Technical Report No. 2013-612

Determining Cost-Effective Configurations for Data-intensive Workloads in the Cloud

Ph.D. Thesis Proposal[1]

by

Rizwan Mian

School of Computing
Queen's University, Kingston,
Ontario, Canada K7L 3N6

EXAMINING COMMITTEE

Dr. Patrick Martin
Dr. Selim Akl
Dr. Ahmed Hassan

Revised submission: 31th October, 2012

[1] Acknowledgements: This proposal has been written under the supervision of Dr. Patrick Martin. Wendy Powley has reviewed two drafts of this proposal. Mastoureh Hassannezhad has also provided some feedback. Dr. Farhana Zulkernine provided her thesis proposal as an example. I thank them all.

Contents

ABSTRACT

The promise of "infinite" resources given by the cloud computing paradigm has led to recent interest in exploiting clouds for large-scale data-intensive computing. Given this supposedly infinite resource set, we need a management function that regulates application workloads on these resources. This doctoral research focuses on two aspects of workload management, namely provisioning and mapping. We envision an autonomic and generic framework for resource provisioning and workload execution. We develop search algorithms, and associated performance and cost models to minimize the cost of executing data-intensive workloads in a cloud. The work is evaluated using diverse workloads in a public cloud. In this proposal, we state our research plan to complete the doctoral study.

1 Outline of the Research Proposal

An outline of our research was presented at CCGrid, 2012 [1]. In this proposal, we provide a revised research plan with additional details on the evaluation. We have examined the state-of-the-art of workload management for data-intensive computing in the clouds [2, 3]. In Section 2, we formulate the constructs for determining appropriate resources and generating an efficient workload mapping. Section 3 presents our vision for a framework for autonomic workload execution in a cloud, parts of which have already been published [4]. Section 4 describes our outstanding research to complete the doctoral study. Section 5 outlines our intended evaluation. The concluding section highlights the major contributions of the proposed research and its significance to the users who wish to execute their workloads in the clouds.

2 Problem Statement

Given an application A, we say that the *workload* for A is a set of requests that are issued by the set of clients for A. Each request is an instance of a request type R_i from a set $R = \{R_1, R_2, \ldots, R_n\}$ for A. The data used by A consists of a set of data objects or partitions $D = \{D_1, D_2, \ldots, D_m\}$.

A request type in $\boldsymbol{R}$ accesses the same subset of data objects $\boldsymbol{P_i} \subseteq \boldsymbol{D}$ and has a service level objective $\boldsymbol{SLO_i}$. We call $\boldsymbol{P_i}$ a *data partition* and assume that the $\boldsymbol{P_i}$'s can overlap. The SLA for a set of workloads, $\boldsymbol{W}$, is composed of the set of all $\boldsymbol{SLO_i}$'s for the request types in the workloads in $\boldsymbol{W}$. We need compute, storage and network resources to execute the workloads in $\boldsymbol{W}$. A configuration $\boldsymbol{C}$ for a set of workloads, $\boldsymbol{W}$, contains the following:

- A set of Virtual Machines (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 set of data partitions used by *W*. The partitions are stored on a set of network-disks. The partitions typically vary in sizes and have different access patterns, resulting in different storage and network costs.
- A mapping of data partitions, $\boldsymbol{P_i}$, to VMs in $\boldsymbol{V}$ such that every data partition is assigned to at least one VM. Assignment to more than one VM involves replication of the partition. Overlapping partitions on the same VM share the same copy of the common data objects.
- A mapping of the workloads, $\boldsymbol{W}$, to VMs in $\boldsymbol{V}$ such that every workload is assigned to one VM.

The problem is then to determine a configuration $\boldsymbol{C}$ for $\boldsymbol{W}$ such that an objective function is optimized. Selecting a suitable configuration involves: (a) determining appropriate resources, and (b) generating an efficient mapping of data partitions and workloads onto those resources. Suppose, the objective is to minimize the configuration cost for the execution of all the workloads – the objective function is formally stated in Section 4. In this case, 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. We transform appropriate resource requirement and efficient mapping into a single search problem, and use standard search methods and heuristics to solve it.

We believe that this problem is NP-hard in the general case because a restricted problem is NP-hard in the general case. We view the mapping of data partitions and workloads to VMs given the objective of minimizing cost as similar to a three-dimensional matching problem with the

objective of maximizing matches. The later is an optimization problem and NP-hard [5]. We leave the detailed discussion for the mapping complexity to our dissertation.

3 Framework

We envision an autonomic and generic framework for resource provisioning and workload execution. The high-level architecture of a workload management framework is shown in Fig. 1. We provide a high-level description of the framework components and their interactions below. We identify four major parties in the framework: (a) a client, (b) a manager, (c) the storage and execution resources, and (d) an image and a data repository. A client has some application workloads to execute. The manager supervises the workload execution. The processing resources are booted with *settings* retrieved from the repository. The storage resources get a copy of the data from the repository. Both processing and storage resources are combined to provide an execution platform. The workloads are executed on a number of execution platforms.

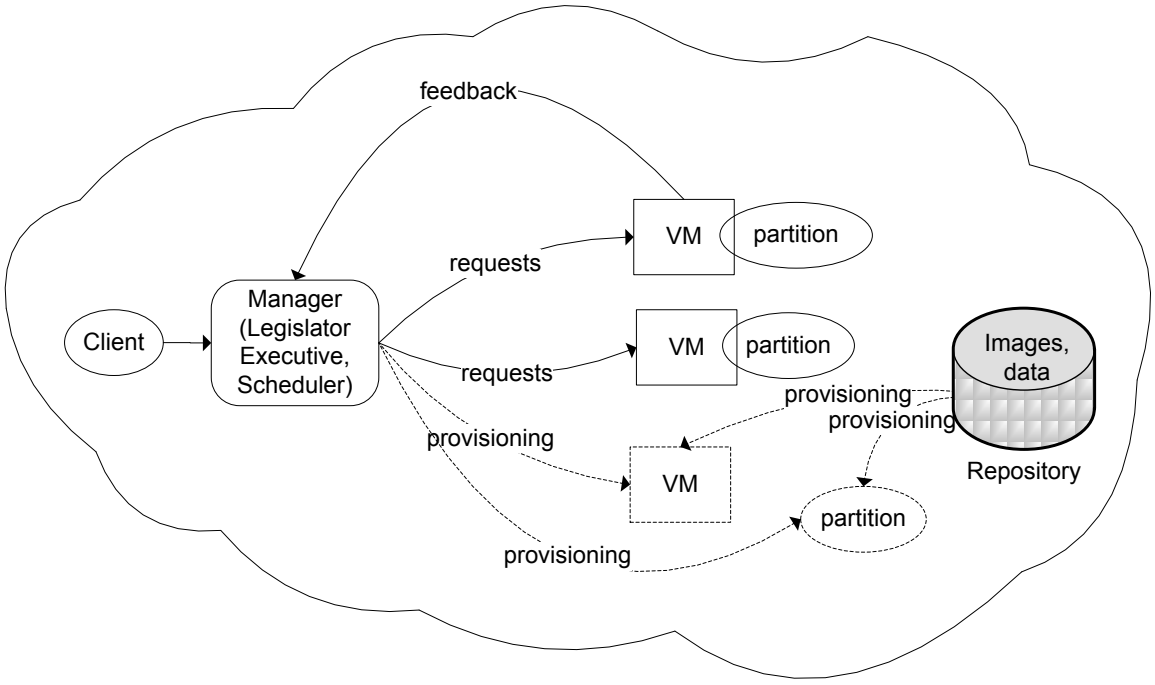

Figure 1. Architecture of a workload management framework in a cloud.

A client submits a set of workloads to the manager. The manager consists of three components (a) a legislator, (b) an executive, and (c) a scheduler. The legislator is the focus of our research. It determines a suitable configuration of storage and processing resources as well as an efficient mapping to execute the workloads to meet an objective. This configuration is then passed to the

implementing process or the executive. The executive instantiates the mapping by provisioning appropriate execution platforms. It allocates the processing resources (VMs) and attaches data partitions to the VMs as required by the configuration. In addition, the executive creates replicas of the partitions if needed. Once the executive finishes, the scheduler uses the mapping to submit the requests of the workloads to the appropriate execution platforms as required by the configuration, and the workload execution begins. With workload bound unknown a priori, we use a simple First-Come-First-Serve (FCFS) scheduling policy to dispatch the requests to appropriate execution platform.

As the workloads executes, some feedback is sent back to the manager periodically. The feedback may include health status pings or execution times. The manager may suggest a new configuration based on the feedback. Revisions to the current configuration may be necessary due to a number of reasons such as excessive SLA violations, or a change in the number or type of workloads. If the deployed configuration is revised, the executive and the scheduler respectively adjust the resources and dispatch the workloads' requests according to the new configuration. The suitable opportunity for implementing revisions is at every time-unit (say an hour) because: (a) the manager deals with the average behaviour of the system rather than a particular instant, and (b) the cloud resources are typically metered by the hour.

Determining a suitable configuration: We represent the set of all possible configurations for an application ***A*** as a directed graph ***Configs = (N(A), E(A)).*** The set of nodes, ***N(A)***, and the set of edges, ***E(A)*** are defined respectively as:

N(A) = *{**C** | **C** is a valid configuration for **A**} and*

E(A) = *{(**Ci**, **Cj**) | configuration **Cj** is obtained from **Ci** using a permitted modification},*

An edge (C_i, C_j) in the search space indicates that configuration C_j can be obtained from configuration C_i by applying one of the modifications. Examples of modifications include adding a VM, or upgrading a VM to a more powerful type.

The legislator employs a search algorithm to explore the search space. The high-level architecture of the legislator is shown in Fig 2. Given a set of workloads and an objective, a search algorithm looks for a suitable configuration. At each iteration, the search algorithm

chooses a suitable modification on the current configuration. The modified configuration is evaluated using a *cost model*. The cost model, in turn, employs a *performance model* to predict the expected behaviour of workloads on a modified configuration. The cost model passes a cost value back to the search algorithm. Then, the algorithm decides whether to keep exploring the search space or to flag the evaluated configuration as a suitable one.

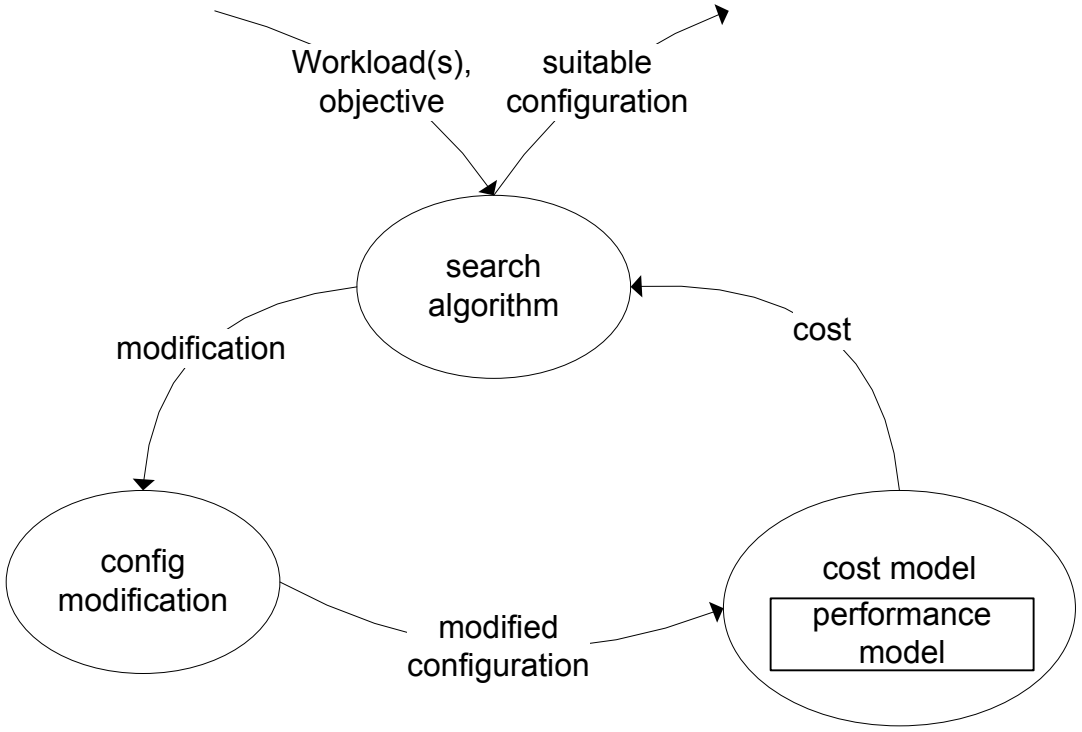

Figure 2. Architecture of the legislator.

The elegance of this architecture is that various search algorithms can be used with various cost models and vice versa. Similarly, different cost models can be used with different performance models. In general, the size of a workload is unknown. Therefore, we argue it is appropriate to calculate the cost of workload execution per unit time, say an hour. Consequently, we parameterize our cost function with a time-unit.

4 Intended Research

We propose to complete the following work to satisfy the requirements for the doctoral thesis.

Performance Model: A performance model is required to forecast the behaviour of a workload execution at a multi-partition data service. Initially, we considered an analytical performance

model, however, we experienced highly inaccurate forecasts in the response times [4]. This is because the analytical models do not capture the complex effects of the concurrently executing requests on the servers [6, 7], which are amplified by the variance in the cloud [4]. Furthermore, the analytical modeling techniques are notoriously hard to evolve with the system. Increasingly the research community is using experiment-driven performance modeling for database management and multitier systems [6-12]. A subset of experiment-driven performance models account for the impact of the interactions among the concurrently executing request types on performance [6, 7, 10-12]. Therefore, we consider an experiment-driven approach in building a performance model.

Our approach consists of three stages: (a) sampling the space of possible request types and their instances in a workload, (b) collecting data by executing possible request mixes or samples, and (c) preprocessing data and building performance models. The possible combinations of request mixes are exponential, so having an effective sampling approach is crucial. Similar to Tozer et al. [6], we sample the N-dimensional space using a Latin Hypercube Sampling (LHS) protocol [13], where each request type is a dimension. The LHS protocol is a variant of stratified sampling. This protocol significantly reduces the number of experiments needed while providing a good coverage of the possible request mixes. The lower bound on the number of request instances of any request type in a workload is zero. Meanwhile, we limit the aggregate number of instances of all the possible request types in a workload by the optimal Multiprogramming Level (MPL) value of a VM type. Consequently, we discard any samples whose aggregate request instances exceed the optimal MPL value of a VM type. We consider both theoretical approaches (p-value type analysis), and empirical approaches (gains in accuracy) to determine the appropriate number of samples. Schad et al. [14] notice large variance in the clouds. Therefore, we narrow the scope of a performance model to a particular VM type to increase the accuracy of the performance model.

Once the samples are obtained, we execute them in a public cloud for each VM type. Both the client and the data-service exist in a public cloud. The buffer pool of the data service is partitioned in proportion to the number of partitions. Each sample is executed for some time (say around 10 minutes). The request mix remains constant throughout the execution of the sample. The client collects run time statistics (such as throughput and response times) for each request

type in the samples. After all the samples execute, we analyze their execution results to see any unusual behaviour, like outliers in the response time and remove them if appropriate. Then, we feed the samples and their results to learning algorithms such as linear regression (LR) or support vector machine (SVM). Finally, we validate the performance model against new data (discussed in Section 5).

Cost Models: 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 a cloud that offers its infrastructure-as-a-service (IaaS): (a) compute, (b) storage, and (c) network. The cost function for the pay-as-you-go pricing scheme is stated as:

$(configuration) = $(compute) + $(storage) + $(network) + $(penalties),

where $ represents dollar-cost of resource usage, penalties or the configuration. This is also the objective function, which needs to be minimized. We find that our current cost model [4] is adequate for data analytics workloads operating over a single partition. However, in this model, we assume a constant cost for network, provide partial storage cost, and we account for penalties only over response times. We need to extend this model to account for *any workload type* (analytical, transactional or mixed) and model costs for all the resources used.

The method used in building the performance model is a typical experiment-driven approach, though the sampling and the underlying regression technique may vary depending on the scope and/or the accuracy required. We believe that the same approach can be used for building and validating our cost model. Unfortunately, it is difficult to collect measurements for large training and test sample sets in a pubic cloud. This is because executing large number of samples sets in a public cloud would render hefty bills. This is aggravated by the long execution time (1 hour) for each workload. Also, setting up each sample execution is manual, unlike for the performance model where we leverage existing tools to automate the sample execution. Fortunately, the VM and the storage costs can be estimated fairly accurately using the published unit resource costs. However, we still need to determine the communication or the network cost experimentally. Therefore, we employ a *hybrid* of an analytical and an experiment-driven model. The costs of VMs, storage and penalties are determined analytically, while the cost of network usage is

estimated based on experimentation. We further discuss each component of the cost function below:

- *Cost of Virtual Machines:* We consider the VM as a compute unit in our cost model. Fortunately, VMs are typically metered by the hour in the pay-as-you-go scheme. Any partial usage is rounded up to the next hour. The cost of virtual machines in a configuration becomes equivalent to:

$$\$(compute) = \left\lceil \sum_{v \in V} \$v \right\rceil$$

  Where V is the set of VMs in the configuration C, and $v is the hourly of cost of a VM v.

- *Cost of storage:* There are different types of storage with different properties and prices. For example, Amazon S3 [15] stores data over multiple network devices. It is a key-value storage that can store a value sized from 1 byte to 5 terabytes. Similarly, DynamoDB [16] is a key-value storage with very low latency but places a limit (of 64kb) on the size of the key and the value.

  We consider that our data partitions are stored on a network-disk type device, which is metered by the month. We prorate the monthly cost down to an hour. The hourly costs for the storage used in the configuration is estimated as:

$$\$(storage) = \left\lceil \frac{\$q \times \lceil E \rceil}{month\_hours} \right\rceil$$

  where $q is the unit cost of storage (in dollar per gigabyte per month), $\lceil E \rceil$ is the aggregated size of data partitions rounded up to the next gigabyte, and month_hours is the number of hours in a month (30days × 24hours). Any fractional cost is rounded up to the next cent.

- *Cost of network usage:* We see two main classes of network costs: (a) cost of transferring data in and/or out of clouds, and (b) network costs for accessing storage. Transferring data from user premises to the cloud is typically a one-time job, and incurs a one-off cost. The data in the cloud may persist indefinitely. While our cost model accounts for the hourly

storage cost, we assume that the data already exists in the cloud. We estimate the network costs for accessing storage as:

$$\$(network) = \left\lceil \sum_{v \in V} c_v \times \$s \right\rceil$$

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

- *Cost in penalties:* We propose a function that assigns a penalty each time-unit in which a breach occurs. This is reasonable since we use average performance metrics (such as average throughput) over a time-unit to detect a breach. For a particular configuration C and a request type q, the penalty incurred in a given time-unit (hour) is given by

$$P_q\ (C) = pcond(q, C) \times penalty\ (q)$$

where *penalty(q)* is the penalty value (in $) for request type q. The binary function *pcond* tells us if an SLO defined over q and C has been violated. For example, *pcond* defined over throughput for transaction q and a configuration C can be written as

$$pcond\ (q, C) = \begin{cases} 1\ if\ avgThroughput\ (q, C) \leq threshold(q), \\ 0\ otherwise \end{cases}$$

where *avgThroughput(q,C)* is the predicted average throughput for transaction *q* in *C,* and *threshold(q)* is the required throughput specified in the SLO for the transaction *q*.

Space Search: The problem of finding the global optimum or the least costly configuration is NP-hard in the general case, as we stated in Section 2. The space of possible configurations is very large and heuristics must be used to prune the search space. We have developed algorithms that search for the minimal dollar-cost configuration. These algorithms consist of: (a) greedy, (b) adaptiveGreedy, and (c) variants of tabu search. The greedy algorithm serves as a baseline for comparing the performance of the algorithms. Tabu search and its variants are used to explore paths that the greed-based algorithms are unlikely to explore. We discuss them further below.

- *Greedy heuristic*: The greedy search algorithm starts by building a configuration from the least expensive VM type. It then greedily selects the lowest cost modification amongst the available modifications. As a possible consequence, the cost of a configuration decreases due to reduced penalties, for example. The algorithm terminates when no modification results in a lower cost configuration. This algorithm stops at the first minimum cost configuration it finds.

- *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. For example, if the adaptive greedy algorithm finds the first minimum in n iterations then it explores the search space a further $2n$ iterations in the hope of finding a better (less costly) minimum. If one is found then it resets the iteration counter and continues to look for a better minimum until one is not found in the additional $2n$ iterations.

- *Tabu greedy:* The tabu greedy algorithm is an extension to the adaptive greedy heuristic. In tabu greedy each chosen modification is intentionally flagged unavailable (tabu'ed) for some iterations despite being a perfectly eligible modification. It does not use additional tabu constructs (e.g. recency, quality, frequency) in deciding which modification to select.

- *Tabu search:* We present a tabu search algorithm which uses additional tabu constructs to select the modifications more intelligently. The algorithm uses intensification and diversification strategies. The intensification strategies promote the selection of modifications 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 unchosen modifications, or replaces the busiest VM with a more powerful VM type.

The above search algorithms vary in their sophistication and their ability to find suitable configurations. However, the algorithms do not tell us whether a suggested configuration is the least costly configuration or whether there exist better configurations that were not yet discovered. Ironically, the algorithms cannot tell us that they have found a global minimum even

when they do. In contrast, mathematical methods like linear programming (LP) are able to determine the provably optimal configuration given an objective. In this case, the problem shifts from search to formulation. Ruiz-Alvarez et al. use LP for optimal placement of data in the hybrid clouds [17]. We intend to explore LP for finding the cheapest configuration. The time complexity for solving LP is still exponential in the worst-case [18].

5 Evaluation

We claim to provide a novel and generic legislator framework for determining appropriate resources and generating an efficient mapping needed for executing data-intensive workloads in the clouds. The evaluation of the framework will be done incrementally, since there is a dependency between individual components. For example, the search methods depend on the cost model, which in turn depends on the performance model. Each objective function may require a different cost model. For example, the makespan objective requires a time-bound cost model instead of dollar-cost model, though both may use the same performance model. Similarly, different performance models can be used to increase accuracy or reduce building effort. This involves a considerable amount of work since each of the modules represents a large area of research. For our Ph.D. dissertation, we evaluate our work with a single objective function: minimal dollar-cost for executing transactional/analytical/mixed workloads that access multiple partitions. Other objective functions and associated cost and performance models are beyond the scope of this thesis.

Tenant Databases and their Workloads: We instantiate and evaluate our models and methods for the Amazon cloud. Amazon is the dominant vendor in the IaaS cloud market. We have used different analytical workloads in our recent work [4]. We explore *any type* workloads, which change in the request types (transactional vs. analytical) or their number, or in their SLOs. While Cooper et al. [19] define realistic workloads for *unstructured* data, we define workloads over *structured* data residing in different partitions or tenant databases.

We use databases from well-known benchmarks as tenant databases in evaluating our work. We consider databases of two transactional benchmarks (TPC-C [20] and TPC-E [21]), and a database of an analytical benchmark (TPC-H [22]). Our workloads consist of a mix of queries and transactions from the stated benchmarks. All the requests in a workload execute concurrently until the specified time. Further, a request type in a workload may also have multiple instances

that execute concurrently. A request instance is continuously re-submitted if finished prior to the end of the specified time. This ensures that the request mix is present at the data service throughout the time bound.

Performance model: We choose three VM types with the intention of demonstrating under, over, and optimal resource setup. We choose: *small*, *large*, and *xlarge*. They vary in their price, processing power and their capacity to hold data in memory. For example, all the three tenants fit in the memory of xlarge but neither fit in the memory of small, and only two tenants fit in the memory of large.

We use the LHS protocol to generate two sets of samples (around 300), one for training and one for validation. We execute both sets in the Amazon cloud using separate VMs and clients. We measure throughput and response time metrics. We train our performance model using one sample set and its results. Using the trained performance models, we predict the metrics for the second sample set. After that, we compare the predicted metrics against the measured metrics for the second sample set.

We use the popular metrics used in the existing literature for comparison: correlation coefficient, average and median prediction errors. Correlation quantifies the similarity between the actual and modeled trends. Meanwhile, prediction errors quantify the gap between the predicted and the measured values. Correlation coefficient and prediction accuracy are complementary, and we use both. Based on existing literature [6, 7, 12], we consider high correlation coefficient (around 0.80 or above) and low prediction errors (around 20% or below) to flag the success of our performance model.

The cloud environments have as much as 35% variance compared to a local server [14]. The variance poses a great challenge in building a performance model. Existing performance models [6-8] are built for a physical server or a VM hosted on a local server. Further, Sheikh et al. [7] developed performance models that can provide predictions for unknown VM or request types, and have the ability to adapt online. In contrast, our performance models [23] are built for the highly variant cloud environments. We reduce variance in building a performance model at the cost of limiting the scope of the model to the known VM and request types. Further, our performance models are entirely built offline. We believe we are the first to present performance

models for a public cloud, so there is no comparison points in clouds. We also provide a comparison of different underlying prediction techniques based on accuracy, and justify our choice. Our comparison includes multi-attribute linear regression, gaussian processes, multi-layer perceptron and support vector regression.

Cost Model: We instantiate our cost model for the Amazon cloud. The method used for validating performance model is a typical validation method for experiment-driven approach. We have already discussed the limitations in executing large sample sets for approximating cost function.

Instead, we narrow the load diversity of workloads (see Table 1). The workloads execute at the optimal MPL level of a VM type. All workloads are presently weighted equally. Therefore, the optimal MPL of a VM type is divided equally amongst the multiple workloads executing together. Any remainder MPL is used by Q1 or Q12 instances. This is an ad hoc choice to keep the data service under optimal load. For example, suppose read-only and write-heavy workloads execute on the large VM type, where the optimal MPL level is 75. There are four request types in the read-only and write-heavy workloads (Q1, Q6, trade-order and trade-update), each getting an equal MPL share (of 18) of the optimal MPL value (75). The remainder of 3 adds into the MPL share of Q1. The MPL share of a request type represents the concurrent instances of that request type in the request mix executed at a data service at any time.

Table 1: Various workloads defined over multiple request types.

Workload type	Tenants	Request types
A – Read only (OLAP)	TPC-H	Q1, Q6
B – Write heavy (OLTP)	TPC-E	trade-order, trade-update
C – Read-write (mixed)	TPC-H, TPC-C	Q12, Q21, new-order, payment

Further, we validate our cost model using sensitivity analysis over different values of the user-controllable variables that determine a configuration cost [24]. A user has direct control over three variables that impact the cost of a configuration in the Amazon cloud. These variables are: (a) workloads, (b) VM types and (c) SLOs' specifications. The network cost varies with the workload and the VM type, while the storage cost varies with the tenant type.

The cost itself has two components (a) the resource costs, and (b) the penalty costs. We can compare the estimated resource cost directly against the invoice rendered by Amazon. However, a cloud provider like Amazon is typically agnostic to the workloads or the associated SLOs. In this case, we calculate the *actual* penalties based on the measured (throughput and response time) metrics, and compare these against the penalty costs calculated on predicted metrics. Then, we calculate the error in cost estimate for each use-case. Low estimation errors (20% or less) in most use-cases mark the success of our cost model. Finally, we discuss our cost model in relation to others in the literature [25-27].

Search Algorithms: Once the cost model has been validated, the search algorithms can use it for finding the most cost effective configuration. We measure the performance of the search algorithms on two quantities: (a) the dollar-cost of the suggested configuration, and (b) the execution time of the algorithm. We compare the performance of the algorithms against the cost of a baseline configuration, against each other, and against globally optimum configurations in some limited cases. The baseline configuration is provided by a greedy or a random search. The globally optimum configurations are worked out using *linear programming* in a few cases. In such cases, we will see how close our search algorithms come to the optimal case.

We validate the results of linear programming formulations against the optimum configurations worked out by hand. For example, the configuration with one small VM instance is optimal in two cases: (a) when there are no SLOs, and (b) when all the SLOs cannot be satisfied by any combination of any VM types. In these cases, the dollar-cost has to match verbatim though the number and types of VMs may vary, for example.

6 Expected Contributions

We formulate the problem of determining appropriate resources and generating an efficient workload mapping, and the constructs to represent it. In traditional workload execution literature, the resource pool is assumed static. In contrast, we extend the execution constructs to include provisioning resources prior to and during execution. An appropriate number of processing and storage resources depend on a suitable configuration. We combine the task of determining appropriate resources and workload mapping into a single search problem, and use standard search methods and heuristics.

We provide a systematic study of workload management of data-intensive workloads in the clouds. We develop a taxonomy of workload management techniques used in the clouds, and classify existing workload management mechanisms based on the taxonomy. Further, we provide a survey of workload management systems in the clouds and a discussion of possible directions for future research in this area.

We propose a novel and generic legislator framework for determining appropriate resources and efficient workload mapping in a cloud. The management function in the framework exploits the cloud's elasticity. Our framework allows pluggable cost and performance models.

We instantiate the framework for a public cloud by developing various search algorithms to find a suitable configuration given an objective function. Further, we develop an objective function and associated cost model. We also develop a performance model to support the cost model.

We integrate dollar-cost with workload execution using our problem formulation and appropriate objective function. We hope our work to be useful from a number of angles: (a) estimate the expense of executing a workload in a cloud, (b) offer scale to workload execution by harnessing cloud's elasticity, (c) reduce time to result by exploiting rapid provisioning of cloud's resources, and (d) shorten the gap between data growth and the processing ability.

Plan

Milestones	**Status**	**Estimate (term)**
formulate the problem of determining appropriate resources and generating an efficient workload mapping	Completed [1, 4]	
Legislator framework	Completed [1, 4]	
Systematic study of workload management in clouds	Completed [2, 3]	
Cost models	Completed [4, 24]	
Analytical performance model	Completed [4]	
Experiment-driven performance model	Completed, submitted for publication [23]	Fall 12
Search algorithms	Completed development, pending validation	Winter 12
LP formulation	Currently developing	Winter 12
Thesis	Planning	Fall 13

Dissertation Outline

1 Introduction and motivation

1.1 Ever growing data and limitations in processing
1.2 Introduction to cloud computing and its offerings
1.3 Relevance of workload management
1.4 Research statement
1.5 Scope of the Research
1.6 Thesis organization

2 Background

2.1 Workload management in traditional dbms and grid computing
2.2 Data-intensive computing architectures
2.3 Data management applications in the cloud
2.3.1. Opportunities and limitations
2.4 Workload management taxonomy in clouds
2.4.1. Scheduling taxonomy
2.4.2. Provisioning taxonomy
2.5 Survey of workload management techniques and systems

3 Overview

3.1 Provisioning problem and objective functions
3.2 Generic search, generate and evaluate loop
3.3 Search methods, cost and performance models
3.4 Autonomic framework

4 Performance models

4.1 Analytical models (Queuing Network Models)
4.1.1. Modelling and evaluation
4.2 Experiment-driven models (Support Vector Machine)
4.2.1. Modelling and evaluation
4.3 Comparison with existing work

5 Cost models

5.1 Generic dollar-cost model
5.2 Instantiating cost model for Amazon cloud
5.3 Workload definitions
5.4 Various use-cases demonstrating optimal, under and/or optimal configurations
5.5 Validating cost model against use-cases
5.6 Comparison with other cost models

6 Search methods and linear formulation

6.1 Complexity of the search problem
6.2 Greedy heuristics
6.3 Tabu search
6.4 Evaluating algorithms
6.4.1. Comparing algorithms against a baseline and against each other
6.4.2. Comparing algorithms against the global optimum in limited cases
6.5 Linear formulations for optimal configuration and validation
6.6 Comparison with other search methods

7 Thesis contribution and future work

8 Conclusions

References

[1] R. Mian and P. Martin, "Executing data-intensive workloads in a Cloud," *Proc. 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, 2012, pp. 758-763.
[2] R. Mian, P. Martin, A. Brown and M. Zhang, "Managing Data-Intensive Workloads in a Cloud," *Grid and Cloud Database Management*, G. Aloisio and S. Fiore, eds., Springer, 2011.
[3] 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. http://research.cs.queensu.ca/TechReports/Reports/2011-580.pdf
[4] R. Mian, P. Martin and J.L. Vazquez-Poletti, "Provisioning data analytic workloads in a cloud," *Future Generation Computer Systems (FGCS)*, 2012, pp. in press http://dx.doi.org/10.1016/j.future.2012.1001.1008
[5] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman Co., 1979, p. 338.
[6] S. Tozer, T. Brecht and A. Aboulnaga, "Q-Cop: Avoiding bad query mixes to minimize client timeouts under heavy loads," *Proc. IEEE 26th International Conference on Data Engineering (ICDE)*, 2010, pp. 397-408.
[7] M.B. Sheikh, et al., "A bayesian approach to online performance modeling for database appliances using gaussian models," *Proc. 8th ACM international conference on Autonomic computing (ICAC)*, ACM, 2011, pp. 121-130.
[8] A. Ganapathi, et al., "Predicting Multiple Metrics for Queries: Better Decisions Enabled by Machine Learning," *Proc. IEEE 25th International Conference on Data Engineering, 2009. (ICDE '09).* , IEEE, 2009, pp. 592-603.
[9] C. Gupta, A. Mehta and U. Dayal, "PQR: Predicting Query Execution Times for Autonomous Workload Management," *Proc. International Conference on Autonomic Computing, 2008. (ICAC '08).* , IEEE, 2008, pp. 13-22.
[10] M. Ahmad, A. Aboulnaga and S. Babu, "Query interactions in database workloads," *Proc. Proceedings of the Second International Workshop on Testing Database Systems*, ACM, 2009, pp. 1-6.
[11] M. Ahmad, A. Aboulnaga, S. Babu and K. Munagala, "Modeling and exploiting query interactions in database systems," *Proc. Proceedings of the 17th ACM conference on Information and knowledge management*, ACM, 2008, pp. 183-192.
[12] M.B. Sheikh, et al., *A Bayesian Approach to Online Performance Modeling for Database Appliances using Gaussian Models*, Technical Report#: CS-2011-13, 2011. http://www.cs.uwaterloo.ca/research/tr/2011/CS-2011-13.pdf
[13] C.R. Hicks and K. Turner Jr, *Fundamental concepts in the design of experiments*, Oxford University Press, New York, 1999.
[14] J. Schad, J. Dittrich and J.-A. Quiane-Ruiz, "Runtime measurements in the cloud: observing, analyzing, and reducing variance," *Proc. VLDB Endow.*, vol. 3, no. 1-2, 2010, pp. 460-471
[15] Amazon, "Simple Storage Service (S3)," http://aws.amazon.com/s3/.
[16] Amazon, "DynamoDB," http://aws.amazon.com/dynamodb/.
[17] A. Ruiz-Alvarez and M. Humphrey, "A Model and Decision Procedure for Data Storage in Cloud Computing," *Proc. Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium on*, 2012, pp. 572-579.

[18] G. Leach, *Topic 11 Computational Complexity*, COSC 1229/1479 Computational Science 1, MIT, 2012. http://goanna.cs.rmit.edu.au/~gl/teaching/cs554/lectures/554-Complexity-4.pdf
[19] B.F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan and R. Sears, "Benchmarking cloud serving systems with YCSB," *Proc. 1st ACM Symposium on Cloud Computing (SoCC)*, Association for Computing Machinery, 2010, pp. 143-154.
[20] TPC-C, "Order Processing Benchmark," http://www.tpc.org/tpcc/.
[21] TPC-E, "Trading Benchmark," http://www.tpc.org/tpce/.
[22] TPC-H, "Decision Support Benchmark," http://www.tpc.org/tpch/.
[23] R. Mian, P. Martin and J.L. Vazquez-Poletti, "Towards Building Performance Models for Data-intensive Workloads in Public Clouds," *Proc. 4th ACM/SPEC International Conference on Performance Engineering (ICPE)*, ACM, 2013, pp. submitted.
[24] R. Mian, P. Martin, F. Zulkernine and J.L. Vazquez-Poletti, "Estimating Costs of Data-intensive Workload Execution in Public Clouds," *Proc. 10th International Workshop on Middleware for Grids, Clouds and e-Science (MGC) in conjunction with ACM/IFIP/USENIX 13th International Middleware Conference 2012*, ACM, 2012, pp. in press.
[25] T. Bicer, D. Chiu and G. Agrawal, "Time and Cost Sensitive Data-Intensive Computing on Hybrid Clouds," *Proc. 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, 2012, pp. 636-643.
[26] B. Sharma, R.K. Thulasiram, P. Thulasiraman, S.K. Garg and R. Buyya, "Pricing Cloud Compute Commodities: A Novel Financial Economic Model," *Proc. 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, 2012, pp. 451-457.
[27] Amazon, "Simple Monthly Calculator," http://calculator.s3.amazonaws.com/calc5.html.