq} g$ AND value of aggregate:x:mark
    is "SAME" for >= 1/2 of $f A_{jq+1} :: A_{jk} g$ then
      S_{kb_i}:tagged = \ PARTLY SIMILAR "
    else
      S_{kb_i}:tagged = \ NOT SIMILAR "

applications have almost no similarity at application architecture level and hence will be tagged as Not-Similar to each other.

3.3 The Two-mode TL scheme

This section explains the core TL algorithm, derived from extensive experimental analysis on public cloud providers and applications of varying requirements. The two modes and the respective knowledge transfer approaches are discussed here:

1) Transfer-All includes three approaches to transfer knowledge from the source to the target domain.
   a) Transferring feature representation knowledge
   b) Transferring instance knowledge
   c) Transferring parameter knowledge
2) Transfer-Model includes two approaches to transfer knowledge from source to target domain.
   a) Transferring feature representation knowledge
   b) Transferring parameter knowledge

Tamakkon works by activating one of its modes based on the inputs from similarity measurement and a base-learner. If the similarity outcome select a "Similar" or "Partly-Similar" application as a source of knowledge and if the learning method is SVR or RF then Transfer-All is activated and the respective knowledge is transferred from the source application to the target application. On the other hand, if there is a "Not-similar" application and the learning method is still SVR or RF, Transfer-Model gets activated. In case of the MPR learning method, the only possibility for activation mode is Transfer-Model. We now explain each of the above approaches involved in the designed scheme.

TABLE 1
Pairwise similarity analysis among three applications: VARD (A), smallpt (B), and C (ItemRecommender). Similarity is indicated by p- and D-values, which are calculated using the KS method for application, cloud and deployment specific features.

	A & B		A & C		C & B	
Feature	D-value	p-value	D-value	p-value	D-value	p-value
app.type (X1)	1	2.20E-16	1	2.20E-16	0	1 **
multithreading (X2)	1	2.20E-16	0	1 **	1	2.56E-07
Ext.file.req. (X3)	1	2.20E-16	0	1 **	1	2.20E-16
load.in.mem (X4)	1	2.20E-16	1	2.20E-16	0	1 **
parallel exec. (X5)	1	2.20E-16	0	1 **	1	2.20E-16
file size (X6)	1	2.20E-16	1	2.20E-16	1	2.20E-16
vm type (X7)	0.3986 *	2.20E-16	0.4081 *	2.20E-16	0.0662 *	2.56E-07
ram (X8)	0.3760 *	2.20E-16	0.3939 *	2.20E-16	0.0178 *	0.6086
comp.units (X9)	0.4309 *	2.20E-16	0.4081 *	2.20E-16	0.0662 *	2.56E-07
vcpu (X10)	0.4309 *	2.20E-16	0.4035 *	2.20E-16	0.0662 *	2.56E-07
app.sub.day (X11)	0.0925 *	2.32E-16	0.0803 *	2.20E-16	0.1729 *	2.56E-07
app.sub.time (X12)	0.0148 *	2.20E-16	0.0178 *	0.6258	0.0161 *	0.7317
exec.time (X13)	1	2.20E-16	1	2.20E-16	0.6329	2.20E-16
Result	Not similar		Partly similar		Not similar	

(a) Transferring feature representation knowledge

The feature space represents specific properties related to application architecture, deployment configurations and execution details. If both the source and target domains have some similarity at the application or deployment level, the higher the chances are for generating a reasonable target-learning model using the same features as for the source model. Therefore, `significant' features are transferred from the source domain to the target domain. If the feature space differs in both source and target domains due to the varying standards of IaaS offerings, mapping of similar features is required. The mapping is done manually using the shared knowledge of both domains.

(b) Transferring instance knowledge

The instance knowledge represents a sample set comprised of the selected feature space. If the source and target domains have high similarity then the instance knowledge transfer can positively contribute for the model generation of the target domain, especially when there are few sample sets available for the target domain. However, the best practice is to transfer instances in an incremental manner in order to avoid any influential effect from the source data.

(c) Transferring parameter knowledge

Parameter knowledge details the mathematical formulation of the estimation or prediction function. The parameter knowledge can be shared with the target domain in accordance with the selected base-learner. If the base-learner is SVR then the kernel function, tuning parameters' learning rates, the learning-cost function and model constants are transferred. For MPR, the polynomial order along with coefficient values and any interaction terms are transferred. In the case of RF, information about the number of decision trees to be generated and a number of variables to be tried at each step are shared with the target domain.

Fig. 6. A high-level description of the three evaluation scenarios.

4 Experimental Evaluation

Tamakkon is evaluated using two public cloud providers with three representative real-world applications that differ in their underlying architectures and also their memory and CPU utilization.

4.1 Evaluation strategy

Our evaluation criteria are as follows:

1) Accuracy of base models , i.e. the traditional ML approach, in terms of prediction performance using overall data and using daily subsets thereof. A poor prediction model can lead to the selection of worst deployment option.
2) Feasibility of our TL approach under 3 distinct different evaluation scenarios: transferring to a new application, to a new provider, and to both a new application and provider, respectively (Figure 6).
3) Blind knowledge transfer examples are illustrated to show the benefits and impact of Tamakkon similarity analysis along with the associated two-mode transfer learning scheme.
4) Time and cost effectiveness of the proposed scheme in terms of incurred cloud costs and model generation time.

All application parameters and input are kept constant between runs to reduce the dimensionality of the experiments. In order to collect enough data for learning purposes, we repeat a workload with a 10 minute delay between each pair of runs. This is spread over different times of the day and different days of the week to capture temporal variance. The Linux tools vmstat , glances and sysstat are used to continuously monitor resource utilization.

Different evaluation metrics were computed to assess the model accuracy, namely R^2, Residual Standard Error (RSE), Mean Squared Error (MSE) and Percentage Relative Root Mean Squared Error (%RRMSE). The metrics presented in this paper are MSE and %RRMSE. MSE is a standard measure of prediction error and we use it to assess individual model performance. %RRMSE provides a fair comparison of prediction accuracy between different learning models and data sets with variable scales by calculating a scale-independent model error relative to the actual value. As such, we use it to evaluate the performance of models for different data sets. %RRMSE is expressed as:

$$\%RRMSE = \sqrt{\frac{1}{N}\sum_{t=1}^{N}\left(\frac{y^i - y}{y}\right)^2}\ 100 \qquad (1)$$

where N is the total number of samples, and y^i and y are the predicted and actual value of application execution time, respectively. The lower the %RRMSE value is the better the model performance.

4.2 Applications

We selected 3 open-source applications of different architectures and requirements of CPU and memory usage.

1) VARD is a tool designed to detect and tag spelling variations in historical texts, particularly in Early Modern English [23]. The output is aimed at improving the accuracy of other corpus analysis solutions. VARD is a single threaded Java application that is highly memory intensive. It holds in memory a representation of the full text, as well as various dictionaries that are used for normalizing spelling variations. VARD is considered a pre-processor tool to other corpus linguistic tools such as keyword analysis, semantic tagging and annotations etc.
2) smallpt is a global illumination renderer written as a multi-threaded OpenMP application and, as such, is considered CPU intensive. The application performs unbiased Monte Carlo path tracing to calculate the amount of light reaching different points in a given scene, offering features such as anti-aliasing, soft shadows, and ray-sphere intersection. For this research we selected a box scene that is constructed out of nine very large overlapping spheres.
3) ItemRecommender uses a collaborative filtering technique in order to recommend similar items, as identified using log-likelihood similarity. ItemRecommender is written as a single-threaded Java based program, and uses the MovieLens[1] dataset collected and maintained by GroupLens. The dataset is comprised of 10 million ratings of 10,000 movies by 72,000 users. This is used to recommend movies to a user based on the preferences of other users. The program takes every other person who has reviewed at least 5 movies in common with the user and calculates the Pearson correlation between these 2 users. Accordingly, weights are calculated and converted to numerical values to be assigned to the user's scale and finally the recommendations are listed after sorting.

TABLE 2
Computational specifications of EC2 instances.

Series	Instance Type	vCPU	ECU	RAM (GiB)	Storage (GB)	Price ($/h)
T2 (General Purpose)	t2.small	1	Var.	2	20	0.026
	t2.medium	2	Var.	4	20	0.052
M3 (General Purpose)	m3.medium	1	3	3.75	4(S)	0.070
	m3.large	2	6.5	7.5	32(S)	0.140
	m3.xlarge	4	13	15	32(S)	0.280
C4 (Compute Optimised)	c4.large	2	8	3.75	20	0.116
	c4.xlarge	4	16	7.5	20	0.232
	c3.xlarge	4	14	7.5	32(S)	0.239

TABLE 3
Computational specifications of GCE instances.

Series	Instance Type	vCPU	GCEU	RAM (GB)	Storage (GB)	Price ($/h)
Standard Type	n1-standard-1	1	2.75	3.75	16	0.036
	n1-standard-2	2	5.5	7.5	16	0.071
	n1-standard-4	4	11	15	16	0.142
High Mem.	n1-highmem-2	2	5.5	13	16	0.106
High CPU	n1-highcpu-2	2	5.5	1.8	16	0.056
	n1-highcpu-4	4	11	3.6	16	0.118
	n1-highcpu-8	8	2.2	7.2	16	0.215

4.3 Cloud infrastructure

For our target cloud providers we use Amazon EC2 and Google GCE, two of the major providers in the IaaS market. From each, we selected a subset of their on-demand (i.e. pay-as-you-go) Linux instance types deemed suitable to run the aforementioned applications. These instance types are summarized in Tables 2-3.

Both Amazon EC2 and GCE provide a differentiated series of instance types, catering to different application needs (e.g.compute-intensive, memory intensive, I/O-intensive, and so on). Each series

1. https://grouplens.org/datasets/movielens

contains a number of instance types offering different setups of computational resources. From EC2 we targeted the General Purpose T2 and M3 series. In addition, the Compute Optimized series is also selected with three representative instance types. Similarly, from GCE we targeted the Standard Type series instances. In addition, we selected the High CPU series as well as the High Memory series in order to evaluate varying combinations of resource capacities over a relatively wide price range. All instances used run 64-bit Ubuntu Linux of different capacities as shown in Tables 2–3. GCE differs from Amazon-EC2 in various aspects such as the pricing scheme, VM configuration measurement units and compute units. Amazon charges on an hourly basis for VMs; in contrast, Google charges a minimum of 10 minutes per VM. Both providers have non-standard categories to offer the pool of VM's computational power and units. The Google compute engine unit (GCEU) [2] is an abstraction of compute resources. According to Google, 2.75 GCEUs represent the minimum power of one logical core. Amazon uses the term `ECU' as a computation unit to express the CPU capabilities of its various compute offerings. The capacity unit for measuring the disk size, machine type memory, and network usage are calculated in gigabytes (GB) for each EC2 instance of Amazon. In contrast, GCE uses gibibyte (GiB) as a measuring unit for describing configurations of the VMs. It is very hard to make a 1:1 comparison with such non-standard and sometimes vague descriptions about the computational units. This creates a difference in terms of the feature space at the domain level. Moreover, some of the information, such as details of parallel workload on VMs, scheduling algorithms and how GCE virtual cores are pinned to physical cores, is not provided by the public cloud providers. So, users of IaaS cannot perceive any collocation or interference effect on their running application. The proposed approach transcends these problems and deals with such differences at the feature space level by mapping of similar features.

5 Results

5.1 Overall accuracy of base models

Fig. 7. %RRMSE comparison of Linear, MPR, SVR and RF based models generated for VARD, smallpt & ItemReccomender.

Base models are generated using a traditional ML approach as explained in Section 3.1.2 and a range of ML methods are involved in this process. The generated models for each application are compared with each other in order to assess the prediction accuracy. Figure 7 displays the %RRMSE of multiple prediction models for three applications as bar plots. For VARD, the prediction errors for MPR, SVR and RF-based models are 13.300%, 14.459%, and 13.472%, respectively, a little higher compared to smallpt and ItemRecomender specific models. MPR, SVR and RF-based models for smallpt are the ones with the lowest model error across (< 0.41%). Similarly, for ItemRecomender, the prediction errors of MPR, SVR and RF-based models are of very similar error values, all being quite low (< 0.72%). The low %RRMSE values for these generated models indicate considerably reduced prediction error. In contrast, the linear model has a slightly higher error rate for VARD and ItemRecommender and a much higher one for smallpt, indicating that this is a poor model in terms of prediction accuracy.

We now explore the variance of model performance on a daily basis to assess the amount of sufficient data needed for accurate predictions. MPR, SVR, and RF based models are evaluated to assess per day prediction accuracy for three applications, as shown in Figures 8–10. The box plots show the spread of %RRMSE values (y-axis) for 7 days of data where each entry is a result of 10-fold cross-validation.

The left plot in Figure 8 displays the prediction error of the MPR-based model for VARD with a minor variability at a day level. The error becomes more consistent with the addition of more data samples for each day and the majority of data samples are gathered around the third quartile and the first quartile for day1 and day2, respectively. The variability at distributional level gives an indication of performance variation at the VM level, investigated in more detail below. The left plot in Figure 9 represents the model error of the MPR-based model for smallpt and surprisingly a lowermost error rate is observed with very steady predictions for each day. The MPR-based model for the ItemRecommender in Figure 10 shows very low error with consistent prediction results for each day where most of the data samples are lying around the median. The low error rates confirm that the model is able to capture maximum data variability with accurate prediction for the majority of data samples.

Similar to MPR-VARD, SVR-VARD has the same variable pattern for model error, as shown in the middle plots of Figure 8. Contrary to RF-VARD, the error spread in SVR-VARD is narrow and the majority of data samples are gathered around the median. The plot for smallpt as shown in Figure 9 displays the highest model error (> 9) for day 1 with this gradually decreasing down to less than 1 by day 3 and then this is consistent until day 7. Finally, the model error for the SVR-based model for ItemRecommender (middle plot in Figure 10) is almost the same as for the MPR based model, making it more consistent for prediction accuracy. On the other hand, RF-based models are displaying a common pattern in all boxplots, as shown on the right in Figures 8–10. A big drop in the model error is observed from day 1 to day 2 and minor drops from day 2 onwards. An interesting observation is the variability pattern for RF-VARD which is similar to MPR-VARD and providing further evidence of our assumption about significant performance fluctuation. The right plot in Figure 9 shows a contrasting behaviour (RF-smallpt) compared to MPR-smallpt. The error values are spread across different quartiles on a wide range for day1 and %RRMSE keeps reducing until day 3 and then the median is consistently at the same level for the rest of the days. For the RF-based model for ItemRecommender, the right plot in Figure 10, we observe a consistent decline for %RRMSE with a narrow spread at quartile ranges and the median values are nearly aligned with the MPR-based model.

All three base-learners are showing similar results with respect to different applications and presenting comparable prediction performance. Overall, MPR is converging to a lower error rate with just a day sample and has less variable prediction performance. SVR also has nearly the same pattern and converges quicker to a lower error value for two of the applications (VARD and ItemRecommender), however, there is not much evidence regarding giving a better performance with more data samples. RF, on the other hand, is more variable in terms of spread and

2. Pronounced as `GQ'.

Fig. 8. Model error (%RRMSE) to assess per day prediction accuracy of three base models of VARD application over the span of 7 days. The models displayed from left to right are: MPR, SVR and RF.

Fig. 9. Model error (%RRMSE) to assess per day prediction accuracy of three base models of smallpt application over the span of 7 days. The models displayed from left to right are: MPR, SVR and RF.

Fig. 10. Model error (%RRMSE) to assess per day prediction accuracy of three base models of ItemRecommender application over the span of 7 days. The models displayed from left to right are: MPR, SVR and RF.

offers better results with more data samples.

5.2 Feasibility and assessment of the approach

We now assess the ability to enhance learning efficacy across domains. We discuss each evaluation scenario (Figure 6) in turn.

5.2.1 Scenario 1: Cross-application

This first evaluation is to ascertain whether Tamakkon is able to satisfy the objective of enhancing the learning efficiency in a cross-application scenario, which could be explained as such:

> A model for application B can be generated to predict its performance on cloud X using the learned knowledge of application A having a model to predict its performance on cloud X.

Table 4 displays the %RRMSE of the MPR, SVR and RF-based models for a cross-application scenario on EC2, where the left most column indicates the source applications with respect to target applications listed in each column. The source applications are used as the source of knowledge and Tamakkon activates the required transfer learning mode according to similarity outcomes. Each cell value in the table is the %RRMSE value for each model, except highlighted cells which correspond to the model performance when Tamakkon is compared to the base model. The column labeled as `VARD-EC2' is referred to as column1 (c1) and so the base model values for VARD are listed in column1 x row1 (c1r1) and so on. Here the term base model indicates the model that is generated using traditional machine learning methods (section:3.1.2), and training and test data sets are drawn from the same application data for which the model is generated.

C1 represents the model results for VARD being a target application and smallpt (c1r2) and ItemRecommender (c1r3) as the source, where VARD shows partial similarity with the later source resulting in the use of the Transfer-All mode for SVR and RF (section: 3.3). The observed difference of model error for MPR is 0.008%-0.015%, SVR is 0.029%-0.022% and RF is from 0%-0.052%.

C2 displays the prediction error of models generated for smallpt (target application). VARD and ItemRecommender are considered as source applications and have no similarity at application level resulting in activation of Transfer-Model (for MPR, SVR and RF). This is the case when the model specifics, as well as parameters, are used to train a model for the target application. The %RRMSE displayed in c2r1 and c2r3 are the prediction errors of the models generated by Tamakkon using the knowledge from VARD and ItemRecommender, respectively. The transferred models are performing the same as the base models (c2r2) and a negligible difference is observed for the model errors. The percentage difference for MPR-based models is 0.002%-0.008%, for SVR the range is 0.006%-0.008% and for RF the difference is 0.089%.

The right most column (c3) follows the same output trend, where ItemRecommender is the target application with smallpt and VARD as a source. The % difference of RRMSE values (c3r1, c3r2) for MPR, SVR and RF-based models is 0.002%-0.025%, 0.001%-0.002% and 0.024%, respectively.

Summary: The overall model error observed among all models is in range of 0%-0.089% which shows that a learning model can be successfully generated using learned knowledge from similar applications. Moreover, the results confirm the applicability of knowledge transfer approaches in a cross application scenario. In addition, the results also justify the feasibility of MPR, SVR and RF

TABLE 4
Scenario-1: %RRMSE for cross-applications on EC2. Transparent cell values represent the model error using Tamakkon where the source applications serve as a source of knowledge for the target application. Highlighted cell values represent model error for three application-specific base models.

		TARGET (EC2)		
		VARD-EC2	smallpt-EC2	ItemRec.-EC2
SOURCE (EC2)	VARD-EC2	MPR-B 13.30073 SVR-B 14.45976 RF-B 13.47269	MPR- 0.3611638 SVR- 0.3658111 RF- 0.4978552	MPR- 0.684563 SVR- 0.71643316 RF- 0.6699096
	smallpt-EC2	MPR- 13.3162 SVR- 14.48187 RF- 13.41974	MPR-B 0.369186 SVR-B 0.37245 RF-B 0.408157	MPR- 0.7119354 SVR- 0.7197244 RF- 0.6699096
	Item Rec.-EC2	MPR- 13.30905 SVR- 14.48974 RF- 13.41974	MPR- 0.372098 SVR- 0.374851 RF- 0.4978552	MPR-B 0.709919 SVR-B 0.717718 RF-B 0.654587

as base-learners to generate a learning model under Tamakkon .

5.2.2 Scenario 2: Cross-provider

The second evaluation aims to ascertain whether Tamakkon is able to satisfy the objective of enhancing the learning efficiency in a cross-provider scenario, i.e. where the same application needs to run on different cloud providers. This is represented by the following example scenario:

> A model for application A can be generated to predict its performance on cloud Y using the learned knowledge of the same application having a model to predict its performance on cloud X.

The data collected from the two cloud providers involved – EC2 (source) and GCE (target) – differ in feature space due to varying configuration settings at the VM level. Each plot in Figure 11 presents the model performance of each application on GCE where the same application's data on EC2 is used as the source of knowledge by Tamakkon . Moreover, these plots show the associated error rate of the transferred model. According to the similarity outcome, each source application in each plot is tagged as closely similar and is applicable for the Transfer-All mode with SVR and RF as base-learners. The left plot in Figure 11 clearly depicts a lower model error for VARD-GCE for MPR, SVR and RF where %RRMSE for SVR model on EC2 is 14.459% which is comparatively higher than 10.049% on GCE. Similarly, the %RRMSE for MPR and RF on EC2 is 13.300%-13.472%, a little higher than 9.094%-8.995% on GCE. In the middle plot, MPR, SVR and RF models are performing almost the same and interestingly with a model error of less than 2% where model error for MPR, SVR and RF on EC2 is 0.3691%, 0.3724% and 0.4081%, respectively. The models are evidently performing well on GCE where the %RRMSE for MPR, SVR and RF is 2.001%, 2.0152% and 1.770%, respectively.

The right plot shows a similar behaviour where the model error for ItemRecommender on GCE is a little higher for all base-learners in Tamakkon . The %RRMSE for the SVR model on GCE is 2.262% which is higher than 0.7177% on EC2. A similar trend is seen with MPR and RF, where the model error on EC2 is 0.7099%-0.654% which is a little lower than 2.836%-1.987% on GCE.

Furthermore, there are some surprising findings: Tamakkon results in significant reduction of model error compared to base models. For this result, the model accuracy metric is MSE as we are presenting diagnostics for one model with a particular data set. The reduced MSE is observed in two cases. First, this is observed when VARD-GCE instance knowledge is used as part of Transfer-All mode to generate a prediction model for VARD-EC2. SVR is used as the base-learner and VARD-GCE instance data is used as source knowledge. The left plot in Figure 12 describes the effect of instance data of source application (VARD-GCE) at the model training for predicting VARD-EC2 performance. The horizontal axis indicates the percentage use of source instance data mixed with the auxiliary target data, and the y-axis indicates the MSE value of test data. The MSE value is observed to consistently decrease on each percentage increase of source data. The MSE value of the base model is 177.960, which is far higher than the values seen in the graph. The lowest MSE by Tamakkon is 144.822 for VARD-EC2.

In the second case, a reduction in MSE is observed when smallpt-GCE instance knowledge is used as part of Transfer-All mode, with SVR as the base-learner, to generate a prediction model for smallpt-EC2. The results are presented in the right plot of Figure 12. As opposed to the above trend, we do not observe a continuous reduction of MSE. Nonetheless, some lower bounds are observed. Most importantly, the use of source instance data is able to maintain a continuous reduced MSE compared to 9.746 as the base model MSE.

Summary: The results confirm the positive influence of transferred knowledge, validating the assumption of applying a TL approach. The experiment also shows that it is possible and feasible to make use of existing knowledge to increase model accuracy. The applicability of this approach is clearly evident in this cross-provider scenario. Furthermore, the reduced model error shows the sensitivity of prediction models to capture data variation on different distributional data.

5.2.3 Scenario 3: Cross-application & Cross-provider

The final evaluation aims to identify whether using Tamakkon helps in providing deployment decisions across different applications and different cloud providers: in other words, Tamakkon is able to enhance learning efficiency in a cross-application, cross-provider scenario where both (a) source and target applications are different, and (b) source and target providers are different. This is illustrated by the following scenario:

> A prediction model can be generated for the performance of application B on cloud Y using the learned knowledge of the performance of application A on cloud X.

Again, this evaluation involves EC2 and GCE. Source knowledge is transferred based on the fact that there is little or no similarity neither at application nor cloud provider level.

Table 5 summarizes the results when applications on GCE are using existing knowledge of different applications on EC2. The results for cross-application cases (highlighted in grey) are excluded from this table as already discussed in 5.2.2. Each cell value in the table illustrates the %RRMSE value for each model and is assessed on the test data from target application only. Under this scenario, the models are compared with each other to assess the model error for different base-learners as well as evaluated for prediction accuracy on target application in comparison of the source application.

Considering VARD-GCE as the target application has the prediction models having a model error difference of 0.060%, 0.028% and 0.263% for MPR, SVR, and RF, respectively. It can also be observed that the models generated for VARD-EC2 are presenting a better performance for VARD-GCE with reduced %RRMSE. Similarly, smallpt-GCE as a target with VARD-EC2 (c2r1) and ItemRecommender-EC2 (c2r3) as source the percentage difference of model error for MPR-based models is 0.01%, for SVR-based models the difference is 0.003% and for RF-based models the error difference is 0.092%. The right most column displays values for ItemRecommender-GCE, where VARD-EC2 and smallpt-EC2 are considered as the source of knowledge. The %RRMSE difference for MPR, SVR and RF-based models is 0.085%, 0.055%, and 0.158%, respectively. This shows that Tamakkon is able to achieve reasonably accurate model generation using

(a) (b) (c)

Fig. 11. Scenario-2: Model error for (a) VARD, (b) smallpt, and (c) ItemRecommender. Learning models are aligned on x-axis and each set of bars represents the error of applying the same model on different cloud providers.

Fig. 12. Scenario-2: The effect of instance knowledge from a source in improving the model in a target domain (i.e. reducing its MSE).

TABLE 5
Scenario-3: %RRMSE for cross-provider, here target applications are listed in columns and source in rows. Each cell value represents the model error for three models (MPR, SVR and RF) when the source domain is EC2 and served as a source of knowledge for model generation on GCE (target domain).

		TARGET		
		VARD-GCE	smallpt-GCE	ItemRec.-GCE
SOURCE	VARD-EC2		MPR- 2.011354 SVR- 2.011354 RF- 1.862442	MPR- 2.75103 SVR- 2.31769 RF- 2.147283
	smallpt-EC2	MPR- 9.094947 SVR- 10.04975 RF- 8.995541		MPR- 2.836855 SVR- 2.26189 RF- 1.9887718
	ItemRec.-EC2	MPR- 9.161341 SVR- 10.07811 RF- 9.258656	MPR- 2.001243 SVR- 2.01526 RF- 1.770105	

knowledge of a source domain and task that have no similarity at the cloud or application level. This also shows that a good feature representation can reduce domain differences even if the domains have some heterogeneity at the cloud provider level.

Summary: The results interpret that prediction models with a reasonable accuracy can be achieved by transferring knowledge across both application and cloud provider.

5.3 Blind knowledge transfer

The results of Tamakkon are also evaluated using a blind knowledge transfer methodology which aims to select a wrong mode for the knowledge transfer. Table 6 displays %RRMSE values to compare blind knowledge transfer results with Tamakkon . smallpt-EC2 as a target application displays slightly high %RRMSE for MPR (4.496%) and very high for SVR model (19.893%) as compared to the Tamakkon achieved models (MPR-0.361% and SVR-0.365%), as shown in rows 1-3 of the table. ItemRecommender-EC2 (target) represents a similar pattern as observed in the above results and displays a model error of 4.1433% which is slightly higher as compared to the Tamakkon

TABLE 6
Comparison of blind knowledge transfer with Tamakkon .

Source (EC2)	Target	Base Learner	%RRMSE (Blind Transfer)	%RRMSE (Tamakkon)
VARD	smallpt-EC2	MPR	4.496	0.361
	smallpt-EC2	SVR	19.839	0.365
	ItemRec.-EC2	MPR	4.1433	0.684
smallpt	ItemRec-EC2	MPR	14.692	0.711
	ItemRec.-GCE	MPR	44.719	2.836
	VARD-EC2	MPR	20.0168	13.316
	VARD-EC2	SVR	17.079	14.481
	VARD-GCE	MPR	218.890	9.094
	VARD-GCE	SVR	834.458	10.049
ItemRec.	VARD-EC2	MPR	29.7177	13.309
	smallpt-GCE	MPR	1272.442	2.001

based model (0.684%). Considering that ItemRecommender as a target on EC2 and GCE has no architectural similarity with smallpt-EC2 (source), selecting the wrong mode (Transfer-All) with MPR as a base-learner results in 15-20 times high prediction error. The first two rows in Table 6 display %RRMSE (Blind Transfer) values of 14.692% and 44.719%, way higher as compared to the model error of 0.711% and 2.836% achieved usingTamakkon . Similarly, VARD as a target application has no similarity with smallpt as a source. The selection of wrong mode (Transfer-All) for VARD-EC2 (target) results in a slightly high model error for MPR (20.016%) and SVR (17.079%) models. Moreover, VARD-GCE (target) results in 24-83 times high error values for MPR and SVR models, as shown in row 8-9 of Table 6. The last two rows display no different results in comparison to the remaining examples in Table 6. For VARD-EC2 the prediction error is twice as high (29.717%) and for smallpt-GCE the error rate is extremely high (1272.442%) as compared to the %RRMSE values of Tamakkon achieved results. trate the benefits of Tamakkon which clearly improves the results The above results illustrate that using a wrong knowledge transfer approach results in a poor prediction models as compared to Tamakkon which clearly improves the prediction results.

5.4 Time and cost effectiveness

In addition to the above validation, the approach is been evaluated for time and cost-effectiveness. A key feature of Tamakkon is the utilization of existing knowledge in the form of the learned model (Transfer-Model) and data instances (Transfer-All). This results in reduced time to generate a model from scratch. Additionally, utilizing existing data instances cuts the cost required to collect a large amount of data for a new application.

Model generation on its own incurs a variable amount of time at each stage: data pre-processing, feature selection, model

calibration, parameter tuning, model training, and assessment. Figure 13 illustrates the approximate time needed to generate a learning model by using traditional model generation methods in comparison to transfer learning using our approach. MPR requires additional effort to identify a best fit model and to examine non-linear curve fitting of multiple predictors with a response variable; i.e. very transparent in ML terms. Therefore, it incurs the highest time (900 minutes) for model generation involving frequent human intervention to evaluate outcomes at each sub-stage. SVR comes second (420 minutes) where the majority of the time is used for parameter tuning. RF takes less time (300 minutes), and is observed to offer better understanding of the relationship of predictors with the response variables as compared to SVR.

Fig. 13. The time taken to generate models using traditional ML and TL.

Clearly, TL can significantly reduce the effort of model generation. The time for TL includes pre-processing of new data, similarity measurement, model transfer, training, assessment and any human intervention to check the process. Model accuracy is a proxy metric for model generation time and training cost: the lower the model error, the smaller the subset of data needed to train the generated model. Such saving brought on by the use of existing knowledge is potentially very significant as collecting data to train a model is a lengthy and costly process in terms of costs needed for cloud resources to collect data samples. Overall, model training is made more efficient by saving 60.12% of the required cost and time. Specifically, we were able to reduce VM usage for the purpose of data collection for a single application from 168 hours to 67 hours. This translates to saving $92.33 out of $153.88 on 8 EC2 VMs. Similarly, on GCE, a cost of $53.323 from a total of $88.872 was saved on one single experiment comprised of 7 nodes.

6 Related Work

A big challenge in the cloud stems from the wide variety of technologies, APIs, and terminologies used [1], [2], [24]. Furthermore, uncertainty associated with how these services are managed (e.g. scheduling algorithms, load balancing policies, co-location strategies, etc. [4], [25], [26]) add a black-box effect to this complexity. Thus, providing customers with application-specific deployment decisions is important. There are two main approaches to providing such decisions, which we review below.

Metric-based solutions rely on representing cloud resources and their capabilities in a certain way, such as using standardized KPIs (e.g.[27], [28], [29], [30]) or through benchmarking (e.g.[31], [32]). The former method results, despite all efforts, in an outdated and reductive representation due to the sheer breadth and proliferation of the cloud computing market. The latter method avoids this through persistent benchmarking in an attempt to capture irregularities and attain a detailed and up-to-date performance profile for every cloud resource type. This of course comes at a high operational cost. Moreover, a disadvantage of both methods is that they are based on application-agnostic ranking and not on knowing how the application will actually perform on a given infrastructure.

Application models focus more on the other part of the match-making decision process: application requirements. Examples include vendor independent ontologies (e.g.[33], [34]) and model-driven engineering (e.g.[35], [36], [37]). These solutions are heavily dependent on fine-grained information from domain experts, analysts and decision makers to get complete knowledge of business models and company strategies. As such, a designer must be aware of the impact of decisions, alternative decisions, actor interactions, dependencies, and processes while designing workflows and architectural models. Such processes require significant developer experience and time to follow domain-specific design principles.

ML-based methods aim to gain the best of both worlds by using experiential data (as benchmarks do) to model application behaviour on different deployment setups. A prominent trend in such studies is to focus on data analytics and MapReduce-style applications [7], [8], [38], [39], due to their operational footprint and having a recurrent workload pattern which is relatively easy to model. However, a common overhead here is training: significant data and time (which translates to cloud costs) are needed to train a model. Therefore, more recent efforts focus on reducing the learning cost. Ernest [11] offers a performance prediction for analytic applications, and uses an optimal experimental design technique to select useful training point for model generation. This, however, requires extensive evaluation to optimize the cost-performance trade-off. CherryPick [9] uses encoding methods to narrow down the search space and obtain a near optimal solution. However, it is designed for recurrent analytic applications and is restricted to provide only one solution as an optimal choice. Tamakkon , on the other hand, is not designed for any particular application domain and can optimize the cost-performance trade-off for different ranges of VMs. Paris [10] uses hybrid offline benchmarking to generate sufficient workload fingerprints to obtain a cost-performance trade-off. Though, combining synthetically generated data with real-world data typically results in the reduction of prediction accuracy as is evident from the low accuracy reported in the Paris paper as compared to that of Tamakkon .

A final note: Recent works use ML to analyse the variation of auctioned (as opposed to on-demand) cloud resources, namely AWS Spot. However, this has been proven to be a relatively trivial optimization problem [40] and is not of interest to our work.

7 Conclusion

Decision making in cloud environments is a challenging task due to the wide and ever expanding variety of IaaS service offerings. A customer entering such diverse market is likely to be confused with the range of choices on offer and without much knowledge about the selection criteria. A decision support system supplemented with traditional machine learning methods is therefore a very attractive service for cloud users. However, this inevitably comes with a significant learning time and monetary cost for making decisions specific to each application and cloud infrastructure.

In this paper we present Tamakkon , a novel solution to increase the efficiency of ML-assisted DSS to make application-specific decisions in a cross-cloud environment. The solution makes use of existing knowledge, transferring such knowledge to be used for other applications and/or cloud infrastructures. More specifically, this work applies TL to identify the type of knowledge to be transferred and details a methodology to identify similarity between different sources of knowledge. Tamakkon is not using source pre-trained models. Instead, it is training a function over the source and target application data. This approach is evaluated