Dynamic Analysis of Application Delivery Network for Leveraging Software Defined Infrastructures

Lav Gupta, Raj Jain
Washington Univ. in St Louis
{lavgupta, jain}@wustl.edu

Mohammed Samaka
Qatar University
samaka.m@qu.edu.qa

2nd IEEE International workshop on Software Defined Systems
Tempe, AZ, 9th March, 2015

These slides and a video recording of this presentation are at:
http://www.cse.wustl.edu/~jain/talks/profile.htm

Washington University in St. Louis
http://www.cse.wustl.edu/~jain/talks/profile.htm
©2015 Raj Jain
Overview

1. OpenADN Architecture
2. Need For Profiling OpenADN
3. Profiling Led Optimization of Multi-Cloud Platforms
4. OpenADN Profiling
Multi-Cloud Application Delivery

New Business Opportunities: Datacenters on Towers, Internet of Things

Users

Network

Distributed Clouds

Cloud/Enterprise Datacenters
Services in a Cloud of Clouds

OpenADN Platform

- OpenADN Resource Driver (OpenStack)
- OpenADN Resource Driver (OpenDayLight)
- OpenADN Resource Driver (EC2)

Northbound Interface

- Application Service Providers
- Application Architects
- Application Deployment Administrators

Southbound Interface

- Resource Providers

Cloud Services

- Virtual Hosts
- Virtual Storage
- Virtual Network

Application Deployment Administrators

- ISP Network
- Network POP Micro-Dataplanes

OpenADN Application Service (AAS) abstraction

OpenADN Application Workflow (AAW) abstraction

OpenADN Application Cloud (AAC) abstraction

OpenADN Application Service Providers

Virtual WAN Services

OpenStack

OpenDayLight

EC2

Washington University in St. Louis

http://www.cec.wustl.edu/~jain/talks/profile.htm

©2015 Raj Jain
Key aspects of OpenADN Architecture

- Hybrid control – global and local controllers
- Centralized management, distributed data plane
- Application and Network layer services
Need for profiling OpenADN

- OpenADN and other similar platforms tend to be massively distributed and complex software.

- Use of multithreading for concurrent execution makes code difficult to optimize. Simple code reading fails to find potential bottlenecks.

- Profiling isolates hotspots, that consume disproportionate resources, and aids optimization.
Profiling Led Optimization

a) Non-Profiling Based Optimization

- Source
- Optimization based on intuition, guesswork, reading of code
- Sub-optimal code

b) Profiling Based Optimization

- Source
- Event timing, code instrumentation
- Optimization
- Optimal Code
- Original code
- Profiling
- Optimization
- Resulting code
Profiling Techniques

- **Profiling**: Analyzing program behavior and gathering data to analyze performance of a platform
- **Static**: Analysis of code by reading or model checking
- **Dynamic**:
  - Deterministic: Instrumented code
  - Statistical: Sampling process states
Static Profiling

- **Manual**: Passive checking of control flows and execution states
  - Manual: Dry run/Reading → Analysis → Results

- **Automatic**: Make model to exhaustively explore inter-leavings
  - Automatic: Source Code → Scanning/conversion → Intermediate Representation → Analysis → Results

  - Computationally expensive for numerous inter-leavings
  - Does not give program behavior under execution
  - Only gives relative execution times

References:
- Washington University in St. Louis
- [http://www.cse.wustl.edu/~jain/talks/profile.htm](http://www.cse.wustl.edu/~jain/talks/profile.htm)
Dynamic Profiling

- The system generates information about its execution parameters while it executes. Primarily two ways to do it.
  - **Statistical**: sampling of process states – relative
  - **Deterministic**: *absolute and precise* measure of events function calls or more fine-grained flow transitions.

Deterministic profiling has been carried out for OpenADN
Virtual Environment for OpenADN Profiling

- Virtual clouds created:
  - One global controller
  - Two local controllers
  - One name server
  - Hosts – 7 per datacenter
  - Client host – 10000 users

- Levels of profiling

Diagram:
- GC
- Nameserver
- LC
- Hosts
- Client host
Platform-Level Profiling

- Assessment of user CPU time, System CPU time and real time

<table>
<thead>
<tr>
<th>Runs</th>
<th>I</th>
<th>II</th>
<th>III</th>
<th>IV</th>
<th>V</th>
<th>Averages</th>
<th>% Run time</th>
</tr>
</thead>
<tbody>
<tr>
<td>User Space</td>
<td>0.53</td>
<td>0.55</td>
<td>0.62</td>
<td>0.6</td>
<td>0.61</td>
<td>0.58</td>
<td>1.65</td>
</tr>
<tr>
<td>System Calls</td>
<td>0.76</td>
<td>0.75</td>
<td>0.65</td>
<td>0.67</td>
<td>0.68</td>
<td>0.7</td>
<td>1.99</td>
</tr>
<tr>
<td>User+System</td>
<td>1.29</td>
<td>1.3</td>
<td>1.27</td>
<td>1.27</td>
<td>1.29</td>
<td>1.28</td>
<td>3.64</td>
</tr>
<tr>
<td>Run time</td>
<td>35.82</td>
<td>35.6</td>
<td>34.65</td>
<td>34.8</td>
<td>35.06</td>
<td>35.19</td>
<td></td>
</tr>
</tbody>
</table>

Comparison of run time and user/kernel time

Time in seconds

Washington University in St. Louis  
http://www.cse.wustl.edu/~jain/talks/profile.htm  
©2015 Raj Jain
Platform-Level Profiling

Advantages:
- Provides CPU time spent in user and system software.
- It gives total run time which tells us how effectively platform is using computing resources.
- It indicates the possibility of higher load on the CPU because of potentially wasteful activities.

Shortcoming:
- If run time is much higher than the total time for user and system calls, it does not tell what is taking this time.
Function Level Profiling

Gives cumulative execution time of various user, system & library functions (including sub-function calls)

Extract of a function level profile run
Analysis of Function Level Profile

Advantages:

- The function level profiling gives cumulative times in the functions of OpenADN modules like GC, LC, hosts etc.
- It confirmed that certain functions like polling (MQ library function) and Python sleep functions take unduly long part of the run time.

Shortcomings:

- Does not tell which modules to look into to locate problems?
- Does not give fine-grain profiling down to the statements level. So statements causing problems cannot be pin-pointed.
Statement-Level Profiling

- It introduces special code or hooks (e.g. in Python) to record the execution time for each statement.

<table>
<thead>
<tr>
<th>Line #</th>
<th>Hits</th>
<th>Time</th>
<th>Per Hit</th>
<th>%Time</th>
<th>Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>273</td>
<td>1</td>
<td>24689</td>
<td>24689.0</td>
<td>0.0</td>
<td>simNetwork.start_client_host()</td>
</tr>
<tr>
<td>274</td>
<td>1</td>
<td>1499</td>
<td>1499.0</td>
<td>0.0</td>
<td>print(&quot;---------\n&quot;)</td>
</tr>
<tr>
<td>275</td>
<td>1</td>
<td>1301</td>
<td>1301.0</td>
<td>0.0</td>
<td>print(&quot;checkpoint 5...after client host&quot;)</td>
</tr>
<tr>
<td>276</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>#start the monitoring</td>
</tr>
<tr>
<td>277</td>
<td>1</td>
<td>17</td>
<td>17.0</td>
<td>0.0</td>
<td>endTime = time() + _runTime</td>
</tr>
<tr>
<td>278</td>
<td>102526</td>
<td>481979</td>
<td>4.7</td>
<td>0.2</td>
<td>while time()&lt; endTime:</td>
</tr>
<tr>
<td>279</td>
<td>102526</td>
<td>208269149</td>
<td>2031.4</td>
<td>87.2</td>
<td>readable = poller.poll(1)</td>
</tr>
<tr>
<td>280</td>
<td>102810</td>
<td>495496</td>
<td>4.8</td>
<td>0.2</td>
<td>for fd, _mask in readable:</td>
</tr>
<tr>
<td>281</td>
<td>285</td>
<td>1032</td>
<td>3.6</td>
<td>0.0</td>
<td>node = Node.outToNode[ fd ]</td>
</tr>
<tr>
<td>282</td>
<td>285</td>
<td>11422</td>
<td>40.1</td>
<td>0.0</td>
<td>outString = node.monitor().strip()</td>
</tr>
</tbody>
</table>

Time in seconds

A sample clip of Statement-level profiling
Statement-Level Profiling (Cont)

- Shows that sleep and polling functions dominate execution times.

- Some modules like Global Controller and Name server are inactive most of the time (so they can share resources!)

- Checking ports for inter-process messages (polling) takes up 87.2% of the entire simulation time

- Optimization of the code may lead to reduced virtual resource demand and operational expense.
Observations

Top-Level Profiling

A large component of non-user, non-kernel time that could be explained by I/O waits. Some part of this time could be spent unproductively using up resources and contributing to energy consumption.

Function level Profiling

Functions that have potential hot spots.

Statement-Level Profiling

On complete platform allows interplay of threads and reveals the parts of the functions that could be helped with optimization efforts.
Summary

1. OpenADN is a platform for managing and controlling resources across multiple clouds.

2. Profiling is useful for optimization as follows:
   a. Critically examine the time spent in I/O waits and take remedial measures wherever possible
   b. Examine the use of sleep statements and fine-tune their durations
   c. Examine the use of heartbeat and ways to make it efficient
   d. Optimize the time taken to dynamically create and destroy virtual resources
Conclusions
Summary

1. OpenADN is a platform for managing and controlling resources across multiple clouds.

2. Multi-cloud management systems need to have their performance optimized

3. Hotspots lead to increased resource requirement and higher operational expenses

4. Increasingly fine grained profiling of platform behavior provides useful data for optimization.