Technical Report No. 2015-624

Cost-Effective Resource Configurations for Multi-tenant Database Systems in Public Clouds[1]

Rizwan Mian
Hortonworks.
Santa Clara, CA 95054, California, USA
rmian@hortonworks.com, mian@cs.queensu.ca

Patrick Martin, Farhana Zulkernine
School of Computing, Queen's University
Kingston, Ontario, Canada, K7L3N6
{martin, farhana}@cs.queensu.ca

Jose Luis Vazquez-Poletti
Departamento de Arquitectura de Computadores y Automatica
Universidad Complutense de Madrid, 28040. Madrid, Spain
jlvazquez@fdi.ucm

Abstract

Cloud computing is a promising paradigm for deploying applications due to its large resource offerings on a pay-as-you-go basis. In this report, we examine the problem of determining the most cost-effective provisioning of a multi-tenant database system as a service over public clouds. We formulate the problem of resource provisioning, and then define a framework to solve it. Our framework uses heuristic based algorithms to select cost-effective configurations. The algorithms can optionally balance resource costs against penalties incurred from the violation of Service Level Agreements (SLAs) or opt for non SLA violating configurations. The specific resource demands on the virtual machines for a workload and SLAs are accounted for by our performance and cost models, which are used to predict performance and expected cost respectively. We validate our approach experimentally using workloads based on standard TPC database benchmarks in the Amazon EC2 cloud.

Keywords: Cloud computing, multi-tenant database system, Infrastructure-as-a-Service, optimization, resource provisioning, heuristics, performance model, cost model.

[1] This work was carried out while the author was a PhD student at Queen's University, and is an extension of our journal paper published in IJCAC. Citation for the journal paper: R. Mian, P. Martin, F. Zulkernine and J.L. Vazquez-Poletti, "Cost-Effective Resource Configurations for Multi-tenant Database Systems in Public Clouds," *International Journal of Cloud Applications and Computing (IJCAC)*, vol. 5, no. 2, 2015, pp. 1-22.

Cost-Effective Resource Configurations for Multi-tenant Database Systems in Public Clouds

Rizwan Mian[a], Patrick Martin[b], Farhana Zulkernine[b], Jose Luis Vazquez-Poletti[c]

[a]*Hortonworks. Santa Clara, California, USA*

[b]*School of Computing. Queen's University. Kingston, Ontario, Canada K7L3N6*

[c]*Departamento de Arquitectura de Computadores y Automatica. Facultad de Informatica. Universidad Complutense de Madrid. 28040 Madrid, Spain*

Abstract

Cloud computing is a promising paradigm for deploying applications due to its large resource offerings on a pay-as-you-go basis. In this report, we examine the problem of determining the most cost-effective provisioning of a multi-tenant database system as a service over public clouds. We formulate the problem of resource provisioning, and then define a framework to solve it. Our framework uses heuristic based algorithms to select cost-effective configurations. The algorithms can optionally balance resource costs against penalties incurred from the violation of Service Level Agreements (SLAs) or opt for non SLA violating configurations. The specific resource demands on the virtual machines for a workload and SLAs are accounted for by our performance and cost models, which are used to predict performance and expected cost respectively. We validate our approach experimentally using workloads based on standard TPC database benchmarks in the Amazon EC2 cloud.

Keywords: Cloud computing, multi-tenant database system, Software-as-a-Service, Infrastructure-as-a-Service, inexpensive deployment, optimization, resource provisioning, heuristics, tabu search, performance model, cost model.

1. Introduction

Increasingly, application providers are using Infrastructure-as-a-Service (IaaS) clouds to provide their service offerings in a Software-as-a-Service (SaaS) setting. For example, Netflix offers online media streaming and its infrastructure is being migrated to Amazon IaaS cloud since 2010 [1]. It is easy to see that Netflix would like to minimize the operational costs using IaaS clouds while providing some guarantees on its service, such as "jerk" free movie streaming.

Similarly, NASDAQ offers a SaaS abstraction to store some financial data of an electronic trading market over the Amazon cloud [2]. This abstraction is called FinQloud. The query component of FinQloud allows broker-dealers to run fast, on-demand queries and facilitate efficient retrievals of stored data. Again, the queries must return results in a timely manner.

Likewise, Gallant FX provides a range of trading services using a SaaS abstraction in a foreign exchange trading market (Forex) using Rackspace IaaS cloud [3]. Average daily trading on Forex is estimated at $3.21 trillion, and brokers are constantly connected to Forex. Gallant FX aims to provide an infrastructure that can withstand this strenuous demand and provide a reliable network uptime using Rackspace. In all the above cases, failure to meet SaaS user expectations results in monetary losses to the SaaS providers.

A key question for a SaaS provider is to minimize the cost of hosting an application in a public IaaS cloud. For this, a SaaS provider needs to determine appropriate amounts of compute, storage and network resources. Further, a SaaS provider must balance resource costs against monetary losses arising from poor service delivery when using inadequate resources in an IaaS cloud.

Our work is aimed at minimizing the cost[2] of providing a multi-tenant database system (mDBMS) as a SaaS service using a public IaaS cloud. We capture the quality of service agreements between SaaS users and providers as SLAs. The potential loss in failing to meet these SLAs is modeled as a penalty cost that is added to the overall cost of hosting the mDBMS in an IaaS. Alternately, any SLA violation may be avoided from the outset by using more powerful IaaS resources.

Determining the least costly resource configuration in an IaaS environment faces several challenges. For example, the space of possible configurations is very large, and therefore, exact solutions cannot be efficiently determined [4]. Secondly, performance unpredictability is a major issue in meeting SLAs [5]. Thirdly, the seemingly unlimited number of resources creates a tradeoff between multiplexing and scaling [6]. Fourthly, the availability of multiple resource types (e.g. multiple storage types) increases the dimensionality of the configuration space. Further, the existence of various pricing schemes for various resource types not conforming to the pay-as-you-go philosophy complicates the provisioning problem.

We see an active interest [7-14] in optimizing an objective in an IaaS cloud, such as resource cost or execution times, typically subject to some constraints such as deadline or budget. We see that some work [7, 10, 11, 13, 15] that models the provisioning problem are able to offer optimality guarantees but usually at the cost of some simplifying assumptions, such as the existence of a single minimum when using linear programming (LP). While others assume advanced knowledge of performance parameters [7], or use analytical performance models [10-12, 14].

Analytical performance models have proven to be inadequate for database workloads in the cloud [4, 16]. We also see some optimization efforts [8, 17] in employing a cost model that is used at run-time, and cannot be used for prediction in the planning phase. Often, the optimization is augmented with a feed-back loop [9, 13, 14, 17, 18], which is promising since a public IaaS cloud is a dynamic environment.

Much of the above work is evaluated against non-standard and/or computational workloads in a simulation, private cloud or at best in a hybrid cloud. A private cloud is a controlled environment unlike a public IaaS cloud, meanwhile, efficiently executing database workloads in a hybrid cloud remains an open-problem. Some work that employs a cost or a performance model typically skips the independent validation of the model and goes straight to the evaluation of the optimization. We believe that the independent validation of the models is relevant because the errors accumulate through the models and the optimization method.

In our previous work [4], we present a framework for provisioning data analytics workloads in a public IaaS cloud. We formulate provisioning as a search problem and use greedy heuristics to find a cost-effective resource configuration prior to application deployment. However, it has several shortcomings including inaccurate response time predictions and partial reporting of resource costs.

Contributions. In this report, we build on our previous work [19, 20] to present a unified framework with improved cost and performance models and search algorithms for resource provisioning in the IaaS cloud to support the SaaS service. The revised performance and cost models [19, 20] are plugged into our framework to provide accurate predictions and account for complete resource costs in the IaaS cloud. The optimal provisioning problem is NP-hard in the general case [21]. Therefore, we consider additional search heuristics including genetic algorithms and tabu search, in determining a cost-effective configuration in an IaaS cloud. These heuristics explore different parts of the configuration space.

The heuristic algorithms hunt for a suitable resource configuration for database workloads given an objective such as the minimal dollar-cost subject to SLA constraints. Alternately, the SLA violations may be allowed at the cost of penalties, which are added to the overall cost of the configuration.

We consider database workloads that consist of transactional, analytical or mixed workloads for evaluation, and access multiple tenants. The workloads are based on standard TPC database benchmarks [22]. The SLAs are defined on a request's performance metric such as response time or throughput. The resulting configurations for

[2] In this report, we use the term monetary cost, dollar cost and cost interchangeably.

the workloads are compared against a baseline, and a global optimum. Finally, the configurations are validated in the Amazon Elastic Compute Cloud (EC2) [23].

We share several insights gained in the process of formulating the problem, developing the framework, building the models and evaluating our work. In particular, we do not see great diversity in the configurations returned by the heuristics used or by their hybrids despite the configuration space growing exponentially with increasing number of workloads and virtual machines (VMs). We primarily attribute this to the discrete nature of the cost and the configuration space.

The rest of the report is structured as follows. Section 2 discusses related work. Section 3 presents the problem addressed in the report. The outline of our framework, and the performance and the cost models are described in Section 4. Section 5 presents our heuristic approach to determine a suitable configuration. Section 6 describes a combined evaluation of the algorithms, and the performance and cost models. Section 7 shares insights gained in the development and experimentation process, and provides suggestions for future work. Section 8 concludes the report.

2. Related Work

We summarize the related work on performance and cost models, and discuss related work from optimization. Summarizing the related work for the performance model, in particular, is important to highlight the inadequacy of the analytical models that are predominantly used in the literature to optimize an objective in a cloud. Meanwhile, some of the cost models are used in the context of optimizing application execution, maximizing profit or reducing cost.

Performance Model: Analytical performance models have enjoyed great popularity in the database management systems (DBMSs) area. Weikum et al. [24] provide a survey of the advances in autonomic tuning in database technology. They conclude that self-tuning should be based on a feedback control loop and should use mathematical models with proper deployment into the system components. Analytical models, however, are hard to evolve with the underlying system and make simplifying assumptions that make them oblivious to the interactions of the dynamically changing workloads and their effects [25], which are amplified by the variance in the cloud . Therefore, there is an increasing interest in experiment-driven machine learning and statistical modeling.

The interactions between the concurrently executing requests, or a request mix, can have a significant impact on DBMS performance [16]. Ahmad et al. [26] develop an interaction-aware query scheduler that targets report-generation workloads in Business Intelligence (BI) settings. Under certain assumptions, the schedule found by this scheduler is within a constant factor of optimal, and consistently outperforms conventional schedulers that do not account for query interactions.

Ahmad et al. [27] use a combination of an offline statistical model trained on sample query mixes and an online interaction-aware simulator to estimate workload completion times. No prior assumptions are made about the internal workings of the DBMS or the cause of query interactions, making the models robust. This is particularly useful for clouds where access to the underlying devices is limited.

The performance models used for the DBMS workloads typically access a single data tenant. Further, the performance models usually provide predictions for response time only, and are validated on a local server or a local VM. In contrast, our experiment-driven performance model [20] predicts both throughput and response times for transactional and analytical workloads, and operates over a multi-tenant data-service. We propose the use of different classifiers that vary in modeling scopes and development efforts.

Cost Model: The problem of resource provisioning and modeling associated costs in clouds has received a great deal of attention recently. Vazquez-Poletti et al. [28] determine a suitable number of homogenous VMs to execute a given workload in the Amazon cloud based on the values of a novel cost-performance metric (C/P). Their method does not consider other resource costs such as storage or communication, and is applied to a workload consisting of a single work-unit, which is equivalent to a single query or a transaction. The C/P-based approach does not

account for any SLAs, or its penalties in case of violations.

Tsakalozos et al. [17] use principles from microeconomics to dynamically converge to a suitable number of VMs for a workload given a user's budget. Their approach is used at runtime and cannot be used to provide an a priori prediction of resource allocations. Bicer et al. [8] also propose a runtime resource allocation framework and their cost model's parameters are acquired by monitoring an application during execution.

Sharma et al. [29] develop a pricing model to provide "high" satisfaction for the users and the providers in terms of QoS guarantees and profitability requirements, respectively. The thrust of their work is towards *valuation* of cloud resources, and they employ financial option theory and treat the cloud resources as underlying assets.

Li et al. [30] propose a cost-effective data reliability mechanism to reduce the storage cost in a cloud. Their mechanism checks the availability of replicas and reduces storage consumption up to one-third by making certain assumptions on the reliability. Du [31] looks at maximizing revenue from cloud vendor's perspective by modeling hybrid and public cloud markets using Markovian traffics. Interestingly, her work suggests that the hybrid cloud is the most profitable model for cloud vendors.

Amazon's monthly calculator [32] estimates charges for Amazon EC2 resources, if they are used for an entire month. While the time-bound on a workload may be unknown in advance, we argue that the time-unit of a month for resource cost is excessively coarse-grained. The calculator does not have any knowledge of a workload and cannot account for application performance with a given set of resource allocations.

Our cost model [19] accounts for all the resources needed (compute, storage and network) to execute a database workload consisting of multiple queries and transactions accessing multiple data partitions. Further, our cost model accommodates user-defined SLAs and associated penalties. Moreover, the execution cost is provided at the granularity of an hour.

Optimization: Some of the cost models described earlier have been used in the context of optimizing application execution, maximizing profit or minimizing cost. For example, Bicer et al. [8] propose a runtime resource allocation framework to optimize time or cost of an application execution given a budget or a deadline, respectively. As stated in Section 1, we see some recent work [7-15, 17, 18] on optimizing resource or cloud provisioning. Many of these works are formulated as constrained optimizations, and contain both linear [7] and non-linear [15] formulations.

One approach is to optimize a goal with "hard" constraints, such as a budget or a deadline [7-11]. Li et al. [10] find minimum cost application deployment subject to processing capacity and throughput SLAs. Ruiz-Alvarez et al. use LP for optimal placement of data in the hybrid clouds [7]. Often problem requirements are transformed into hard constraints, for example, computation required does not exceed the site capacity [7, 33].

Another approach is to treat the constraints as having "soft" boundaries [12], or to combine them into a utility function that is optimized [13, 14]. Li et al. [12] find optimal deployments for large service centers and clouds subject to many constraints but with soft limits on the license availability by imposing additional licensing costs if the permitted license quota is exceeded.

Maximizing a utility function allows multi-objective optimization. Li et al. [14] find the solutions that describe the best tradeoff between conflicting performance and cost-saving goals instead of a single global optimum. In particular, they explore "good" tradeoffs between minimizing cost and maximizing QoS attributes, and observe their solutions concentrate around the "knee" of a multi-objective curve aiming for Pareto-optimal solutions.

Some of the above work are augmented with a feed-back loop, and offer revised solutions to adapt to the changes in the system [9, 13, 14, 17, 18]. For example, Ghanbari et al. [13] allocate resources in a private cloud to minimize cost to the provider while accounting for the application's SLAs. They also employ a utility function, and update applications' performance models to adapt to the changes in the system, which change the optimal configuration.

Work that solves the provisioning problem using methods like LP and mixed integer programming (MIP) offer optimality guarantees [7, 10, 11, 13, 15], but in doing so make some simplifying assumptions. For example, Ruiz-Alvarez et al. [7] use LP, which assumes linear relationships among the building blocks of the problem. Others [10-12, 14] employ analytical performance models like queuing network models (QNM) or its variants. We find that the response times for queries on a VM as predicted by simple single server center models, vary by as much

as 70% from the measured response times [4]. A simple model does not capture the impact on the workload performance of the interactions among different query types. Developing a more detailed QNM for a VM is not feasible because of the difficulties in acquiring detailed performance parameters in a public cloud environment.

Our optimization algorithms are based on heuristics and do not guarantee to find a global optimum. Heuristic based algorithms have also been explored to optimize an objective function given some constraints. For example, Wada et al. [34] use genetic algorithms to find efficient deployment of different application instances, which have different levels of SLAs. Our heuristic algorithms employ standalone cost and performance models that have been validated in a public cloud. Our algorithms and models are aimed at providing a suitable resource configuration for database workloads, which access multiple tenants. Finally, the resulting resource configurations are validated in a public cloud such as Amazon unlike much of the above work.

3. Problem Statement

We start describing the problem by providing a simple example, and then providing the formal and the general problem statement. Suppose we are given some applications as shown in Table 1. The databases used by applications belong to TPC-C, TPC-E and TPC-H database benchmarks [22]. TPC-C emulates an order processing system, TPC-E mimics workload of a brokerage firm, while TPC-H models a decision support system.

Table 1: Examples of Applications, Workloads, Request Types and Databases.

Application	Workload	Request type	Database
Analytics	Read-only	Q1, Q6	TPC-H
Trading	Write-only	Trade-order, trade-update	TPC-E
Intelligent Ordering	Hybrid	Q12, Q21, new-order, payment	TPC-H, TPC-C

The workloads stated in Table 1 consist of a number of requests that are issued by the clients of the applications. Each request is an instance of a request type, such as payment in the hybrid workload for the Intelligent Ordering application. The instances of the payment transaction vary in the payee or the amount of debit. The payment transaction accesses (reads/writes) data from the TPC-C database, but Q12 query accesses TPC-H database; therefore, the hybrid workload accesses multiple databases. In contrast, read-only and write-only workloads only access a single database, namely TPC-H and TPC-E respectively. Service-level Objectives (SLOs) are defined on a request type, such as trade-update. The SLA on the write-only workload consists of SLOs on all its request types, namely trade-order and trade-update.

An mDBMS hosts all three databases as tenants, and serves workloads from the clients of all three applications. The provisioning problem is to select a configuration for the mDBMS such that the resource costs in the cloud are minimal and all the SLAs are satisfied.

We formalize and generalize this problem statement as follows. Given a set of applications $\boldsymbol{A = \{A_1, A_2, \ldots, A_m\}}$, we say that a workload $\boldsymbol{W_i}$ for $\boldsymbol{A_i}$, is a set of requests that are issued by the set of clients of $\boldsymbol{A_i}$. Each request is an instance of a request type $\boldsymbol{R_{ij}}$ from a set $\boldsymbol{R_i = \{R_{i1}, R_{i2}, \ldots, R_{in}\}}$ for $\boldsymbol{A_i}$. The databases used by $\boldsymbol{A}$ consist of a set of data objects $\boldsymbol{D = \{D_1, D_2, \ldots, D_m\}}$. A request type $\boldsymbol{R_{ij}}$ for $\boldsymbol{A_i}$ has a service level objective $\boldsymbol{SLO_{ij}}$ and accesses some data objects in $\boldsymbol{P_i}$, where $\boldsymbol{P_i}$ is a *data partition* and $\boldsymbol{P_i \subseteq D}$ and $\boldsymbol{P_i}$ contains all the data objects accessed by $\boldsymbol{W_i}$. The SLA for $\boldsymbol{W_i}$ is composed of the set of all $\boldsymbol{SLO_i}$'s for the request types in $\boldsymbol{A_i}$.

We need compute, storage and network resources to execute $\boldsymbol{W_i}$. A configuration $\boldsymbol{C}$ for a set of workloads, $\boldsymbol{W = \{W_1, \ldots, W_n\}}$, consists of the following:

- A set of VMs $\boldsymbol{V = \{v_1, v_2, \ldots, v_r\}}$, where each VM $\boldsymbol{v_k}$ is a specific type (for example *small*, *large*, *xlarge*). Each VM type has a specific set of system attributes (e.g. OS, memory, cores), and a specific cost rate.
- A mapping of the workloads, $\boldsymbol{W}$, to VMs in $\boldsymbol{V}$ such that every workload is assigned to one VM.

- A mapping of data partitions used by ***W*** to VMs in ***V*** such that every data partition is assigned to at least one VM. The partitions are stored in cloud storage. The partitions typically vary in sizes and have different access patterns, resulting in different storage and network costs. Overlapping partitions on the same VM share the same copy of the common data objects. Assignment to more than one VM involves replication of the partition, and we assume that the replicas are read-only.

The provisioning problem is then to determine a configuration ***C*** for ***W*** such that the resource cost for executing workloads in ***W*** is minimized and all the SLAs are satisfied. Selecting a suitable configuration involves: (a) determining an appropriate set of VMs, and (b) generating an efficient mapping of data partitions, and workloads onto those VMs. Determining appropriate resources balances resource costs against the penalty costs generated by SLO violations. Meanwhile, generating an efficient mapping of data partitions and workloads to VMs balances the execution time of the requests on the provisioned resources against the thresholds defined in the SLOs in order to minimize penalties.

Executing a configuration in a public cloud results in a dollar-cost expense. Such an expense is a function over resource costs. We extend this expense with penalties for violations of SLOs defined over the workload. There are primarily three types of resources needed to execute a workload in an IaaS-cloud: (a) compute, (b) storage, and (c) network. The cost function over a configuration ***C*** in the pay-as-you-go pricing scheme is stated as:

$$\text{cost}(\boldsymbol{C}) = \text{compute\_cost}(\boldsymbol{C}) + \text{storage\_cost}(\boldsymbol{C}) + \text{network\_cost}(\boldsymbol{C}) + \text{penaly}(\boldsymbol{C}) \quad (1)$$

This is also the objective function, which needs to be minimized. Assume that ***W*** and ***V*** are finite sets containing w and v elements, respectively. Then, the number of unique mappings from ***W*** to ***V*** is v^w. This serves as the lower bound on the number of possible configurations. Determining an optimal configuration for a set of workloads given some SLO constraints or penalties is a NP-hard problem in general [21].

We consider that a data partition represents a database tenant in our report, and use them interchangeably. Tenants on the same VM share the same instance of a database system. Meanwhile, tenants on different VMs have their own database system instances, and may share the host server if the VMs are deployed on the same server. Otherwise, they only share the network.

4. Framework

We present a generic framework for determining effective configurations. The high-level architecture of the framework is shown in Figure 1. Given a set of workloads, a search algorithm looks for a minimal dollar cost configuration. In each iteration, the search algorithm chooses a suitable modification of the current configuration. The modified configuration is evaluated using a cost model. The cost model, in turn, employs a performance model to predict workload performance on a modified configuration. The cost model passes a cost value back to the search algorithm. Then the algorithm decides to either keep exploring the search space or to flag the evaluated configuration as a suitable one.

The elegance of this architecture is that various search algorithms can be used with various cost models. Similarly, different cost models can be used with different performance models. In this report, our objective is to minimize the dollar cost of the mDBMS deployment given user preferences expressed as SLOs. This is equivalent to minimizing the objective function (eq. 1) subject to SLO penalties or constraints. In the case of a SLO specified as a constraint, the algorithm discards any violating configuration. Meanwhile in the case a penalty is defined as a part of the SLO, the penalty cost is added to the overall cost of the configuration when the SLO is violated.

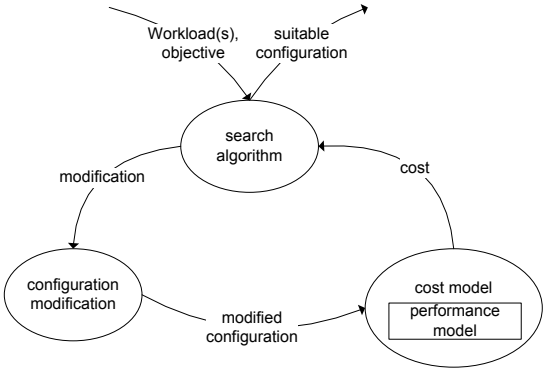

Figure. 1: Architecture of the framework for determining configurations given an objective function

4.1. Performance Model

The performance model predicts the performance of a workload on a VM type. We explore an experiment-driven approach for creating performance models for data-centric workloads on an IaaS public cloud, namely the Amazon EC2. Our approach consists of three steps: (a) sampling the space of possible request types and their instances for a request mix, (b) collecting data by executing possible samples or request mixes in a real cloud, and (c) pre-processing data and building performance models. The pre-processing activity includes analyzing the data to identify any data patterns such as non-linear trends and removing the outliers.

We use the Latin Hypercube Sampling (LHS) protocol [35], a variant of stratified sampling, over possible request types to generate two sets of samples with different random seeds. We execute both sets in the Amazon cloud using separate VMs and clients. We consider the larger set (150 samples) for training and the smaller set (100 samples) for validation. It is possible to train multiple classifiers on this data set. Therefore, we evaluate a number of classifiers on the correlation metric including linear regression, multi-layer perceptron, gaussian processes and support vector machine (SVM), and choose linear and non-linear variants of SVM [36] for our performance model. We validate the performance model against the test set. High correlation coefficients (around 0.80 or above) and low prediction errors (around 20% or below) indicate the success of our performance model.

Our performance models predict throughputs for transactions, and response times for queries. We find that linear classifiers, such as linear regression, are suitable for most request types and are fast to build and validate. They require less involvement on a developer's part and can often be employed straight out-of-the-box with default parameters in a commonly used machine learning toolkit such as Weka [37]. However, the results are unsatisfactory where there are non-linear trends in the performance data. In such cases, we explore non-linear modeling methods, which require choosing a suitable kernel and searching for appropriate parameter values.

We validate our approach by building a performance model for the workloads described in Table 1 for three different VM types, namely *small*, *large* and *xlarge* in the Amazon EC2 cloud. Studies have shown that EC2 does not always provide consistent performance so we chose to run our experiments in the region with the least variance, namely US-East-1d [5].

We wrap up the tenant databases with MySQL database system and Ubuntu Linux, and store that as an image[3] in the Amazon cloud. This greatly simplifies the engineering process, and the workloads can start execution as soon as the compute and storage resources are available, i.e. when the image is instantiated on a VM. On instantiation, the buffer pool of the mDBMS occupies 80% of the total memory of a VM instance, and is partitioned in proportion

[3] Our image (ami-7bc16e12) is publicly available at: http://thecloudmarket.com/owner/966178113014. Once the image is instantiated, the clients can connect (ssh in) to the instance and access the MySQL DBMS as root user with wlmgmt password.

to the number of tenants.

The VMs vary in their price, processing power and their capacity to hold data in memory as specified in Table 2. The request types vary in their characteristics and resource requirements. Meanwhile, the VM types vary in their system capacity, which includes the ability to hold data in memory.

Table 2: VM Types for Amazon EC2.

VM Type	Cores (#)	Memory (gb)	Cost/hr($)[4]
m1.small	1	1.7	0.08
m1.large	2	7.5	0.32
m1.xlarge	4	15	0.64

We show the mean %error[5] for the performance model built for the small VM type in Table 3. It can be seen that all mean-%errors lie below the threshold of 20%. The mean-%errors and correlation coefficients for the performance model for both large and xlarge VM types are better. This is due to the availability of more resources on powerful VMs that result in lesser variance in response times and throughput compared to the small VM type.

Table 3: Mean-%errors of the Performance Model built for the small VM type.

Queries	Q1	Q6	Q12	Q21
mean-%error (correlation coefficients)	16% (0.90)	8% (0.97)	13% (0.90)	17% (0.90)
Transactions	**New-order**	**Payment**	**Trade-order**	**Trade-update**
mean-%error (correlation coefficients)	16% (0.96)	10% (0.97)	3% (0.98)	4% (0.79)

The behavior of a request is also affected by other concurrently executing requests both in terms of the request types and their number of instances. For example, a smaller number of query instances in the request mix results in less load, and consequently an overall lower response time for queries, and higher throughput for transactions. We also observe that interactions between concurrently executing requests, such as lock contentions, can have a significant impact on the performance of a database system as claimed by Ahmed et al. [16].

4.2. Cost Model

Our cost model estimates the dollar costs for the resources executing the database workloads in a public cloud. The SaaS user's performance requirements are expressed as SLOs, violation of which incur a penalty and increase the cost of the configuration. Our cost model provides an hourly cost of workload execution, and assumes that the data already exists in the cloud. This is a reasonable assumption since many data sets such as US census data or NASA images are available in the Amazon cloud [38]. We account for the component cost for each resource type as well as penalty values in eq. 1.

The VM and the storage costs can be estimated analytically using the published unit resource costs. However, we still need to determine the communication or the network cost experimentally.

A VM is the typical compute unit in an IaaS cloud. Its price is generally metered by the hour, and any partial usage is rounded up to the next hour. The compute cost for a configuration $\boldsymbol{C}$ can be expressed as:

[4] Amazon has revised these costs since we started experimentation.

[5] %error = |measured value – predicted value| / measured value.

$$\text{compute\_cost(C)} = \left\lceil \sum_{v \in V} VMCost(v) \right\rceil \quad (2)$$

where $\boldsymbol{V}$ is the set of VMs in the configuration $\boldsymbol{C}$, and *VMCost(v)* is the hourly cost of a VM, *v*.

We assume that the tenants' databases are stored on a shared cloud storage, which is metered by the month. We prorate the monthly cost down to an hour. The hourly cost for the storage used in a configuration $\boldsymbol{C}$ is estimated by:

$$\text{storage\_cost(C)} = \left\lceil \frac{q \times E}{month\_hours} \right\rceil \quad (3)$$

where q is the unit cost of storage (in dollars per gigabyte per month), E is the aggregated size of tenants' databases in gigabytes, and *month_hours* is the number of hours in a month (e.g. 24h ×30days). Any fractional cost is rounded up to the next cent.

The network costs are estimated by:

$$\text{network\_cost(C)} = \left\lceil \sum_{v \in V} c_v \times s \right\rceil \quad (4)$$

where c_v is the estimated number of accesses to the network storage in a time-unit (hour), determined experimentally, and s is the unit network cost for accessing storage. Like storage costs, the network cost is rounded up to the next cent.

We propose a function that assigns a penalty in each time-unit in which a violation occurs. For a particular configuration $\boldsymbol{C}$ and a request type *r,* the penalty incurred in a given time-unit (hour) is given by:

$$\text{penalty(C)} = \sum_{r \in R} pcond(r, C) \times penalty(r) \quad (5)$$

where *penalty(r)* is the penalty value (in dollars) for requests of type *r* missing their SLOs in a time-unit. The binary function *pcond* indicates whether or not an SLO defined over *r* and $\boldsymbol{C}$ has been violated. In our case, SLOs consist of two metrics, namely a threshold and a penalty. We employ a binary penalty model to calculate the penalties based on the throughput and response times. In the case when a threshold specified in the SLO is not met, the full penalty stated in the SLO is applied.

We examine the effectiveness of our proposed cost model for the Amazon EC2 cloud, and consider possible configurations for an mDBMS with each tenant with its own workload. The workloads are made up of the request types stated in Table 3, meanwhile, the SLOs over a subset of the request types are defined in Table 4.

Table 4: SLOs for different Request.

Tenant	Request	Threshold	Penalty
a	Q1	200s	$0.05
b	Trade-update	0.04tps	$0.15
c	Payment	50tps	$0.10

We instantiate our cost model for the Amazon cloud and choose Elastic Block Storage (EBS) [39] to store tenant databases, primarily because EBS appears as a network mounted hard disk. Further, the data on EBS persists after the termination of VMs. In our evaluation, the clients and mDBMS are present in the same area so the only communication charges are for accesses to EBS storage. We experimentally determine the number of accesses

required for each workload W_i on each VM type at the optimal multi-programming level (MPL)[6]. This profiling is used to estimate the number of storage accesses per hour for executing W on a VM type as the average of the number of accesses by each W_i in W.

We observe that there are three main factors that influence the cost for a configuration, namely the mix of workloads in W, the VM types used in the configuration and the SLOs enforced in the configuration. The network cost varies with the workload and the VM type, while the storage cost varies with the tenant type.

We conduct three experiments where each factor is varied while holding the other two constant and the workloads are executed in the Amazon cloud. We compare the estimated resource costs directly against the invoice rendered by Amazon. We calculate the penalties based on the measured metrics (throughput and response time) instead of estimated metrics at this point. This is because we want to study errors in the cost model independent of the performance model.

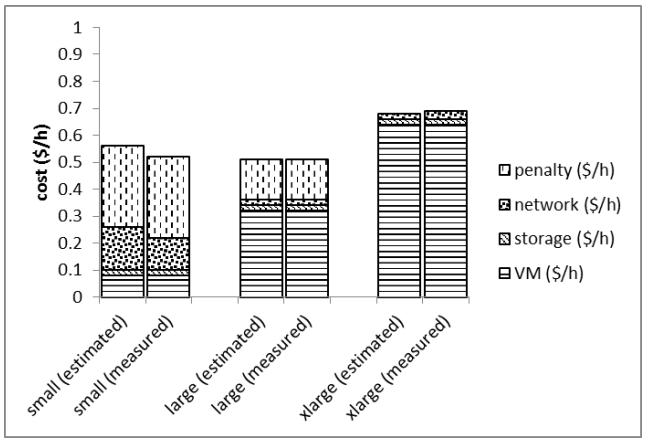

Figure 2: Estimated and measured costs for all workloads executing simultaneously with payment+update+Q1 SLOs on each VM type.

We share the details of one experiment where VM type is varied while keeping the workload mix and SLOs constant. In this case, we compare the costs of simultaneously executing all workloads with SLOs stated in Table 4 on each VM type, and display the results in Figure 2. The average error in estimating costs varies by about $0.01, which is about 2% of the total measured cost on average.

The average errors in the cost estimate for all three experiment are stated in Table 5.

Table 5: Difference between estimated and measured costs on average.

VM type	Workload Mix	SLA Penalties
$0.01	$0.03	$0.04

Given the smallest chargeable cost unit in the Amazon cloud is a cent, we argue that these errors are small and tolerable.

[6] Conceptually, the workload throughput increases as the number of concurrent requests increase, up to a point where the MPL plateaus, and then it starts decreasing. We consider the optimal MPL value to be the beginning of the plateau. The optimal MPL of small, large and xlarge VM types are 14, 75 and 115 respectively for the workloads stated in Table 1.

5. Determining a Cost-Effective Configuration

We employ heuristic-based search algorithms to find a cost-effective configuration. We represent the set of all possible configurations for a set of workloads ***W*** as a directed graph ***Configs = (N(W), E(W)).*** The set of nodes, ***N(W)***, and the set of edges, ***E(W),*** are defined respectively as:

N(W) = *{**C**$_i$ | **C**$_i$ is a valid configuration for **W**}* and

E(W) = *{(**C**$_i$, **C**$_j$) | configuration **C**$_j$ is obtained from **C**$_i$ using a permitted modification},*

We discuss modifications and the algorithms below.

5.1. Modifications

We define modifications that change the number and types of VMs in a configuration to adjust the cost. This is because we find that setting SLO penalty costs aside, the highest cost is typically incurred by the types of VMs used. Also, a user has more control in selecting the VM type in a configuration but not much over storage size and network usage.

We embed some additional heuristics in the modifications at a finer level based on four metrics, namely VM utility, workload weight, VM utilization and a busy rank. We define the utility of a VM instance as a ratio between the number of workloads and the price for the VM. Any SLO violations on a VM decrease its utility value.

The workloads may consume different amounts of resources for execution, and we represent the resource usage property by a weight value. For example, online transaction processing (OLTP) workloads consist of short and efficient transactions that require small amounts of CPU and disk I/O to complete, and are represented by a small weight value. Whereas, online analytical processing (OLAP) workloads are typically longer, more complex and resource-intensive queries that can take hours to complete, and are represented by a large weight value. The workload with the greatest weight is the heaviest workload on that VM.

We also define the utilization of a VM instance as a ratio of the sum of the number of workloads and their weight values divided by the system memory. Finally, we differentiate between a highly utilized VM instance from a VM instance with multiple workloads by using a busy metric, which simply represents the number of workloads on a VM.

The legal modifications to a configuration allowed in our model are listed below:

- *Upgrade*: Upgrade by scaling up the most utilized VM in the configuration to the next more expensive VM type. If the most utilized VM is already at the highest cost rank, then scale up the VM with the lowest cost rank.
- *Add-cheapest*: If the number of VMs is less than the number of workloads, then add an instance of the least expensive VM type to the configuration, and offload the heaviest workload from the busiest VM to the new VM.
- *Add-expensive*: This modification is identical to the Add-cheapest modification except that the newly added VM instance belongs to the most expensive VM type.
- *Add-same*: If the number of VMs is less than the number of workloads, then identify the VM with the highest utility, add an instance of the same VM type, and offload the heaviest workload from the busiest VM to the new VM.
- *Load-Balance:* If there is at least one VM executing two or more workloads, then move the heaviest workload from the busiest VM to the least utilized VM.
- *Downgrade:* Identify a VM with the lowest utility and replace it with the next cheaper VM type.
- *Downsize:* Offload all the workloads from the VM instance with the lowest utility to the least utilized VM, and remove the former from the configuration.

Note that we do not need to be concerned with tenant migration in our modifications since these modifications are applied prior to workload execution. All the modifications but one change the VM costs in a configuration, and

therefore, the cost change is referred to as modification cost. The modified configuration may result in decreased overall cost due to reduced network and penalty costs despite an increase in the VM costs. Alternately, the overall cost may increase due to under-provisioning and increased penalties. The resultant overall cost is determined after modifying the configuration and invoking the cost model with the modified configuration.

Figure 3 shows a conceptual view of the configuration space. An edge (C_i, C_j) in the search space indicates that a configuration C_j can be obtained from a configuration C_i by applying the modification.

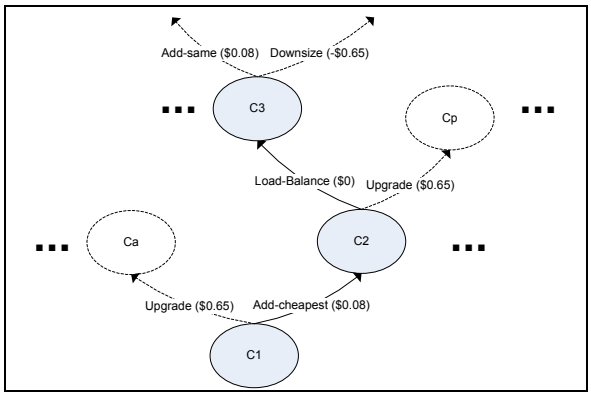

Figure 3: Conceptual view of the configuration space. An edge transforms a configuration into another configuration.

5.2. Search Algorithms

We consider three heuristics to select the modifications in each iteration, and they explore different parts of the configuration space. These algorithms are adaptive and continue to explore the configuration space provided that they keep finding cheaper configurations. We also provide non-SLO violating variants of these algorithms, by defining a simple switch which in its 'on' state discards configurations that violate SLOs.

The adaptive algorithms are based on a common template based on greedy algorithm [40]. However, they differ in their choice of modification selection. We describe the algorithms below, and present their pseudo-code in Appendix A.

Greedy template: The greedy search algorithm starts by building an initial configuration by mapping all the workloads and the data partitions on a single instance of the cheapest VM type. It then greedily selects the lowest cost modification amongst the permitted modifications in each iteration. As a possible consequence, the cost of a configuration decreases due to reduced penalties, for example. The algorithm stops at the first minimum cost configuration it finds, which serves as a baseline for the experimental results.

Adaptive greedy heuristic: The adaptive greedy algorithm extends the greedy algorithm with an ability to continue to look ahead for another minimum once the first one is found. The extension is a function of the number of workloads in the configuration and the number of iterations taken to find the last minimum. For example, if the adaptive greedy algorithm finds the last minimum in n iterations and the number of workloads is w, then it explores the search space a further wn iterations in the hope of finding a better (cheaper) minimum. If one is found then it resets the iteration counter (n) and continues to look for a better minimum until one is not found in the additional wn iterations.

Pseudo genetic algorithm (pseudo-GA): In contrast to the greedy heuristic, the pseudo-genetic algorithm chooses a random modification in each iteration from all possible modifications. This algorithm makes excessive use of random selection amongst the permitted modifications. Each permitted modification has an *equal* probability of being selected. Therefore, this algorithm is not entirely random nor does it contain all the building blocks of a genetic algorithm [41], thus, the name pseudo-GA.

Tabu search: In contrast to pseudo-GA, the tabu search algorithm selects the modifications systematically. The algorithm uses tabu constructs that consist of intensification and diversification strategies [42]. The intensification strategies promote the selection of modifications which were historically found to be good. For example, recent modifications that lowered the cost, or the modification that has lowered cost most of the time. The diversification stage, on the other hand, encourages the search process to examine unvisited regions and to generate configurations that differ significantly from those considered earlier. For example, this strategy promotes previously unselected modifications. In tabu search, each chosen modification is intentionally flagged unavailable (tabu'ed) for some number of iterations despite being a perfectly eligible and/or a promising modification. The tabu'ing of a modification is particularly useful in breaking out of cycles. The tabu duration is determined randomly over the size of the permitted modification list.

The starting point for all the algorithms is the initial configuration, where all the workloads and the tenants are mapped to a single instance of the cheapest VM type. This configuration exploits a heuristic, namely it has the lowest VM costs. Further, this configuration often turns out to be the optimal configuration when there are no SLOs defined.

6. Evaluation

The scope of the evaluation is to gauge the combined work of the framework components in finding the most cost-effective configurations with a focus on algorithms. In this report, we execute two sets of experiments to evaluate different aspects of the algorithms. The objective of the first set is to compare the results with the optimal solutions determined by an exact algorithm using small numbers of workloads, namely three. Meanwhile, the objective of the second set is to compare the performance of the promising algorithms in executing realistic workloads where the number of workload instances is thirty. In both cases, we validate some of the configurations in the Amazon cloud in order to confirm our framework's outputs.

There are four variables to the configuration cost, namely VM, storage, network and penalty costs. All four are varied in our experiments. A user has direct control over workloads and tenants which impact storage and network costs, and over SLOs which impact penalty costs. Considering different workload types allows us to vary the tenants, and hence the aggregate storage size and costs. Considering a large number of workload instances and placing SLOs on them rewards the algorithms that place them on different VM types, hence treating VM cost as a variable. The performance of an algorithm is measured by the dollar-cost of the configuration provided.

We first present the tenants and their workloads. After that, we evaluate the algorithms against the global optima in some restricted cases. Then we evaluate the algorithms with realistic workloads and discuss the diversity of the configurations provided. A summary is presented at the end of this section.

6.1. Tenants and their Workloads

The tenants for the mDBMS used in our experiments are described in Table 6. The tenants' workloads are made up of requests from the benchmarks and are chosen to exhibit different behaviours, namely read-only, write-heavy and mixed read/write. They consist of data-intensive request types, which spend significant part of their execution time accessing (reading and/or writing) data.

Table 6: Example Application Tenants.

Tenant	Workload	Data-bases	Request types
a	read-only	TPC-H	Q1, Q6 (TPC-H)
b	write-heavy	TPC-E	trade-order, trade-update (TPC-E)
c	read-write (mixed)	TPC-H, TPC-C	Q12, Q21 (TPC-H), new-order, payment (TPC-C)

A request type in a workload may have multiple instances that execute concurrently. In general, the size of a workload is unknown. Therefore, we parameterize our workload execution by a time-unit, an hour. During the workload execution, a request instance is continuously re-submitted if finished early. This ensures that the request mix is consistent at an mDBMS throughout the hour.

6.2. Evaluation against optimal comparison point

We first experiment with a small number of workloads so that the correctness of our algorithms can be judged by comparing them to an exact solution for these limited cases. We keep the workloads fixed at the combination *abc*, where the workloads *a*, *b* and *c* are defined in Table 6. This combination is executed concurrently at the optimal MPL level of the VM types specified in Table 2. We keep the SLOs' thresholds fixed, but vary their penalty values. We use the penalties in Table 4 as a base case, and amplify them, two times, rerunning the algorithms to see the resulting configurations provided by the algorithms. We compare the configuration costs returned by the search algorithms in Figure 4.

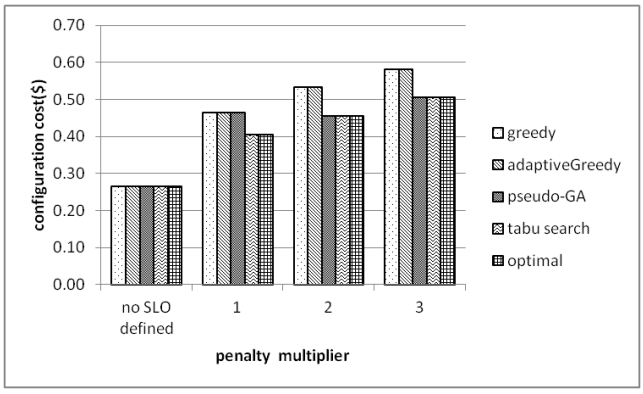

Figure 4: Cost of resulting configurations when SLOs' penalties are varied, and workload and SLOs' thresholds are fixed with a single instance of a, b and c.

Interestingly, adaptiveGreedy always returns the same configurations as greedy in the above cases. This is because once it finds a cost minimum, it always chooses the lowest cost modifications that do not allow it to move away from the minimum. Surprisingly, pseudo-GA, despite selecting random modifications, returns cheaper or equal cost configurations than adaptiveGreedy. In contrast, tabu search always provides the optimal configuration in the above cases. It may appear that the algorithms provide only two types of configurations, which are either the initial configurations or the optimal configurations. This pattern changes when we consider realistic workloads, where we see more variation in the configuration costs. In this analysis, we find pseudo-GA and tabu search promising, and we evaluate them using realistic workloads in Section 6.3.

We validate the costs of the configurations provided by the algorithms for the base case (i.e. when penalty multiplier is 1). The algorithmic validation also serves as the correctness analysis. All the heuristic algorithms except tabu search return the initial configuration, which consists of only a single small VM instance. Meanwhile, tabu search and brute-force return identical configurations, which have a single large VM instance. We execute these configurations in the Amazon cloud for about an hour. The costs of simultaneously executing all workloads on each configuration is compared in Figure 5.

We see considerable penalty costs due to lack of resources to avoid violations in the case of the configuration containing a small VM instance. We see that the overall cost reduces slightly for the configuration containing a

large VM instance due to reduced penalties but higher VM costs. This is an example of the tradeoff between penalties and resource costs.

In Figure 5, we also see that the total measured cost of the optimal configuration is just under the measured cost of the initial configuration. We anticipate that there will be cases when estimation errors will lead to a wrong configuration identified as the most cost-effective configuration. We see such a case in Section 6.3, and explore possible reasons there.

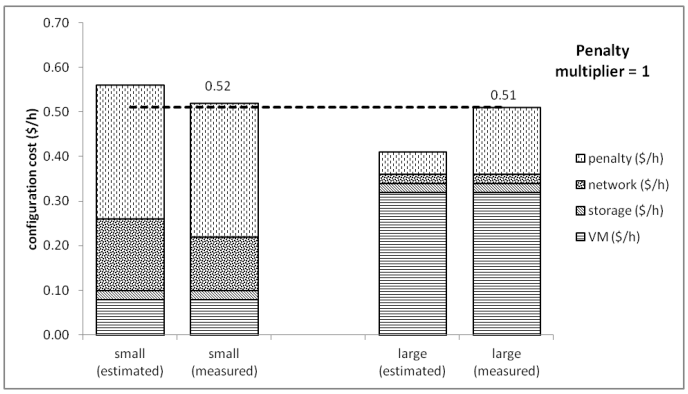

Figure 5: Estimated and measured costs for a, b and c workloads executing simultaneously subject to payment, trade-update and Q1 SLOs.

6.3. Evaluation with realistic workloads

With difficulties in obtaining real workloads, we do a best-effort job in defining workloads that exhibit characteristics of real applications. We use a combination of our workloads to exhibit behaviour similar to the aspects of web-applications listed by Cooper et al. [43] as shown in Table 7.

Table 7: Web-application Type Workloads.

Workload type	**Percentage Mix**	**Web-application characteristics**
Read-only	a (100%)	User profile cache, where profiles are constructed elsewhere (e.g., Hadoop)
Read-heavy	a (80%), b (20%)	Photo tagging; add a tag is an update, but most operations are to read tags
Update-heavy	c (100%)	Session store recording recent actions in a user session

Using a combination of the workloads similar to the ones used by Cooper et al., we define realistic workloads. We define two levels of thresholds and penalties in an SLO, namely lenient and strict. Suppose our workloads are present in equal proportions in a normal session. For example, if the number of permitted workload instances is thirty then each W_i gets an equal proportion of ten. We present a list of cases below, where one workload becomes dominant in the aggregate workload mix.

Read-only dominant (ro-dom): Let us assume that write and update workloads deplete at night. For example, consider the example of a trading market which closes in the evening, and the back-end and house keeping workloads kick in. This also provides a window to execute more analytical workloads over-night, hence we see read-only dominant.

Update-heavy dominant (up-dom): Group-on is a popular daily deal website that features discounted gift

certificates usable at local or national companies. It has over 35 million subscribers and offers coupons in over 150 markets. Recently, it sold over 25,000 GPS units in a span of few days [44]. We see update-heavy dominant in this behaviour.

Read-heavy dominant (rh-dom): Yahoo! News reported "The 5 Most Successful Viral Videos Ever" in early 2012 [45]. The number one video is a short clip about the atrocities committed in Uganda by Joseph Kony and his rebel army. This clip aims to raise awareness about Kony, who is believed to have kidnapped and enslaved some 66,000 children since the late 1980s. The film generated immense interest with a total of 100 million views over the Web in a record *six days*, with some viewers posting comments. We see read-heavy dominant in the workload mix in this case. With the above description, we define the mix of workloads in $\boldsymbol{W}$ in Table 8. The fractions represent the share of a workload type from the total permissible number of workload instances, which we set at thirty.

Table 8: Workload Mix on a mDBMS Representing Different Cases.

Share / **Use-case**	**Read-only (threshold /penalty)**	**Update-heavy (threshold/ penalty)**	**Read-heavy (threshold/ penalty)**
Normal	1/3rd (lenient/lenient)	1/3rd (strict/lenient)	1/3rd (lenient/strict)
Read-only dominant	2/3rd (strict/strict)	1/6th (lenient/lenient)	1/6th (lenient/lenient)
Update-heavy dominant	1/6th (lenient/lenient)	2/3rd (strict/strict)	1/6th (lenient/strict)
Read-heavy dominant	1/6th (lenient/lenient)	1/6th (lenient/strict)	2/3rd (strict/strict)

We define SLOs over request types accessing different tenants in Table 9. All workloads instances belonging to a single workload type have the same SLOs.

Table 9: SLO Definitions Over Different Request Types in our Workloads.

Request types	**Tenants**	**Threshold (tps)**		**Penalty ($)**	
		Lenient	Strict	Lenient	Strict
Q1	a	0.005	0.01	0.05	0.08
Payment	c	50	140	0.10	0.24
Trade-order[7]	b	40	60	0.15	0.32

Due to the large number of workload instances, determining a global optimum using an exact method becomes impractical. Therefore, we use two promising algorithms, namely pseudo-GA and tabu search and their non-violating variants, to determine suitable configurations. We plot the cost of the resulting configurations in Figure 6.

In Section 6.2, we saw that the resource cost trades off against the penalty cost. To avoid violations, an algorithm may have to over-provision resources, or try alternate VM types. Over-provisioning resources is likely to result in a higher configuration cost, which we see for pseudo-GA (non-violating) in the normal and the update-heavy cases. Alternate VM types pay off for pseudo-GA in the case of read-only and read-heavy cases.

The tabu search and its no SLO variant, though searching for different number of iterations, find identical configurations for each of the use-cases. They consist of either a single large or an xlarge VM instance. These configurations appear to be the optimal configurations, given the small size and cost.

Finally, we validate the configurations provided by the algorithms in the normal case, where they are allowed to

[7] Our client crashes when executing large number of workload instances and trade-update is present in the workload mix. Instead, we replace it with trade-order transaction that accesses the same tenant.

violate the SLOs, and configurations in the read-only dominant case, where they are not allowed to violate SLOs. We compare the estimated and measured costs of executing normal and read-only dominant cases in Figure 7.

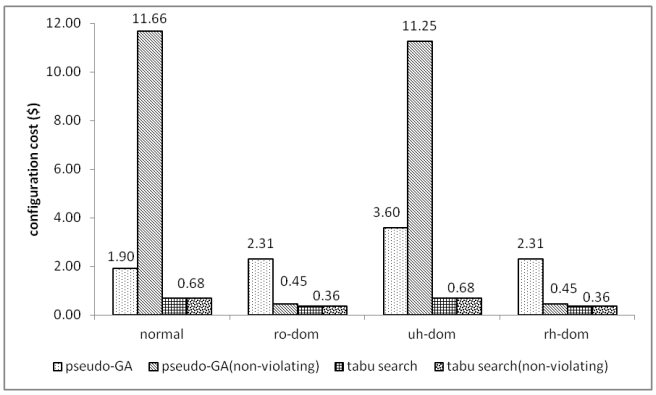

Figure 6: Costs of resulting configurations provided by pseudo-GA and tabu search, and their non-violating variants with the realistic workloads.

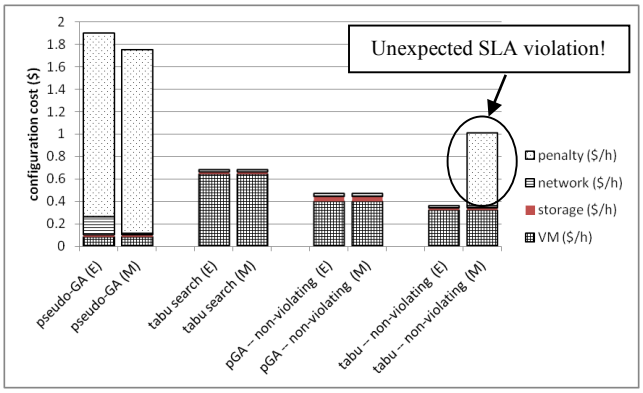

Figure 7: Estimated (E) and measured (M) costs for the normal case, where pseudo-GA and tabu search are allowed to violate the SLOs, and the read-only dominant case, where they are not allowed to violate SLOs.

We observe high penalty costs, where the SLOs are violated. This is because each instance of a workload has an SLO associated with it. On a single VM instance, either all SLOs of a workload type are met or none. The resulting penalty is the aggregation of all SLO violations. More importantly, the non-violating variant of the tabu search over-estimates the throughput in the read-only dominant case. It gives an illusion of SLOs being met, but the suggested configuration incurs violations when executed in the Amazon cloud as explained below resulting in a false negative. In this case, the %error in cost is a three digit number (183%). Meanwhile, the %error is a single digit number in the other cases. We explore the possible reasons below.

First, we use a binary penalty model, where full penalty applies when a SLO is violated, which tends to exaggerate the failure. Therefore, we see a large discrepancy between the estimated and measured costs if our performance model fails to predict the SLO breaches.

Second, our performance models are validated against a large set of training samples, and we determine their quality over the prediction values over the entire validation set of 100 samples. It is possible to have a few bad predictions but still have a good model as evident by non-zero mean-%errors in Table 3. Therefore, we expect to see fewer bad predictions for a larger number of use-cases.

Third, errors are cumulative in our framework. This is because the algorithms use the cost model, which in turn use the performance model. Therefore, any errors in the lower layers are likely to be amplified as they travel upwards. The errors could be handled better if the models and algorithms provide confidence in their solutions. For example, an algorithm may over-provision to avoid penalties if the performance model is unsure in its predictions.

SUMMARY

The space complexity of the configuration search is, at worst, linear in the number of workloads. This is because each VM type must have at least one workload mapped to it, and the size of data structures to store mappings is linear in the number of VMs, and the tenants and their replicas. Otherwise, our algorithms consider only three configuration types at any time. They are current configuration, modified configuration, and the best configuration found so far.

The upper bound on the time complexity of our algorithms is equal to the optimal provisioning, which is NP-hard in general. This is due to the adaptive nature of the algorithms. The lower bound is linear in the number of workloads since that is used in defining the size of look ahead window. In the experiments considered, our algorithms explore the search space between 10 and 1,000 iterations, where each iteration takes around 25ms on our desktop machine.

We observe that tabu search and its non-violating variant appear to find the best configurations compared to other algorithms. Further, it takes relatively fewer iterations to return a configuration. We consider tabu search to be the most effective algorithm in the heuristics considered.

In all the validation results, we usually observe a moderate %error in the estimated cost. Nonetheless, there are cases when we observe very high %error in the estimated cost due to over-estimating the performance of the mDBMS. This high %error is an exaggeration of over-estimation of the predicted throughput.

Setting penalty costs aside, we see that the VM costs are the next highest. They are also the highest costing resources. Amazon EC2 now offers over a dozen different VM types [46] that vary in their system capacities and hourly rates. Further, Amazon has recently introduced EBS-Optimized instances that provide bandwidth guarantees at an additional premium [39]. An interesting study would be to optimize an objective given more VM types with or without bandwidth guarantees, and validating the results in the Amazon cloud.

In all the experiments, we observe low storage costs, especially compared to the network and VM costs. This is because monthly storage costs are already low, and prorating it gives even lower hourly cost, which is then rounded up to the next cent. This relatively lower storage cost is in-line with the widening cost-value gap between storage and other computational resources including network and processors.

Our workloads are distorted versions of the realistic workloads presented by Cooper et al. [43]. Nonetheless, we argue that they suffice for the evaluation. Our workload combination contains at most eight request types. This is reasonable since TPC-C and TPC-E benchmarks have five and ten transactions, respectively, although TPC-H has 22 queries. A real database system is rarely a read-only or a write-only service. It usually serves a combination of transactional and analytical workloads [47].

7. Insights and Opportunities

We gained several insights in the process of formulating the problem, developing the framework, building the models and evaluating our work. We discuss them below with suggestions for future work.

Discrete configuration space: We do not see a great variety in the configurations returned by algorithms, despite employing different heuristics. We attribute this to the discontinuous or the discrete nature of the cost and the configuration space. The cost of a configuration consists of four component costs: VM, storage, network, and penalty costs. The VM instances are available in discrete units, i.e. there is no way of acquiring "two-and-a-half" VM instances, and only paying for that. Similarly, the storage space is usually allocated in discrete units, say 1gb [39], and typically charged using a step function. Similarly, the network cost associated with accessing data storage also follows a step function. Finally, we use a binary penalty model with fixed thresholds and penalties. Any SLO

violations result in discrete penalties. Alternate views are possible by using a prorated model for calculating penalty, for example, and are left for future consideration.

The discrete nature of the configuration cost is very limited compared to a continuous quantity like time. This does not reduce the complexity of the search problem, which is still NP-hard in the general case. Nonetheless, exploring whether the above characterization of the configuration cost be exploited to provide us with an optimal configuration merits some investigation.

Hybrid heuristics: The basic heuristics our algorithms use are greedy, pseudo-GA and tabu. We have explored different hybrids of these heuristics but find that they provide similar configurations as the pure heuristics in most cases with the workloads and VM types considered. This might be due to the discrete nature of the configuration space. For example, the tabu greedy algorithm is an extension to the adaptive greedy heuristic, where each chosen modification is tabu'ed for some iterations. Tabu greedy does not use additional tabu constructs. We find that the results of adaptive greedy and tabu greedy are the same in many cases. The hybrid heuristics may become relevant when considering many more workloads and VM types. This is because they may perform "fine-tuning" of the configuration. We leave this for future consideration.

Guaranteed global optimum: The heuristic search algorithms vary in their sophistication and their ability to find suitable configurations. However, the algorithms do not guaranty optimality of the configuration. It can be seen that we are optimizing an objective function subject to some constraints.

With appropriate formalization of the problem statement and constraints, it becomes possible to use off-the-shelf modeling packages like AIMMS [48] to find the guaranteed global optimum though it may take a very long time. We are particularly inspired by the work of Curino et al. [15] in this matter, who perform non-linear constrained optimization to find an assignment that minimizes the number of machines while avoiding resource over-commitment.

Performance model: Presently, the performance model provides raw predictions without expressing any confidence in them. This is an important issue since the errors are cumulative in our framework, and we need some method of managing the errors across the framework components. We seek performance models that express confidence in their predictions, and have the ability to reuse prior data and adapt online for unknown requests or VM types.

We consider an adaptive model to satisfy these requirements, and see some promising work in this direction by Sheikh et al. [25]. While the adaptive model may eventually evolve to an unknown environment, the evolution can be sped up by an "appropriate" initial state. Therefore, we envision a meta-model that generates the initial version or the bootstrap of the adaptive model given a workload and SLOs. Both meta and adaptive models are complementary and are particularly suited for a cloud environment. This is because a public cloud has a high level of variance [5].

Cost model: Our cost model does not account for any migration expense, which would be required in the case of deploying a revised configuration during workload execution. A configuration revision may be necessary with changing workloads or SLAs. There are two major components to migration, namely execution state and the data state. The primary goal of the execution state to our interest is the progress-so-far of the currently executing requests. Meanwhile, the data partitions exist on the network-type disks, which can be remounted to the new VM.

The execution state can be migrated in a few ms [49], meanwhile, the EBS volumes can be reattached to the new VM using Amazon EC2's Application Programming Interface (API) [50]. There will be some disruption to the workload execution. However, in both cases, the scope of the cost model is to account for the expense and not for the process.

Similarly, our work does not include the processes of data partitioning and maintaining data consistency. We assume that partitions already exist, which they do in the case of multi-tenant databases. Like migration, the partitioning process is orthogonal to our work, and we see some promising research on partitioning and providing consistency guarantees [51, 52]. Modeling the cost of partitioning and consistency also requires extensions to our cost model.

Our cost model assumes that the data already exists in the cloud, and does not model data transfer over WAN.

While adequate for workload execution in a single data-center, our cost current cost model needs to be expanded in order to deal with any inter data-center communication costs.

We believe that the migration, partitioning and consistency processes can be incorporated into our framework to provide dynamic refinements and an autonomic framework.

Workloads: We use transactional, analytical and mixed workloads based standard transactional (TPC-C and TPC-E) and analytical benchmarks (TPC-C) in our evaluation. However, these workloads are static. We intend to explore dynamic workloads, which change in request types or number, or in SLA. We intend to consider different workload types including analyzing data of E-opinions.com [53]. Finally, we will use a random benchmark, which will be a synthetic dataset aimed to stretch the prediction and the cost models.

8. Conclusions

We formulate the provisioning problem for providing a multi-tenant DBMS in an IaaS cloud, and evaluate the combined effects of the performance and cost models with the heuristic based algorithms in a public IaaS cloud. In the search space, each node is a possible configuration and edges between nodes are the modifications that convert one configuration into another. We are able to consider a variety of possible modifications with this representation.

The search space representation allows us to apply standard search heuristics and algorithms. Given that the problem of finding a suitable configuration is NP-hard, we present heuristic-based algorithms to find a suitable configuration. We see from the evaluation of the algorithms that there are a number of local minima in the configuration space and that the adaptivity of the algorithms results in better configurations.

The evaluation supports the claim that our framework is an effective tool for provisioning multi-tenant DBMS as a SaaS over an IaaS cloud. The framework takes into account properties of the workload, such as request types, frequencies and SLOs, as well as the resource costs in the IaaS cloud, and discovers a minimal cost configuration for the workload. The impact of SLOs is captured by a penalty cost or a constraint.

Our work is relevant to multi-tenant DBMSs that seek to find the best resource configuration for the workloads of their tenants. We claim that our work is a valuable contribution and provides a basis for executing any database workload type in a SaaS service over an IaaS cloud.

Acknowledgements

The authors acknowledge research support from National Science and Engineering Research Council of Canada (NSERC), and ServiceCloud (MINECO TIN2012-31518).

References

[1] A. Cockcroft, Netflix (keynote). *OSCON data open source convention* (Portland, OR, USA), O'Reilly, 2011. http://www.oscon.com/oscon2011/public/schedule/detail/20187.

[2] NASDAQ-OMX, *FinQloud* Retrieved on 25th Dec, 2015. http://ir.nasdaq.com/releasedetail.cfm?releaseid=709164.

[3] RackSpace, *Gallant FX Finds On-demand Support for an On-demand Market via Rackspace* Retrieved on 25th Dec, 2015. http://www.rackspace.com/knowledge_center/case-study/gallant-fx-finds-on-demand-support-for-an-on-demand-market-via-rackspace.

[4] R. Mian, P. Martin, J.L. Vazquez-Poletti, Provisioning data analytic workloads in a cloud. Future Generation Computer Systems (FGCS), vol. 29, issue 6, 2013, pp. 1452–1458.

[5] J. Schad, J. Dittrich, J.-A. Quiane-Ruiz, Runtime measurements in the cloud: observing, analyzing, and reducing variance. Proceedings of VLDB Endowment vol. 3, issue 1-2, 2010, pp. 460-471.

[6] R. Mian, Managing Data-Intensive Workloads in a Cloud (Ph.D. Depth Paper). Technical Report#: 2011-581, P. Martin. School of Computing, Queen's University 2011. [Online] Retrieved on Dec 25th, 2015. http://research.cs.queensu.ca/TechReports/Reports/2011-581.pdf.

[7] A. Ruiz-Alvarez, M. Humphrey, A Model and Decision Procedure for Data Storage in Cloud Computing. *12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)* (Ottawa, ON, Canada), 2012. pp. 572-579.
[8] T. Bicer, D. Chiu, G. Agrawal, Time and Cost Sensitive Data-Intensive Computing on Hybrid Clouds. *12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)* (Ottawa, ON, Canada), 2012. pp. 636-643.
[9] Q. Zhu, G. Agrawal, Resource Provisioning with Budget Constraints for Adaptive Applications in Cloud Environments. IEEE Trans. on Services Computing, vol. 5, issue 4, 2010, pp. 497-511.
[10] J. Li, J. Chinneck, M. Woodside, M. Litoiu, Fast scalable optimization to configure service systems having cost and quality of service constraints. *Proceedings of the 6th International Conference on Autonomic computing (ICAC)* (Barcelona, Spain), ACM, 2009. pp. 159-168.
[11] J. Li, J. Chinneck, M. Woodside, M. Litoiu, G. Iszlai, Performance model driven QoS guarantees and optimization in clouds. *Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing* (Vancouver, BC, Canada), IEEE Computer Society, 2009. pp. 15-22.
[12] J.Z. Li, M. Woodside, J. Chinneck, M. Litoiu, CloudOpt: Multi-goal optimization of application deployments across a cloud. *Network and Service Management (CNSM), 2011 7th International Conference on* (Paris, France), IEEE, 2011. pp. 1-9.
[13] H. Ghanbari, B. Simmons, M. Litoiu, G. Iszlai, Feedback-based optimization of a private cloud. Future Generation Computer Systems (FGCS), vol. 28, issue 1, 2012, pp. 104-111.
[14] H. Li, G. Casale, T. Ellahi, SLA-driven planning and optimization of enterprise applications. *Proceedings of the first joint WOSP/SIPEW international conference on Performance engineering* (San Jose, CA, USA), ACM, 2010. pp. 117-128.
[15] C. Curino, E.P.C. Jones, S. Madden, H. Balakrishnan, Workload-aware database monitoring and consolidation. *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data* (Athens, Greece), ACM, 2011. pp. 313-324.
[16] M. Ahmad, A. Aboulnaga, S. Babu, Query interactions in database workloads. *Proceedings of the Second International Workshop on Testing Database Systems* (Providence, RI, USA), ACM, 2009. pp. 1-6.
[17] K. Tsakalozos, H. Kllapi, E. Sitaridi, M. Roussopoulos, D. Paparas, A. Delis, Flexible use of cloud resources through profit maximization and price discrimination. *27th International Conference on Data Engineering (ICDE)* (Hannover, Germany), IEEE, 2011. pp. 75-86.
[18] M. Litoiu, J. Rolia, G. Serazzi, Designing process replication and activation: a quantitative approach. IEEE Transactions on Software Engineering, vol. 26, issue 12, 2000, pp. 1168-1178.
[19] R. Mian, P. Martin, F. Zulkernine, J.L. Vazquez-Poletti, Estimating Resource Costs of Data-intensive Workloads in Public Clouds. *10th International Workshop on Middleware for Grids, Clouds and e-Science (MGC) in conjunction with ACM/IFIP/USENIX 13th International Middleware Conference 2012* (Montreal, QC, Canada), ACM, 2012.article. 3.
[20] R. Mian, P. Martin, F. Zulkernine, J.L. Vazquez-Poletti, Towards Building Performance Models for Data-intensive Workloads in Public Clouds. *4th ACM/SPEC International Conference on Performance Engineering (ICPE)* (Prague, Czech Republic), ACM, 2013. pp. 259-270.
[21] R. Mian, Cost-Effective Resource Configurations for Executing Data-Intensive Workloads in Public Clouds (PhD Thesis). School of Computing. Queen's University 2013. [Online] Retrieved on 25th Dec, 2015. http://qspace.library.queensu.ca/jspui/bitstream/1974/8497/1/Mian_Rizwan_201311_PhD.pdf.
[22] TPC, *Transaction Processing and Analytical Database Benchmarks* Retrieved on 25th Dec, 2015. http://www.tpc.org/information/benchmarks.asp.
[23] Amazon, *Elastic Compute Cloud (EC2)* Retrieved on 25th Dec, 2015. http://aws.amazon.com/ec2/.
[24] G. Weikum, A. Moenkeberg, C. Hasse, P. Zabback, Self-tuning database technology and information services: from wishful thinking to viable engineering. *Proceedings of the 28th international conference on Very Large Data Bases* (Hong Kong, China), VLDB Endowment, 2002. pp. 20-31.

[25] M.B. Sheikh, U.F. Minhas, O.Z. Khan, A. Aboulnaga, P. Poupart, D.J. Taylor, A bayesian approach to online performance modeling for database appliances using gaussian models. *8th ACM international conference on Autonomic computing (ICAC)* (Karlsruhe, Germany), ACM, 2011. pp. 121-130.
[26] M. Ahmad, A. Aboulnaga, S. Babu, K. Munagala, Modeling and exploiting query interactions in database systems. *Proceedings of the 17th ACM conference on Information and knowledge management* (Napa Valley, CA, USA), ACM, 2008. pp. 183-192.
[27] M. Ahmad, S. Duan, A. Aboulnaga, S. Babu, Predicting completion times of batch query workloads using interaction-aware models and simulation. *Proceedings of the 14th International Conference on Extending Database Technology (EDBT'11)* (Uppsala, Sweden), ACM, 2011. pp. 449-460.
[28] J.L. Vazquez-Poletti, G. Barderas, I.M. Llorente, P. Romero, A Model for Efficient Onboard Actualization of an Instrumental Cyclogram for the Mars MetNet Mission on a Public Cloud Infrastructure. PARA2010: State of the Art in Scientific and Parallel Computing, Lecture Notes in Computer Science (LNCS), vol. 7133, issue 2010, pp. 33-42.
[29] B. Sharma, R.K. Thulasiram, P. Thulasiraman, S.K. Garg, R. Buyya, Pricing Cloud Compute Commodities: A Novel Financial Economic Model. *12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)* (Ottawa, ON, Canada), 2012. pp. 451-457.
[30] W. Li, Y. Yang, J. Chen, D. Yuan, A cost-effective mechanism for Cloud data reliability management based on proactive replica checking. *12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)* (Ottawa, ON, Canada), 2012. pp. 564-571.
[31] L. Du, Pricing and Resource allocation in a Cloud Computing Market. *Workshop on Cloud Computing Optimization (CCOPT 2012) in conjunction with 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*. (Ottawa, ON, Canada), 2012. pp. 817-822.
[32] Amazon, *Simple Monthly Calculator* Retrieved on 25th Dec, 2015. http://calculator.s3.amazonaws.com/calc5.html.
[33] S. Chaisiri, L. Bu-Sung, D. Niyato, Optimal virtual machine placement across multiple cloud providers. *IEEE Asia-Pacific Services Computing Conference (APSCC)* (Singapore), 2009. pp. 103-110.
[34] H. Wada, J. Suzuki, K. Oba, Queuing Theoretic and Evolutionary Deployment Optimization with Probabilistic SLAs for Service Oriented Clouds. *2009 World Conference on Services - I* (Los Angeles, CA, USA), IEEE, 2009. pp. 661-669.
[35] C.R. Hicks, K. Turner Jr, Fundamental concepts in the design of experiments, Oxford University Press, New York, 1999.
[36] S.K. Shevade, S.S. Keerthi, C. Bhattacharyya, K.R.K. Murthy, Improvements to the SMO algorithm for SVM regression. IEEE Trans. on Neural Networks, vol. 11, issue 5, 2000, pp. 1188-1193.
[37] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, I.H. Witten, The WEKA data mining software: An update. ACM SIGKDD Explorations Newsletter, vol. 11, issue 1, 2009, pp. 10-18.
[38] Amazon, *Public Data Sets* Retrieved on 25th Dec, 2015. http://aws.amazon.com/datasets.
[39] Amazon, *Elastic Block Store (EBS)* Retrieved on 25th Dec, 2015. http://aws.amazon.com/ebs/.
[40] D. Jungnickel, Chapter 5: The Greedy Algorithm, in: Graphs, Networks and Algorithms, Springer, 2005, pp. 123-146.
[41] M. Mitchell, An Introduction to Genetic Algorithms, 5th ed., MIT Press, Cambridge, Massachusetts, 1999.
[42] F. Glover, M. Laguna, Tabu Search, Springer, 1998.
[43] B.F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, R. Sears, Benchmarking cloud serving systems with YCSB. *1st ACM Symposium on Cloud Computing (SoCC)* (Indianapolis, IN, USA), ACM, 2010. pp. 143-154.
[44] R. Mohammed, *How to Save Groupon in Harvard Business Review* 2012.[Online] Retrieved on 25th Dec, 2015. http://blogs.hbr.org/cs/2012/12/how_to_save_groupon.html.
[45] Wolchover, *The 5 Most Successful Viral Videos Ever by Yahoo! NEWS* 2012.[Online] Retrieved on 25th Dec, 2015. http://ca.news.yahoo.com/5-most-successful-viral-videos-ever-154806218.html.
[46] Amazon, *EC2 Instance Types* Retrieved on 27th April, 2011. http://aws.amazon.com/ec2/instance-types/.

[47] M. Zhang, B. Niu, P. Martin, W. Powley, P. Bird, Utility Functions in Autonomic Workload Management for DBMSs. International Journal On Advances in Intelligent Systems, vol. 5, issue 1 and 2, 2012, pp. 66-75.
[48] AIMMS, Retrieved on 25th Dec, 2015. http://www.aimms.com/.
[49] C. Clark, K. Fraser, S. Hand, J.G. Hansen, E. Jul, C. Limpach, I. Pratt, A. Warfield, Live migration of virtual machines. *USENIX Association Proceedings of the 2nd Symposium on Networked Systems Design & Implementation (NSDI)* (Berkeley, CA, USA), Usenix Assoc, 2005. pp. 273-286.
[50] Amazon, *Amazon EBS Dimensions and Metrics* Retrieved on 1st Aug, 2010. http://docs.amazonwebservices.com/AmazonCloudWatch/latest/DeveloperGuide/CW_Support_For_AWS.html#ebs-metricscollected.
[51] C. Curino, E. Jones, Y. Zhang, S. Madden, Schism: a workload-driven approach to database replication and partitioning. Proc. VLDB Endow., vol. 3, issue 1-2, 2010, pp. 48-57.
[52] A. Pavlo, C. Curino, S. Zdonik, Skew-aware automatic database partitioning in shared-nothing, parallel OLTP systems. *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data* (Scottsdale, AZ, USA), ACM, 2012. pp. 61-72. http://dl.acm.org/citation.cfm?id=2213844&CFID=83613397&CFTOKEN=28959253.
[53] P. Massa, P. Avesani, Controversial users demand local trust metrics: An experimental study on Epinions.com community. *20th National Conference on Artificial Intelligence and the 17th Innovative Applications of Artificial Intelligence Conference, AAAI-05/IAAI-05* (Pittsburgh, PA, USA), American Association for Artificial Intelligence, 2005. pp. 121-126.

Appendix A: Pseudo-code of Algorithms

Figure 8 shows how different variants of the heuristic algorithms are derived from the greedy search *template*. The greedy algorithm is a standard heuristic. Therefore, we do not reproduce its pseduo-code. Instead, we present the pseduo-code of the adaptiveGreedy algorithm, and the differences in pseduo-GA and tabu search compared to the adaptiveGreedy heuristic.

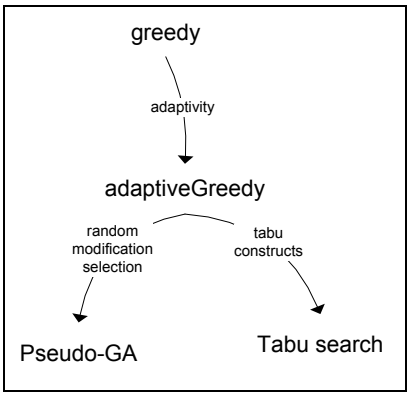

Figure 8. Relation between search algorithms, and their derivation path.

AdaptiveGreedy

```
Input: initial configuration # set of all the workloads
                             # mapped to the cheapest VM type
Output: a minimum

Proc adaptiveGreedy(Config initial){
 current_conf := initial    # initial configuration
```

```
# boolean for representing a minimum
minimumFound := false
# boolean indicating when we should terminate
terminate := false

# representing mandate to explore after a minimum is found
lease := 0
# to keep track of mandate explored
counter := 0
look_ahead  := init_conf.numOfWorkloads

while (terminate = false) {
   # variable to keep track of number of iterations
   iter = iter + 1

      # selecting a suitable modification
      # and modify current config accordingly
      candidateMods :=          ValidModsListOrderedByIncreasingModCost (current_conf)

   if (candidateMods.size = 0) {
      if (minimumFound = true AND counter = lease) {
         # minimum exists and lease expired
         terminate := true
         break  # break loop
      } else if (minimumFound = true) {
         #skip iteration but increment counter
         counter := counter + 1
      }
      # else just skip iteration and
      # hope some modification will become available
      continue # skip iteration
   } else {
      chosenMod := candidateMods.head

      #modify current modification
      # based on chosen modification
      new_conf :=modify(current_conf, chosenMod)
   }

   # detecting minimum and setting lease period
   # call to cost model is embedded in reference to ".cost"
   if (current_conf.cost < new_conf.cost ){ # minimum
      if (minimumFound = false){
         minimumFound := true              # 1st minimum found
         counter := 1                      # initialize counter for lease
         lease := look_ahead x iter# initialize lease
         minimum := current_conf
         current_conf := new_conf
      } else {
         if (current_conf.cost < minimum.cost) {
            lease := lookahead x counter   # reset lease
            counter := 1                   # reset counter
            current_conf := new_conf
         } else if (counter ≤ lease) {
            # new minimum is not lower than current minimum,
            # so ignore it and increment counter
               counter := counter + 1
               current_conf := new_conf
```

```
            }  else { # lease expired -- terminate
                  terminate := true
            }
         }
      } else { # not a minimum
         If (minimumFound = true) {
            If (counter ≤ lease) {
               counter := counter + 1
               current_conf := new_conf
            } else { # lease has expired
               terminate := true
            }
         } else {
            # no minmum found yet so keep looking
            current_conf := new_conf
         }
      }
   }
   Return minimum
}
```

Psuedo-GA

Pseduo-GA is identical to adaptiveGreedy except that the modification is chosen randomly at each iteration. The differences are stated below.

```
candidateMods := ValidModsList(current_conf)
randomIndex := randomgenerator.nextInt
(candidateMods.size())
chosenMod := candidateMods.randomIndex
```

Tabu GA

Tabu search is also similar to adaptiveGreedy except the modification is selected based on tabu constructs. The differences are stated below.

```
candidateMods :=
   generateModificationListUsingTabuConstructs(current_conf)
```

We present the pseduo-code for sorting permitted modifications based on tabu constructs below.

```
Input: current configuration
Output: modification list generated by tabu constructs

Proc generateModificationListUsingTabuConstructs
(Config conf){
  # this procedures uses a set of auxiliary procedures
  # (prefixed by get)  to obtain modificaitons that satisfy a
  # certain property. An auxiliary procedure only returns  a
  # modification that is valid for the current configuration

    # hash avoids adding the same modification
    # twice to the candidate list
    OrderedHash          candidatelist
```

```
    # intensify
        recent := getRecentModificationThatLoweredCost
          candidatelist.add(recent)

        # quality is the ratio of number of times lowered
        # cost/number of times chosen
          quality := getHighestQualityModification
          candidatelist.add(quality)

          frequent = getHighestChosenModification
          candidatelist.add(frequent)

    #diversify
        unchosen := getNeverChosenModification
          candidatelist.add(unchosen)

          neverImprove := getNeverLoweredCostModification
          candidatelist.add(neverImprove)

    # intensify/diversify depending on the VMs
    # in the current configuration
        cheapest := getCheapestModification
          candidatelist.add(cheapestModification)

        # get most expensive modification
        # for current configuration
          expensive := getExpensiveModification
          candidatelist.add(expensiveModification)

    return candidatelist
}
```

Further, the selected modification is checked for tabu status and tabu override. A tabu status can be overridden if the new configuration results in a lower cost than the current minimum.

```
 Foreach mod in candidateMods {
chosenMod = mod
        if (chosenMod.tabu=false) {
                chosenMod.tabu := true

                # set tenure based on the current candidate
   # size list instead of a constant this makes tenure duration
   # adaptive and avoids large cycles
                chosenMod.tenure := randomgenerator.nextInt

                (candidateMods.size)
                new_conf := modify(current_conf, chosenMod)
                break     # break out of foreach loop
        } else if (minimumFound) {
                # tabu is true but still need to check for tabu or
   # aspiration override before discarding chosenMod

                new_conf := modify(current_conf, chosenMod)

                If (new_conf.cost < minimum.cost) {
                        # aspiration override successful
                        Break                   # break out of foreach loop
                } else {
```

```
                                # override unsuccessful
                                # continue foreach and explore next modification
                                New_conf = null

                                # continue to the next cycle of foreach
                                # to explore next modification in the candidateList
                                Continue
                        }
    } # foreach loop ends
```