Paragon: QoS-Aware Scheduling for Heterogeneous Datacenters

Christina Delimitrou & Christos Kozyrakis
Electrical Engineering Department, Stanford University
{cdel, kozyraki}@Stanford.edu
Introduction

• Increase amount of computing in the cloud systems
  • It’s a cost benefit for both end user and operator of DC

• Scheduling the new coming application to the available servers is done by the cloud system’s operator and it must be:
  • Fast execution
  • High resource efficiency
  • Enabling better scaling at low cost

• Ignoring heterogeneity can cause
  • Poor efficiency
Introduction (cont.)

- Interference
  - It happens because of co-scheduling multiple workloads in a single server in order to increase utilization and achieve better cost efficiency

- To achieve better scalability
  - Co-locate the application so that the number of servers can host larger number of applications

- The paragon has been evaluated on different workload scenarios
Data centers issues

• The problems in the large data centers:
  1. Interference between correlated workloads
  2. Which application will assigned to which hardware platform

• The existing solution
  • Solved the pervious problems
  • Drawback:
    • Can’t be applied online
    • Doesn’t scale few beyond application
    • Depend on prior analysis to get knowledge about the applications.
The goal of this paper is to implement an online and scalable scheduler which is heterogeneity and interference aware to eliminate the problems related to the heterogeneous datacenters.

It will focus on solving:

- Hardware platform heterogeneity
- Workload interference
Analytical Method

• The key is to classify incoming applications in accurate and fast way

• Some information is needed:
  1. How fast the application will run on each server configuration available
  2. How much interference it will cause itself and how much it will get from other workloads on various shared resources.

• Benefits:
  1. Strong analytical guarantees information quality used in scheduling
  2. Computationally efficient, scale well with large number of application and SCs
Analytical Method

• Analytical methods
  • will not use any prior knowledge about incoming application.

• Collaborative filtering technique
  • Use singular value to preform it and identify similarities between new and previously scheduled workloads (similarities between application preferences)
Collaborative filtering background

- It is frequently used in recommendation systems
- Netflix challenge
  - Provide valid movie recommendation for Netflix users given the rating they have provided for various other movies.
- The analytical methods used are
  1. Singular Value Decomposition (SVD)
  2. PQ-reconstruction (PQ)
Sparse Matrix

- Analytical Method’s input
- Netflix Recommender method

<table>
<thead>
<tr>
<th></th>
<th>Movie 1</th>
<th>Movie 2</th>
<th>Movie 3</th>
<th>Movie 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>User 1</td>
<td>7</td>
<td>8</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>User 2</td>
<td>8</td>
<td>?</td>
<td>?</td>
<td>10</td>
</tr>
<tr>
<td>User 3</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>User 4</td>
<td>7</td>
<td>9</td>
<td>8</td>
<td>7</td>
</tr>
</tbody>
</table>
Why Collaborative filtering?

- It will merge new application profile with a large amount of data related to the application that has been already scheduled.
- Identify similarities between new and known applications
- It can determine how will an application can run in different hardware platforms available.
- The result will be a high efficiency and accurately classification within a minute of its arrival, and efficiently scheduled the incoming workload on large-scale cluster.
Classification Heterogeneity

- Using collaborative filtering
- Rating represent normalized application performance on each SC
- Identifies similarities between new and existing application

<table>
<thead>
<tr>
<th></th>
<th>SC 1</th>
<th>SC 2</th>
<th>SC 3</th>
<th>SC 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application X</td>
<td>7</td>
<td>8</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>Application Y</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>Application Z</td>
<td>7</td>
<td>9</td>
<td>8</td>
<td>7</td>
</tr>
<tr>
<td>Application N</td>
<td>8</td>
<td>?</td>
<td>?</td>
<td>10</td>
</tr>
</tbody>
</table>
Classification for Interference

- The interference will be detected due to shared resource contention and assign a score to the sensitivity of application to a type of interference.

- This will be achieved by using microbenchmarks which each will stress a specific resource with tunable intensity.

- Identify SoI by identifying ten co-scheduled applications that contend on large number of shared resources (consider them as SoI); then design tunable micro-benchmark for each one.

- Collaborative filtering for interference:
  - The same as the one for heterogeneity
  - Applications as rows, SoI as columns, and the element of the matrices are the sensitivity score of an application to the corresponding benchmark.
Classification for Interference

- Validation

<table>
<thead>
<tr>
<th>Metric</th>
<th>Percentage (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average sensitivity error across all SoIs</td>
<td>5.3%</td>
</tr>
<tr>
<td>Average error for sensitivities &lt; 30%</td>
<td>7.1%</td>
</tr>
<tr>
<td>Average error for sensitivities &lt; 60%</td>
<td>5.6%</td>
</tr>
<tr>
<td>Average error for sensitivities &gt; 60%</td>
<td>3.4%</td>
</tr>
<tr>
<td>Apps with &lt; 5% error</td>
<td>ST: 65%  MT: 58%</td>
</tr>
<tr>
<td>Apps with &lt; 10% error</td>
<td>ST: 81%  MT: 63%</td>
</tr>
<tr>
<td>Apps with &lt; 20% error</td>
<td>ST: 90%  MT: 89%</td>
</tr>
<tr>
<td>SoI with highest error</td>
<td></td>
</tr>
<tr>
<td>for ST: L1 i-cache</td>
<td>15.8%</td>
</tr>
<tr>
<td>for MT: LLC capacity</td>
<td>7.8%</td>
</tr>
<tr>
<td>Frequency L1 i-cache used as offline SoI</td>
<td>14.6%</td>
</tr>
<tr>
<td>Frequency LLC cap used as offline SoI</td>
<td>11.5%</td>
</tr>
<tr>
<td>SoI with lowest error</td>
<td></td>
</tr>
<tr>
<td>for ST: network bandwidth</td>
<td>1.8%</td>
</tr>
<tr>
<td>for MT: storage bandwidth</td>
<td>0.9%</td>
</tr>
</tbody>
</table>

Table 2: Validation metrics for interference classification.
Classification Efficiency

- Paragon Classification shows that it will decrease the interference and increase the server utilization.

- The result is fast and high accurate classification for the incoming applications with respect to heterogeneity and interference, and incoming workload can be scheduled efficiently on a large scale cluster.
Figure 2: The components of Paragon and the state maintained by each component. Overall, the state requirements are marginal and scale linearly or logarithmically with the number of applications (N), servers (M) and configurations.
Greedy Server Selection

- The selection will be as the following
  1. Identify servers that doesn’t violate QoS
  2. Select the best SC between them

- The greedy scheduler proves it’s ability to decrease the interference and increase server utilization

- If no candidate is found backtracking may extended to more levels and in the worse case may extend all the way to the first SoI.
Statistical Framework for Server Selection

- It’s based on sampling
- It’s more efficient than greedy for large scale servers 10–100k
- In the greedy, there will be overhead in examining the server state.
- Instead of examining the whole server state, only a small number of servers will be sampled.
- Hash functions will be used to show randomness in server selection
- Candidate were ranked by colocation quality
  - A metric that will define how a given server is suitable with the new workload.
Discussion

- **Workload phases**
  1. Huge number of workloads have to be not scheduled to the same server because of the interference information will be inaccurate for this workload
  2. Find a mechanism for migration

- **Suboptimal scheduling due to**
  1. Greedy selection algorithm
  2. Pathological behavior in application arrival patterns

- **Latency-critical applications and workload dependencies**
  1. The latency critical applications and dependencies between application component is not considered in the paragon
Methodology

• **Server Systems**
  - The paragon has been evaluated on small local cluster and 3 cloud computing services

• **Schedulers**
  - The paragon compared with LL, NH and NI schedulers

• **Workloads**
  - Different workloads were used such as ST, MT, MP and I\O

• **Workload scenarios**
  - The applications above used to create multiple workload scenario.
  - The experiment done for small and large scale were three workloads were examined.
Evaluation

1. Comparison of Schedulers: Small scale
   • QoS Guarantees
   • Scheduling decision quality
   • Resource allocation
   • Server utilization
   • Scheduling overhead

2. Comparison of Schedulers: Large scale
   • Decision quality
   • Resource allocation
   • Windows Azure & Google Compute Engine
Comparison of Schedulers: Small scale

Figure 6: Performance comparison between the four schedulers for three workload scenarios: low, oversubscribed and workloads with phases (Fig. 6(a, b, c)) and performance over time for the scenario where workloads experience phases (Fig. 6d).
Comparison of Schedulers: Large scale

Figure 9: Performance comparison between the four schedulers, for three workload scenarios on 1,000 EC2 servers.
Related work

• Datacenter scheduling
• VM management
• Resource management and rightsizing
• Scheduling for heterogeneous multi core chips
Conclusion

• DC scheduler is a heterogeneity and interference aware
• It’s derived from analytical methods (Collaborative filtering).
• Classification depend on the information from previously scheduled workloads
• Classification result is used by the greedy scheduler to assign the workload to the server that will enhance application performance and decrease resource usage
• The paragon has been evaluated in both small and large scale systems
• Paragon preserves QoS guarantees and improve server utilization which will be a benefit for both end-user and DC operator
• Future work : considering how to couple Paragon with the VM management and rightsizing systems for large scale data center
Discussion

• Can you consider this paragon as optimal solution?
  • If no, why?

• Is the classification methodology used is the optimal?
  • If no, why?

• Is there is a better scheduling technique to improve the utilization and the performance?