A Generic Auto-Provisioning Framework for Cloud Databases

Jennie Rogers¹, Olga Papaemmanouil² and Ugur Cetintemel¹

¹Brown University, ²Brandeis University
**Introduction**

- Use Infrastructure-as-a-Service (IaaS) to provision virtual machines
  - Rent on demand
  - Pay reservation period and per-request charges
  - Many VM configurations

<table>
<thead>
<tr>
<th>Instance Type</th>
<th>Cost per hour</th>
<th>Sequential Blocks / sec</th>
<th>Seeks / sec</th>
<th>CPU (cores x compute units)</th>
<th>RAM (GB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>m1.small</td>
<td>$0.085</td>
<td>295000</td>
<td>74913</td>
<td>1x1 (1)</td>
<td>1.7</td>
</tr>
<tr>
<td>m1.large</td>
<td>$0.34</td>
<td>604000</td>
<td>72330</td>
<td>2x2 (2)</td>
<td>7.5</td>
</tr>
<tr>
<td>m1.xlarge</td>
<td>$0.68</td>
<td>591333</td>
<td>59327</td>
<td>4x2 (8)</td>
<td>15</td>
</tr>
<tr>
<td>c1.medium</td>
<td>$0.17</td>
<td>654000</td>
<td>82691</td>
<td>2x2.5 (5)</td>
<td>1.7</td>
</tr>
<tr>
<td>c1.xlarge</td>
<td>$0.68</td>
<td>681000</td>
<td>57511</td>
<td>8x2.5 (20)</td>
<td>7</td>
</tr>
</tbody>
</table>

- Complex decision space
DBMS on the Cloud

• DBMS a natural fit for the cloud
• Complex to configure
• Very resource- and time-varying workloads
• Can require expensive hardware
  – High up-front expenses, maintenance
• Historically have been over-provisioned
  – Pay-as-you-go afford the user significant flexibility
• Example: data warehousing, scale up at night, pay less by day
Problem Statement

• Optimize for monetary cost as a first-class concern

• Provision to meet user-specified QoS requirements (latency, throughput)
  – Defined on a per-query-class basis

• Gives users the opportunity to trade off between QoS and $

• Provisioner reserves and releases machines as needed
Experimental Configuration

- Proof-of-Concept implementation on Amazon Web Services
- Used PostgreSQL
  - Hand-configured with indexes and accurate buffer pool size
  - Modified to display QEP costs in terms of I/Os and CPUs
- Used Elastic Block Storage volumes
- Instrumented resources with iostat and procfs
- TPC-H queries 1-5, scale factor 10
Isolated Options, TPC-H Query 1

<table>
<thead>
<tr>
<th>Instance Type</th>
<th>Cost per hour</th>
<th>Latency (s)</th>
<th>Cost Per Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>m1.small</td>
<td>$0.085</td>
<td>1268.4</td>
<td>$1.96</td>
</tr>
<tr>
<td>m1.large</td>
<td>$0.34</td>
<td>583.77</td>
<td>$2.24</td>
</tr>
<tr>
<td>m1.xlarge</td>
<td>$0.68</td>
<td>533.69</td>
<td>$2.58</td>
</tr>
<tr>
<td>c1.medium</td>
<td>$0.17</td>
<td>620.51</td>
<td>$2.04</td>
</tr>
<tr>
<td>c1.xlarge</td>
<td>$0.68</td>
<td>603.12</td>
<td>$2.58</td>
</tr>
</tbody>
</table>

• Complicated; non-linear
  - Need to estimate what is worth paying for
## Concurrency Options, Query 2

<table>
<thead>
<tr>
<th>Queries / Hour</th>
<th>Latency Requirement</th>
<th>Cheapest Configuration</th>
<th>Cost / Query</th>
<th>Cost / Hour</th>
</tr>
</thead>
<tbody>
<tr>
<td>720</td>
<td>30</td>
<td>c1.xlarge</td>
<td>$0.00344</td>
<td>$2.47590</td>
</tr>
<tr>
<td>360</td>
<td>30</td>
<td>m1.large</td>
<td>$0.00120</td>
<td>$0.43109</td>
</tr>
<tr>
<td>240</td>
<td>20</td>
<td>m1.large</td>
<td>$0.00145</td>
<td>$0.34912</td>
</tr>
<tr>
<td>180</td>
<td>20</td>
<td>m1.large</td>
<td>$0.00192</td>
<td>$0.34508</td>
</tr>
<tr>
<td>144</td>
<td>15</td>
<td>c1.xlarge</td>
<td>$0.00474</td>
<td>$0.68300</td>
</tr>
<tr>
<td>120</td>
<td>15</td>
<td>c1.xlarge</td>
<td>$0.00569</td>
<td>$0.68260</td>
</tr>
</tbody>
</table>

- Workload varies
  - Queries / hour
  - QoS
- No one-size-fits-all solution
QoS vs. Monetary Cost Trade Off
General Solution Flow

**Inputs**
- Query-related data
- Workload specifications

**Prediction Engine**
- Convert inputs into predictions
- Estimate query performance
- Formulate constraints
- Apply solver

**Deployment Plan**
- Q1: 2 large instances
- Q2: 1 medium instance
- ...

...
Concrete Problems

• Provisioning
  • When to deploy virtual machine instances?
  • How many? What type? EBS or not for storage?
    • When to declare overload and launch new instances? When to scale down?

• Scheduling
  • How to route queries to deployed instances?
  • Combine above into continuous allocation of instances

• Our focus: modeling query behavior to create deployment plans for a workload
Overview

• System model
• Predictor input options
• Three approaches:
  – White Box (optimizer estimate-based)
  – Gray Box (semantic info and profiling data)
  – Black Box (profiling-based)
• Related Work
• Open Questions
System Model

Data Management Application

Performance goals
Target Workload

Constraint Solver

Workload profiler

Cloud Profiler

VM provisioning & monitoring
Pricing Model
VM specs

Cloud Provider

VM Type 1
VM1
CPU1
MEM1
...

VM2
CPU2
MEM2
...

VM Type 2
VM3
CPU3
MEM3
...

VM4
CPU4
MEM4
...

DB replicas
## Input Options

<table>
<thead>
<tr>
<th>White Box</th>
<th>Gray Box</th>
<th>Black Box</th>
</tr>
</thead>
<tbody>
<tr>
<td>Estimate resource requirements from query execution plans</td>
<td>Use semantic data from query plans and profiling data to build performance predictions</td>
<td>Use a set of profiling-generated configuration options</td>
</tr>
</tbody>
</table>

- Provide different options for predictor data
Bin Packer

- Convert resource consumption to a rate w/QoS
  - Sum dimensions
  - Enforce constraint to assure enough resources
- Can add constraints
  - Adhere to expressive predictive modeling
  - I.e., domain-specific policies, semantic groups
**White Box Provisioning**

- Use benchmarking for each instance type
  - Estimate availability of individual resources (CPU rate, I/O rate)

- Estimate query resource usage from modified PostgreSQL explain statements
  - Breaks estimator costs into CPU and I/O time
  - Cost units ~time of one sequential page read
  - Correct it with a scale factor to convert time into units consumed (CPU cycles and I/O requests)
White Box Provisioning

Query Work Vectors
(q₁, I/Os₁, cpu₁, ram₁)
(q₁, I/Os₁, cpu₁, ram₁)
(q₂, I/Os₂, cpu₂, ram₂)

Cloud Resource Vectors
(m₁.small, I/Os, CPUs, ram, $)
(m₁.large, I/Os, CPUs, ram, $)
(c₁.medium, I/Os, CPUs, ram, $)

Workload Specification
Q₁: 110s, 25 / hr
Q₂: 30s, 60 / hr
Q₃: 240s, 10 / hr

Constraint Solver
Sum of (I/O || CPU) allocated <= available
All workload queries provisioned
One query class / instance
All vector RAM <= VM RAM
Objective: minimize cost

Deployment Plan
Q₁: 2 large instances
Q₂: 1 medium instance
...
White Box Scheduling

• Create online bin packer for each instance provisioned
  • Aware of what queries are currently assigned to instance
• Create new work vector for each incoming query
• Select instance with most available resources provisioned for its query class
Gray Box Provisioning

• Extend white box approach
  – Use profiling information

• Create scale factors for query plan steps
  – Discount shared steps
  – Penalize discrete steps

• Take into account aggregate magnitude of resources
Discount and Penalty Rates

-> HashAggregate
   (cost=1696603..1696603) (actual time=510736..510736)

   -> Seq Scan on lineitem
      (cost=0..960131) (actual time=25..115192)

• Classify steps as shared by the mix or not
• Use explain analyze to determine actual run time and optimizer cost for each step
• Create two plots based on classification and perform power regression
Black Box Provisioning

• Create profiles of different input rates and query classes on each instance type
  • Record latency PDF and cost per query
• Take in workload specifications for QoS, percentile and query mix
• Quantify min-cost configurations to complete a workload
  – Use constraint solver
Black Box Provisioning

Performance Vectors
- Q1: large, 18 / hour, $0.30, pdf
- Q1: small, 7 / hour, $0.20, pdf
- Q2: medium, 16 / hour, $0.10, pdf

Workload Specification
- Q1: 110s, 100%, 25 / hr
- Q2: 30s, 100%, 60 / hr
- Q3: 240s, 100%, 10 / hr

Constraint Solver
All workload queries provisioned
QoS percentile and latency met
Objective: minimize cost

Deployment Plan
- Q1: 2 large instances
- Q2: 1 medium instance
...
Provisioning Options

• White box geared toward:
  • Finer-grained provisioning
  • More variation within query classes
    • Work vectors created during scheduling phase

• Gray Box intended for heterogeneous mixes
  - Allows them to be scheduled together

• Black box better optimized for:
  • Efficiently making plans for larger workloads
  • Has more options for QoS trade off
    • Generates many work vectors for a configuration
Related Work

• Cost and the Cloud

  • Cost as a performance metric (D. Florescu, et al., *ACM SIGMOD Record*, March 2009)
  
  • An example of cost-performance trade-off (E. Deelman, et al., SC '08)

• VM Provisioning

  • Provisioning VMs to meet QoS (P. Shivam, et al., SIGMOD 2007).

  • Amazon AutoScale, Amazon Elastic Load Balance

• Query Performance Prediction

  • Predicting query resource usage with machine learning (A. Ganapathi, et al., ICDE 2009)

  • Predicting the effects of concurrency on QoS (M. Ahmad, et al., CIKM '08)
Open Challenges

• Mixed workloads
  • Dealing with query contention
  • Both latency and cost

• Probabilistic modeling of resources and queries
  • Uncertainty in cloud performance, specific query interactions, optimizer estimates, etc.

• Dynamically allocating the system
  – Form a control loop
Open Challenges (cont'd)

• Fine grained scheduling modeling
  • Learn about how the DBMS actually allocates resources

• Multiple payment strategies
  • On-demand instances vs. reserved
  • Spot provisioning
Thank You!
Challenge: Query Latency Over Time (m1.large)

- Model uncertainty?