inteliScaler

Workload and Resource Aware, Proactive Auto Scaler for PaaS Cloud

Paper #10368
RS Shariffdeen, UKJU Bandara, DTSP Munasinghe, HS Bhathiya, and HMN Dilum Bandara

Dept. of Computer Science & Engineering, University of Moratuwa, Sri Lanka
Introduction
Autoscaling at PaaS Level

- **Reactive Nature**
  - Inability to adapt to complex workload patterns
  - Not considering the execution time (Spin Up/Down time)
- **Rule-Based System**
  - Event triggered system to manage on-demand resources
- **User Involvement**
  - User is required to have a deeper understanding of the domain to utilize the auto scaling mechanism, define rules, and threshold
Research Goal

Autoscale computing resources in a PaaS cloud environment based on current workload and resource usage, while predicting the workload to reduce cost and meet desired QoS/SLA goals.

Research Contributions

- Performance study of an existing auto scaler
- A workload forecasting technique
- A penalty-based proactive scaling method
- Evaluation of feasibility of smart killing on various IaaS providers
- Analysis of different combinations of scaling methods
- Implementation of proposed combination on Apache Stratos and AWS
Our Solution

Key features

- Proactive
- Better predictive ability
- Better resource utilization
- Considering resource acquisition costs and service degradation penalties

Evaluation

- Against Apache Stratos
Analysis of Stratos Auto Scaling Performance
Analysis of Stratos Auto Scaling Performance

**Workload**

**RUBiS Workload**

![RUBiS Workload Graph](image1)

**Moraspirit Workload**

![Moraspirit Workload Graph](image2)
Prediction Evaluation

Actual workload

Apache Stratos prediction using $S = ut + 0.5at^2$
Resource Comparison

Analysis of Stratos Auto Scaling Performance

Workload Analysis

High Threshold

Total Instances Spawned = 12

Low Threshold

Total Instances Spawned = 14

Actual Workload  Resource Allocation
Workload Prediction
Base Model Selection

- **ARIMA** - Linear model with ability to capture seasonal behavior
- **Neural Network** - Nonlinear model which is data driven and adaptable
- **Exponential Model** - Nonlinear exponential models have no counterparts in ARIMA
- **Naive Prediction** - As an error correction step
Ensemble Technique

- Ensemble forecast for (t+h) is weighted average over the individual models.
- Weights are calculated from the inverse error of the forecasts.

\[
\hat{x}(t+h) = \frac{\sum_{i=1}^{k} c_i \hat{x}^{(i)}(t+h)}{\sum_{i=1}^{k} c_i}
\]

\[
c_i = \frac{1}{e(i,t)}
\]

E.g.:

\[
\hat{y}_t = \frac{\alpha y_{t,arima} + \beta y_{t,exp} + \gamma y_{t,nnet}}{\alpha + \beta + \gamma}
\]

\[
\alpha = \frac{1}{error_{T,arima}} \quad \beta = \frac{1}{error_{T,exp}} \quad \gamma = \frac{1}{error_{T,nnet}}
\]
Determination of Weights

Effectiveness of $C_i$ depends on how you use the past forecast error from the $i$-th model.

**Averaged Error over past data**

$$RMSE = \sqrt{\frac{1}{n} \sum_{t=1}^{n} (\hat{x}_t - x_t)^2}$$

**Most recent error**

$$SE_t = (\hat{x}_t - x_t)^2$$

We fitted the past forecasting error in $i$-th model using exponential smoothing model and calculated the contribution coefficients $C_i$ based on the result.
## Comparison of Results

<table>
<thead>
<tr>
<th>Model</th>
<th>Google Cluster</th>
<th>Memory</th>
<th>CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RMSE</td>
<td>MAPE</td>
<td>RMSE</td>
</tr>
<tr>
<td>ARIMA</td>
<td>12.963</td>
<td>0.051</td>
<td>7.238</td>
</tr>
<tr>
<td>Exponential</td>
<td>12.886</td>
<td>0.041</td>
<td>7.005</td>
</tr>
<tr>
<td>Neural net.</td>
<td>12.530</td>
<td>0.036</td>
<td>8.169</td>
</tr>
<tr>
<td>Stratos</td>
<td>19.757</td>
<td>0.116</td>
<td>9.928</td>
</tr>
<tr>
<td>ARMA-based model</td>
<td>12.549</td>
<td>0.069</td>
<td>7.185</td>
</tr>
<tr>
<td>Mean Ensemble</td>
<td>12.099</td>
<td>0.051</td>
<td>7.036</td>
</tr>
<tr>
<td>Median Ensemble</td>
<td>12.059</td>
<td>0.055</td>
<td>7.010</td>
</tr>
<tr>
<td>Proposed Ensemble</td>
<td>11.934</td>
<td>0.027</td>
<td>6.972</td>
</tr>
</tbody>
</table>
Resource Allocation
Scaling Algorithm

\[ C_t(n) = C_{ins} \cdot n + C_{ins} \cdot n \cdot f \left( \frac{v_i}{T} \right) \cdot 100 \]

\[ n_{opt} = \arg\min_{n \in N} n_{\in [\text{min, max}]} C_t(n) \]

\[ T = \text{Total time for prediction} \]
\[ C_t(n) = \text{Total cost for time } T \]
\[ C_{ins} = \text{Cost for an instance} \]
\[ n = \text{Number of instances} \]
\[ v_i = \text{Violation time} \]
Scaling Algorithm: Example Penalty Function
Awareness of Resource Pricing Model

Considerations

- Each IaaS provides many different instance types
- Non-uniform pricing policies
  - AWS hourly billing vs. GCE per-minute billing

Solution

- Cost optimize considering IaaS pricing model separately
  - Smart Killing for AWS
Resource Scaling - Approaches

- Stratos Predicted
- Stratos Blind Killing
- Stratos Smart Killing
- Smoothed Data
- Reactive Blind Killing
- Reactive Smart Killing
- Proactive Blind Killing
- Proactive Smart Killing
Simulation - Cost

<table>
<thead>
<tr>
<th>Model</th>
<th>Blind Killing</th>
<th>Smart Killing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stratos Prediction</td>
<td>34.29</td>
<td>12.35</td>
</tr>
<tr>
<td>Reactive</td>
<td>9.14</td>
<td>4.13</td>
</tr>
<tr>
<td>Proactive</td>
<td>5.53</td>
<td>3.25</td>
</tr>
</tbody>
</table>

Proactive Smart Killing
Violation Percentage After 7 Hours: 9.26%
Violation Cost After 7 Hours: 0.235
Overall Solution
Architectural Design

Overall Solution
Architectural Design
Tests on Mock IaaS
Tests on AWS EC2
Tests on AWS EC2

Stratos

inteliScaler
Summary

Performance Evaluation
  • Current auto scaler implementation in Apache Stratos

Workload Prediction
  • Evaluating current workload prediction techniques
  • Proposing a new ensemble workload prediction technique for PaaS

Resource Allocation
  • Comparison of different combinations of resource allocation methods
  • Proposing a penalty-based resource allocation technique

Overall Solution
  • Implementation and testing on Apache Stratos
Further Improvement

• Defining different penalty functions based on required QoS
• Bidding for Spot Instances on AWS
• Inclusion of a Performance Model
• Support for Heterogeneity
Q & A

ridwan.11@cse.mrt.ac.lk
Thank You
Tests on Mock IaaS

Stratos - Low Threshold

<table>
<thead>
<tr>
<th>allocation</th>
<th>utilization</th>
<th>scale up</th>
<th>scale down</th>
</tr>
</thead>
</table>

Stratos - High Threshold

inteliScaler
Upcoming Sections

Performance Evaluation
  • Current auto scaler implementation in Apache Stratos

Workload Prediction
  • Evaluating current workload prediction techniques
  • Proposing a new ensemble workload prediction technique for PaaS

Resource Allocation
  • Comparison of different combinations of resource allocation methods
  • Proposing a penalty-based resource allocation technique

Overall Solution
  • Implementation and testing on Apache Stratos

Project Summary
Workload Types

- **Nature**
  - Synthetic
  - Empirical

- **Pattern**
  - On-and-Off
  - Growing
  - Stable
  - Sudden Peak
  - Cyclic Bursting
Why Simulation?

• Clouds exhibit varying demands, supply patterns, system sizes and computing resources.

• Cloud users have heterogeneous, dynamic and competing QoS requirements.

• Cloud applications have varying performance, workload, and dynamic application scaling requirements.
**Experimental Platforms**
- CloudSim
- MDCSim
- GreenCloud

**Synthetic Workload Generators**
- Httperf
- Faban
- Rain

**Experimental Workloads**
- 98 World Cup
- Google Cluster Data
- ClarkNet

**Application Benchmarks**
- RUBiS
- RUBBoS
- TPC-W
Weight Calculation

\[ b_{i,t} = \alpha e_{(i,t)} + (1 - \alpha) b_{(i,t-1)} \quad (10) \]

where \( 0 \leq \alpha \leq 1 \)

\[ b_{i,t} = \alpha e_{(i,t)} + \alpha(1 - \alpha)e_{(i,t-1)} + \alpha(1 - \alpha)^2e_{(i,t-2)} + \ldots \]

\[ c_{i,t} = \frac{1}{b_{i,t}} \quad (11) \]
Contributions

• Performance study of an existing auto scaler
• A workload forecasting technique
• A penalty based proactive scaling method
• Evaluation of feasibility of smart killing on various IaaS providers
• Evaluation of various combinations of scaling methods
• Implementation of proposed combination on Apache Stratos

Achievements

Research Papers

• Acceptance of “Adaptive Workload Prediction for Proactive Auto Scaling in PaaS Systems” for International Conference - Internet of Things and Cloud Computing Technologies, Singapore
• Working on another paper for International Cloud Computing Conference - IEEE Cloud 2016, USA

Research Grant

AWS Education Research Grant for 1 year
Apache Stratos

A PaaS framework
- Can be set up on existing IaaS
- AWS, GCE, Azure
- Private cloud support

Why Stratos?
- Open source
- Apache Community support
- Mock IaaS - eliminates need and cost of using actual IaaS resources

Architecture
- Services via cartridge instances (PHP, MySQL, Tomcat, etc.)
- Managed by modules (Cloud Controller, Auto Scaler, etc.)
- Pub-sub messaging
Resource Allocation

Current PaaS Auto Scalers

- User need to define threshold parameters
- Autoscaler unaware of the pricing models of IaaS

Inteliscaler

- Cost and QoS aware scaling algorithm to calculate resource count required
- Pricing model aware resource scaling
Stratos Auto Scaling
Literature - Prediction Techniques: Statistical

**Naïve Prediction**

\[ \hat{y}_{T+h|T} = y_T \]

**Exponential Smoothing**

\[ \hat{y}_{t+1|t} = \alpha y_t + (1 - \alpha) \hat{y}_{t|t-1} \]

\[ \hat{y}_{T+1|T} = \alpha y_T + \alpha(1 - \alpha)y_{T-1} + \alpha(1 - \alpha)^2y_{T-2} + \cdots \]

**Moving Average**

\[ \hat{y}_{T+h|T} = \frac{1}{T} \sum_{t=1}^{T} y_t \]
Autoregressive Integrated Moving Average Model \((p,d,q)\)

A combination of,

**Differencing** \((d)\)

\[ y'_t = y_t - y_{t-1} \]

**Autoregression** \((p)\)

\[ y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \cdots + \phi_p y_{t-p} + e_t \]

**Moving Average** \((q)\)

\[ y_t = c + e_t + \theta_1 e_{t-1} + \theta_2 e_{t-2} + \cdots + \theta_q e_{t-q} \]
Literature - Prediction Techniques: Machine Learning

Autoregressive Neural Networks

Lagged values of the time series can be used as inputs to a neural network.
Challenges In PaaS Workload Prediction

- A PaaS cloud may be used to build different applications with vastly different workload patterns.
- As the workload dataset grows with time, predictive model should evolve and continuously learn the latest workload characteristics.
- The workload predictor should produce results within a bounded time.
- The predictor should produce sufficiently accurate results over a sufficiently large time horizon.
Evaluation of Existing Models - Datasets
### Evaluation of Existing Models - Results

<table>
<thead>
<tr>
<th>Model/Dataset</th>
<th>sunspotarea</th>
<th>euretail</th>
<th>oil</th>
<th>Memory</th>
<th>CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MSE</td>
<td>MARE</td>
<td>MSE</td>
<td>MARE</td>
<td>MSE</td>
</tr>
<tr>
<td>Arima</td>
<td>382.360</td>
<td>1.359</td>
<td>0.524</td>
<td>0.004</td>
<td>55.313</td>
</tr>
<tr>
<td>Exponential</td>
<td>505.750</td>
<td>1.161</td>
<td>0.576</td>
<td>0.011</td>
<td>54.989</td>
</tr>
<tr>
<td>Neural Network</td>
<td>473.924</td>
<td>0.465</td>
<td>1.882</td>
<td>0.006</td>
<td>51.616</td>
</tr>
<tr>
<td>Stratos</td>
<td>546.938</td>
<td>0.965</td>
<td>0.650</td>
<td>0.004</td>
<td>61.807</td>
</tr>
</tbody>
</table>

- Each model performs well on some datasets while performing worse on other datasets
- A single model will not perform well in all online training scenarios
- Different applications on PaaS have different workload characteristics
- Dynamic adjustment of models is required for robust results
Our Simulator

- **Workloads**
  - Synthetic
    - Rain-Toolkit
    - Httpperf
  - Empirical
    - Sports website (Moraspirit Log)
    - 98 World Cup Trace
    - Rackspace Cloud Data (Hosting a HR System)
    - Google Cluster Data

- **Platform**
  - 4 Server Nodes
  - Apache Stratos 4.1 setup on top of AWS

- **Benchmark**
  - RUBiS
Experimental Setup

Server Node Specification
- Intel Core i3 1.6 GHz
- 4 GB DRR3 RAM
- 500 GB Hard Disk
- MySQL 5.5
- Apache 2.4
- PHP 5.5

Client Specification
- Intel Core i5 2.6 GHz
- 4 GB DRR3 RAM
- Rain Tool Kit
- Java 1.7
Analysis of Stratos Auto Scaling Performance

AWS Setup

Our Simulator
Workload
Analysis
Stratos Prediction

\[ s = ut + \frac{1}{2}at^2 \]

- \( s \) = predicted load
- \( u \) = first derivative (eg. first derivative of current load average)
- \( t \) = time interval (eg. one minute)
- \( a \) = second derivative (eg. second derivative of current load average)
Stratos Scaling Process

• Policies
  ▪ Application Deployment Policy
  ▪ Auto-Scaling Policy
    o Load Average Threshold
    o Memory Consumption Threshold
    o Requests in Flight Threshold
Project Summary
Summary

Performance Evaluation
  • Current auto scaler implementation in Apache Stratos

Workload Prediction
  • Evaluating current workload prediction techniques
  • Proposing a new ensemble workload prediction technique for PaaS

Resource Allocation
  • Comparison of different combinations of resource allocation methods
  • Proposing a penalty-based resource allocation technique

Overall Solution
  • Implementation and testing on Apache Stratos
Challenges Faced

- Resource and Infrastructure Cost
- Insufficient sources for Workload Data
- Setting up Apache Stratos Paas Cloud on AWS EC2
Key References


