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

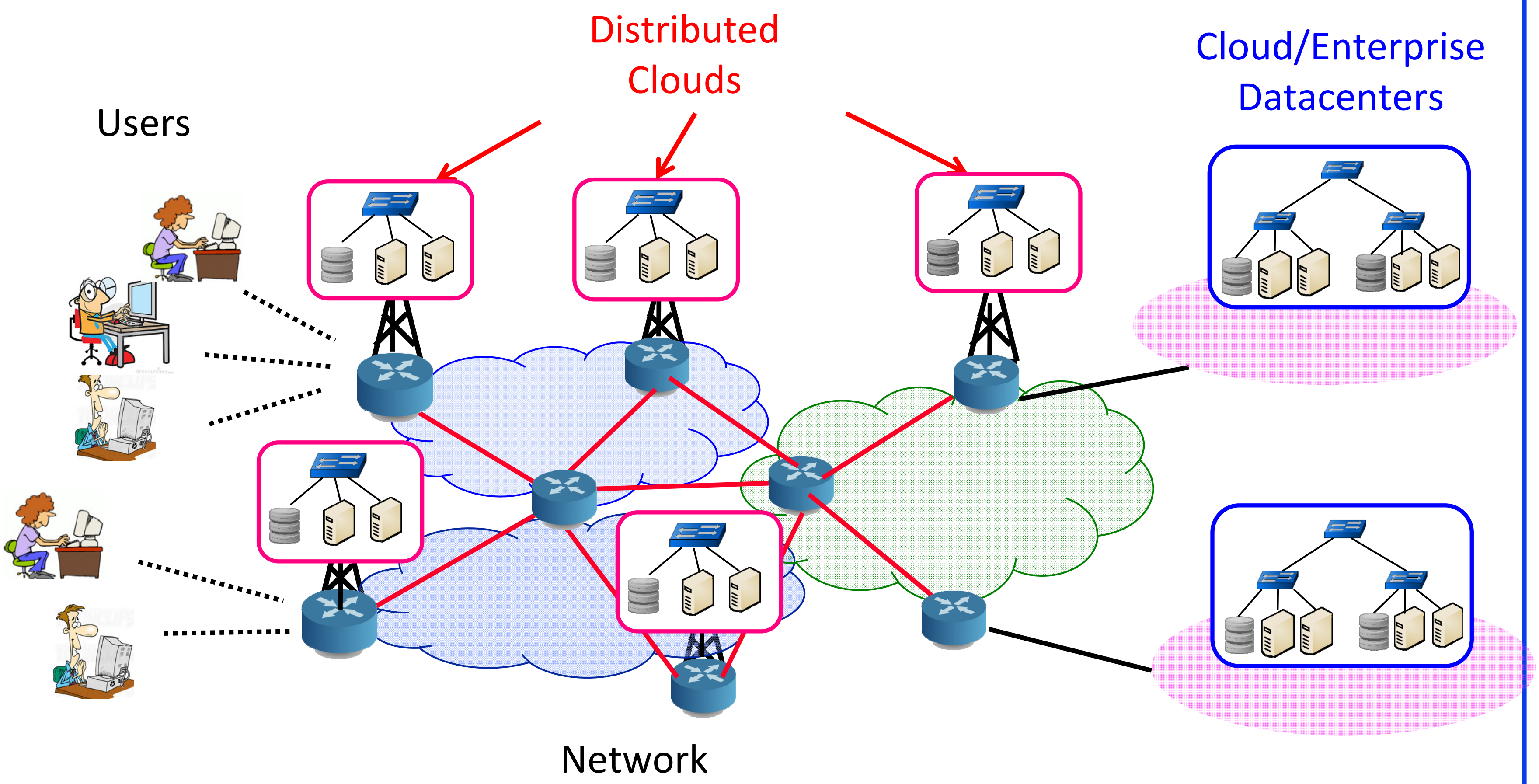
These slides and a video recording of this presentation are at:
http://www.cse.wustl.edu/~jain/talks/profile.htm

©2015 Raj Jain

Overview

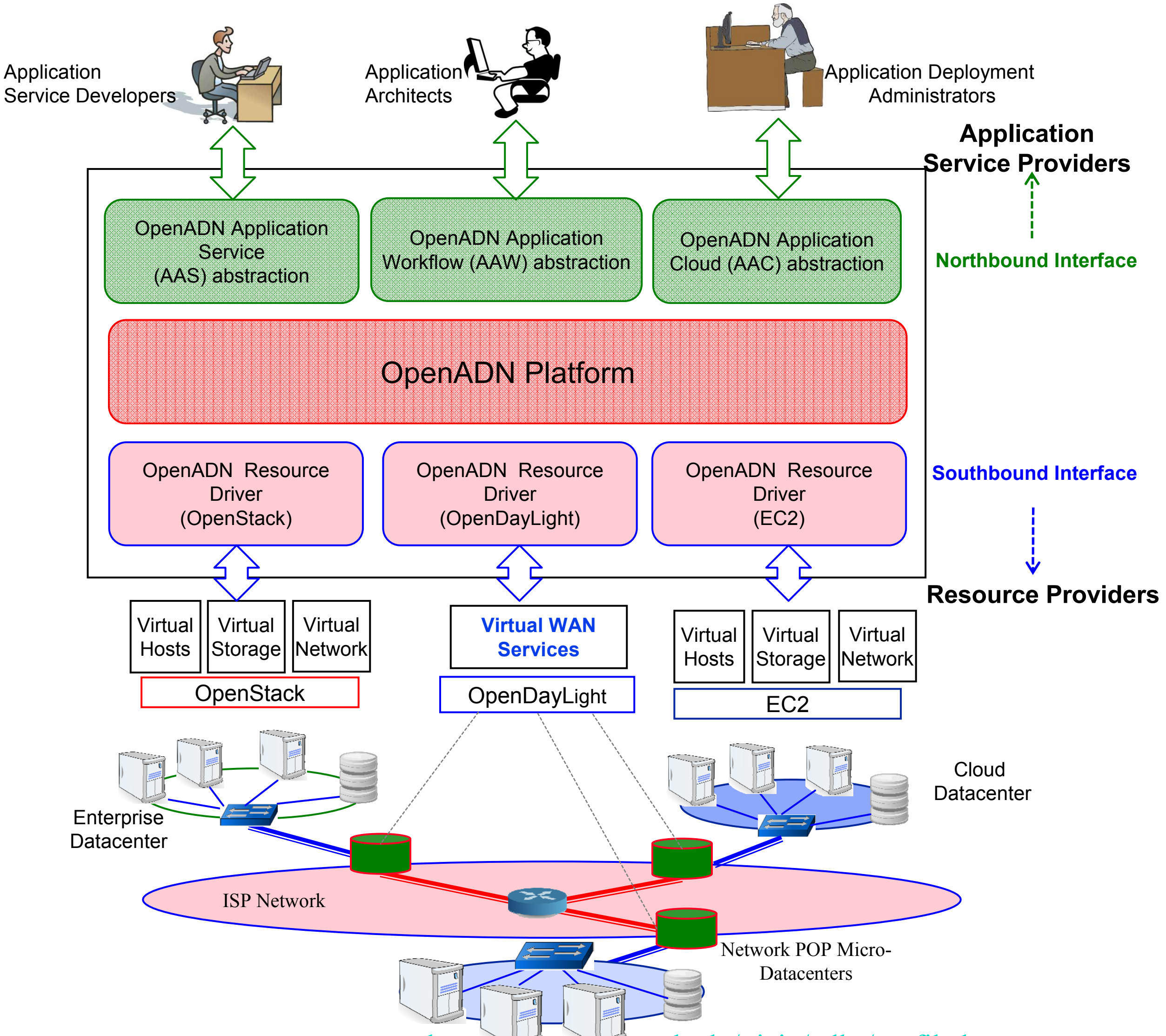
1. OpenADN Architecture
2. Need For Profiling OpenADN
3. Profiling Led Optimization of Multi-Cloud Platforms
4. OpenADN Profiling

Multi-Cloud Application Delivery



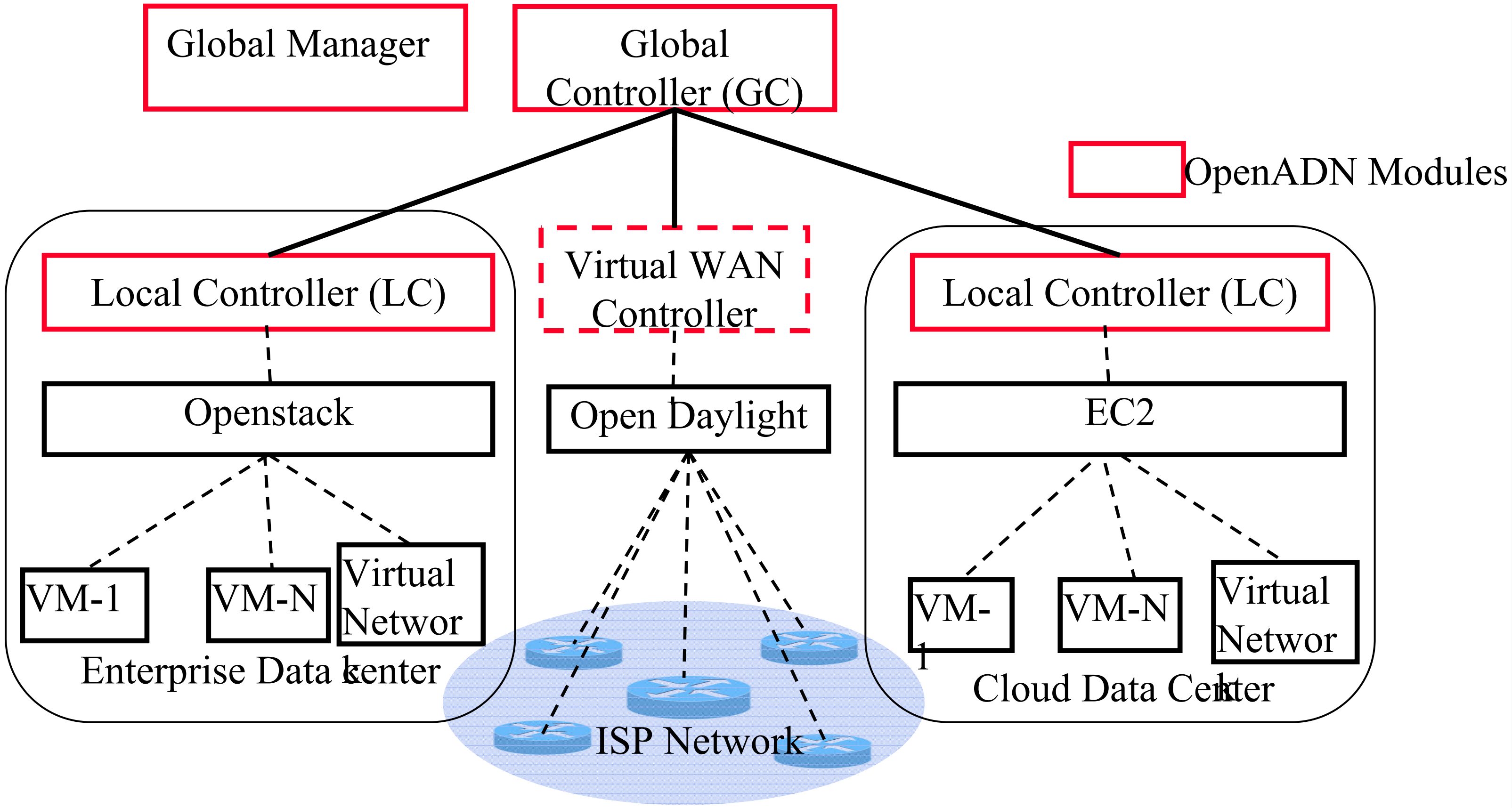
New Business Opportunities: Datacenters on Towers, Internet of Things

Services in a Cloud of Clouds


Application Service Developers
Application Architects
Application Deployment Administrators
Application Service Providers
Northbound Interface
OpenADN Application Service (AAS) abstraction
OpenADN Application Workflow (AAW) abstraction
OpenADN Application Cloud (AAC) abstraction
OpenADN Platform
OpenADN Resource Driver (OpenStack)
OpenADN Resource Driver (OpenDayLight)
OpenADN Resource Driver (EC2)
Southbound Interface
Resource Providers
Virtual Hosts
Virtual Storage
Virtual Network
OpenStack
Virtual WAN Services
OpenDayLight
Virtual Hosts
Virtual Storage
Virtual Network
EC2
Enterprise Datacenter
Cloud Datacenter
ISP Network
Network POP Micro-Datacenters

©2015 Raj Jain

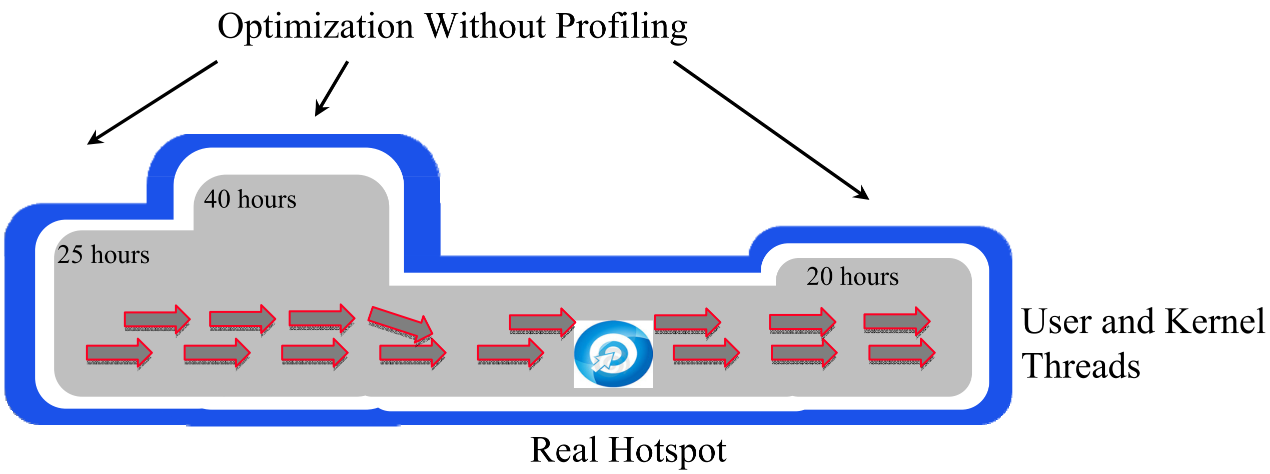
Key aspects of OpenADN Architecture



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

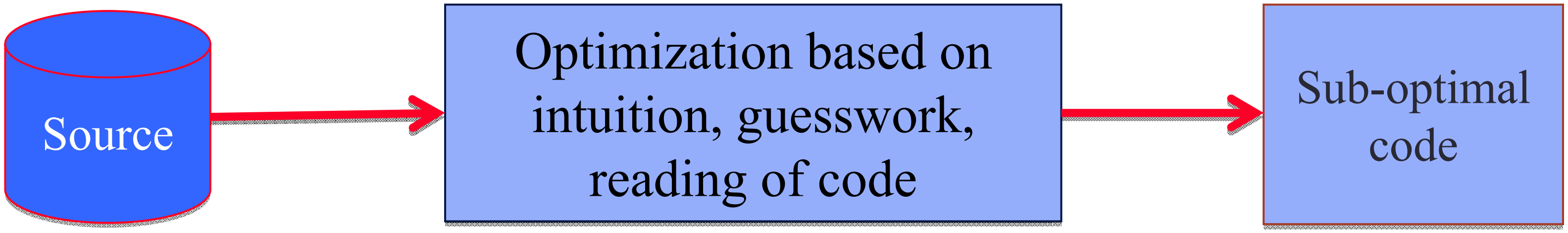
Need for profiling OpenADN

- 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.

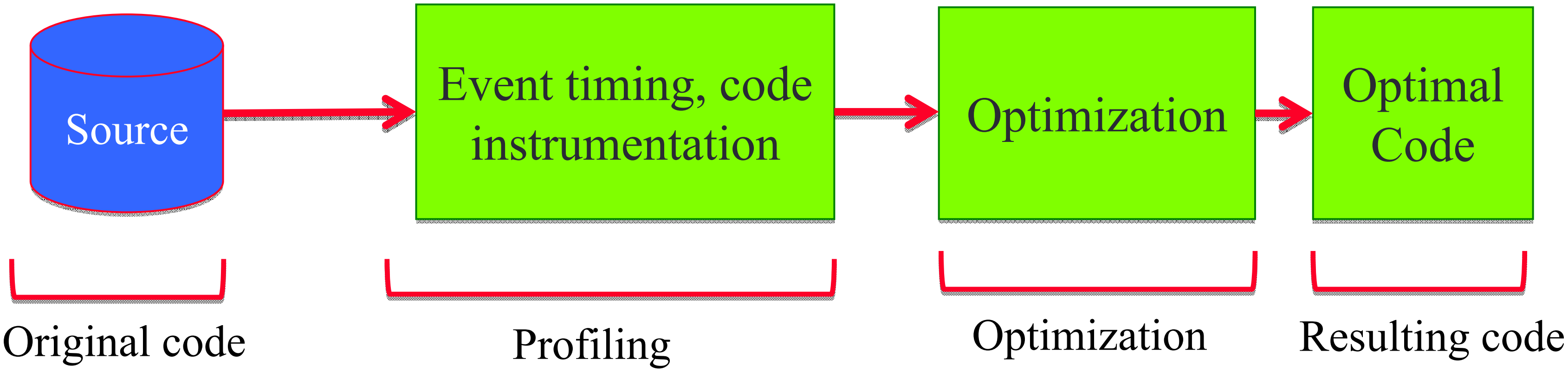


- Profiling isolates hotspots, that consume disproportionate resources, and aids optimization

Profiling Led Optimization



a) Non-Profiling Based Optimization



b) Profiling Based Optimization

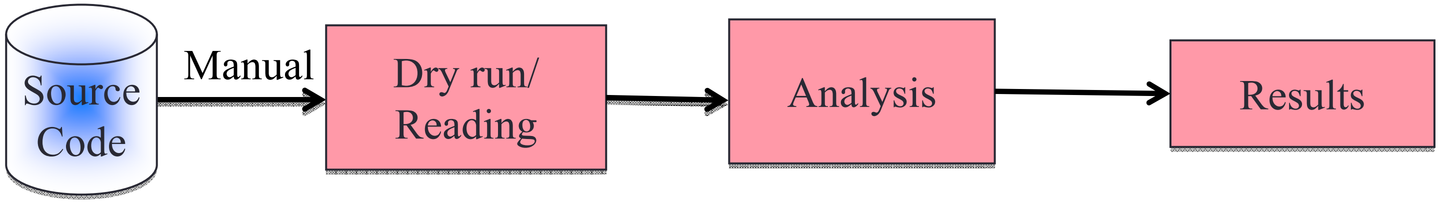
©2015 Raj Jain

Profiling Techniques

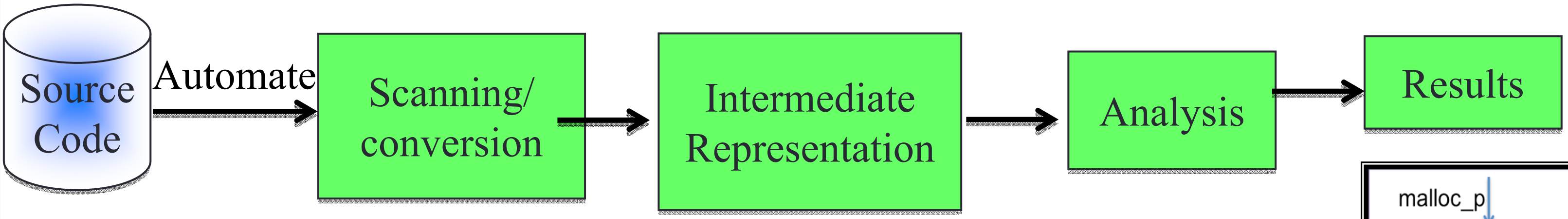
- **Profiling**: Analyzing program behavior and gathering data to analyze performance of a platform
- **Static**: Analysis of code by reading or model checking
- **Dynamic**:
  - Deterministic: Instrumented code
  - Statistical: Sampling process states

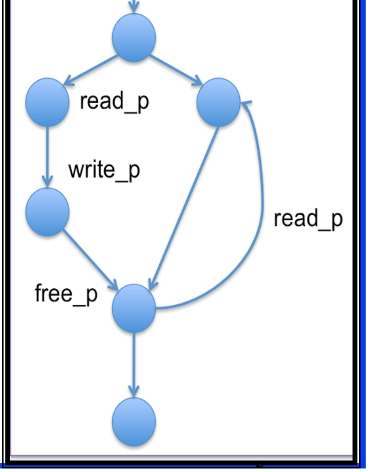
Static Profiling

- **Manual**: Passive checking of control flows and execution states



- **Automatic**: Make model to exhaustively explore inter-leavings

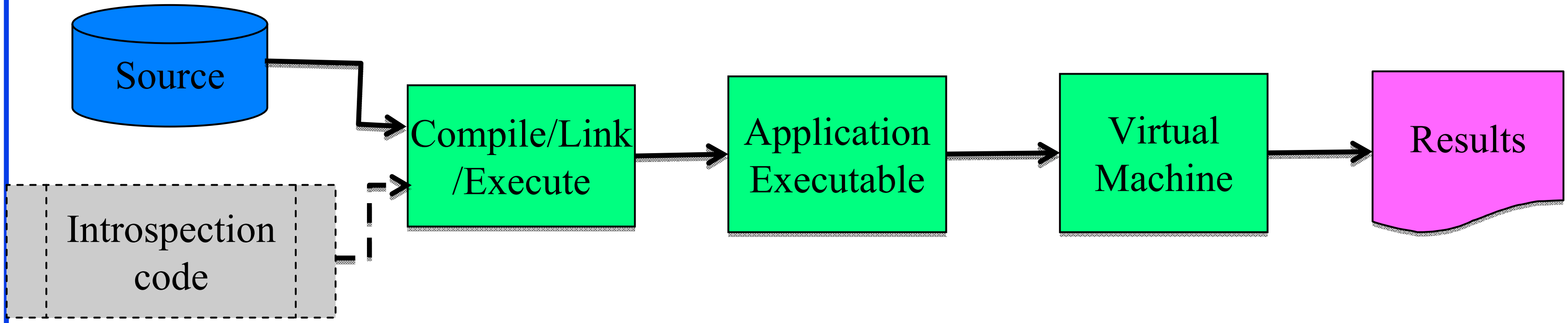




- Computationally expensive for numerous inter-leavings
- Does not give program behavior under execution
- Only gives relative execution times

Dynamic Profiling

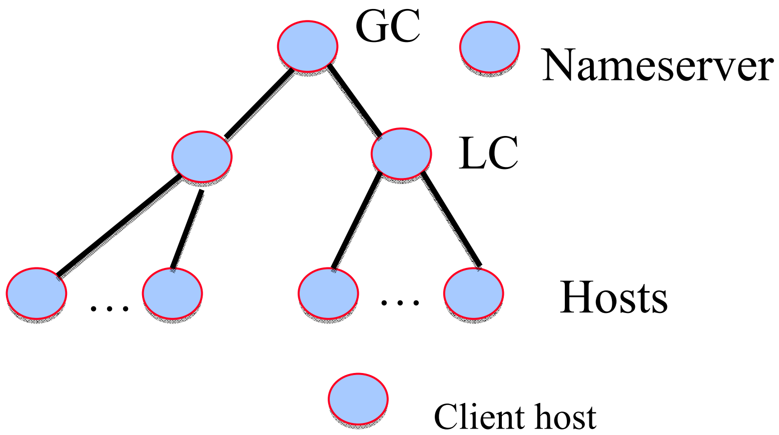
- The system generates information about its execution parameters while it executes. Primarily two ways to do it.
- **Statistical**: sampling of process states – relative
- **Deterministic**: *absolute and precise* measure of events function calls or more fine-grained flow transitions.



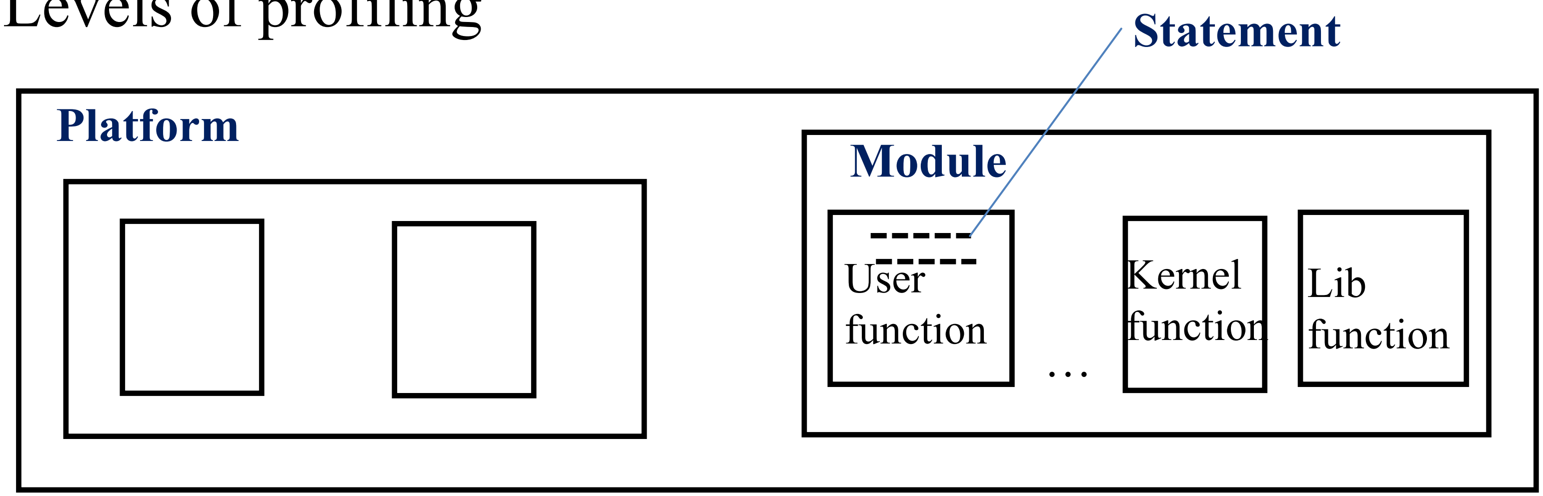
- Deterministic profiling has been carried out for OpenADN

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

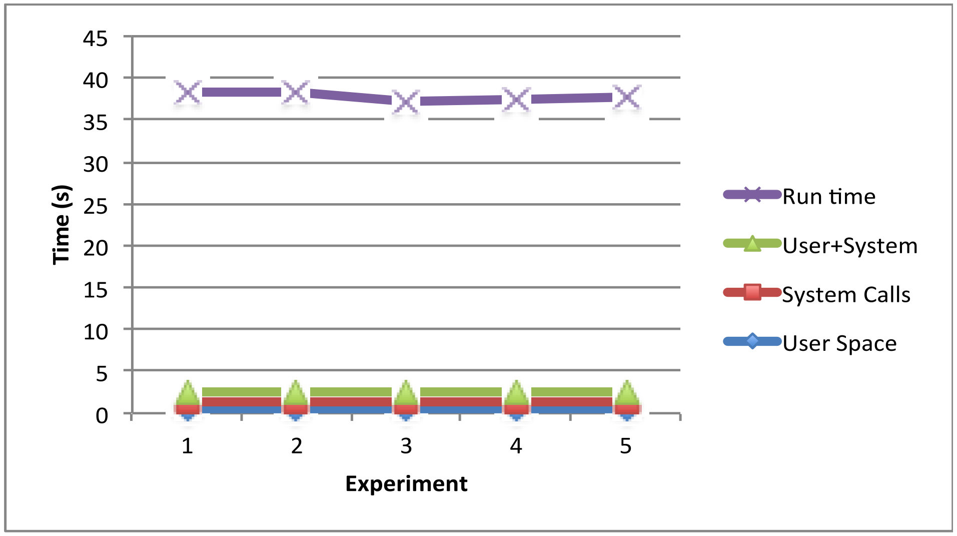


Platform-Level Profiling

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

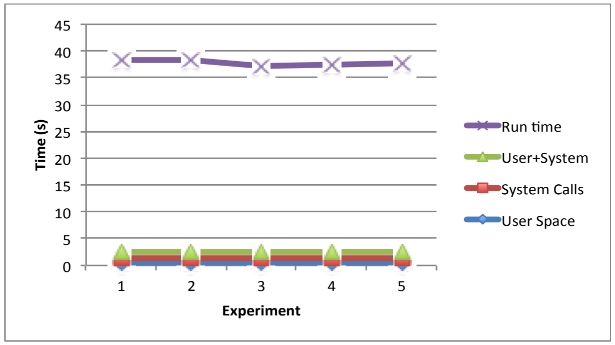
Runs	I	II	III	IV	V	Averages	% Run time
User Space	0.53	0.55	0.62	0.6	0.61	0.58	1.65
System Calls	0.76	0.75	0.65	0.67	0.68	0.7	1.99
User+System	1.29	1.3	1.27	1.27	1.29	1.28	3.64
Run time	35.82	35.6	34.65	34.8	35.06	35.19	

Time in seconds



Comparison of run time and user/kernel time

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.

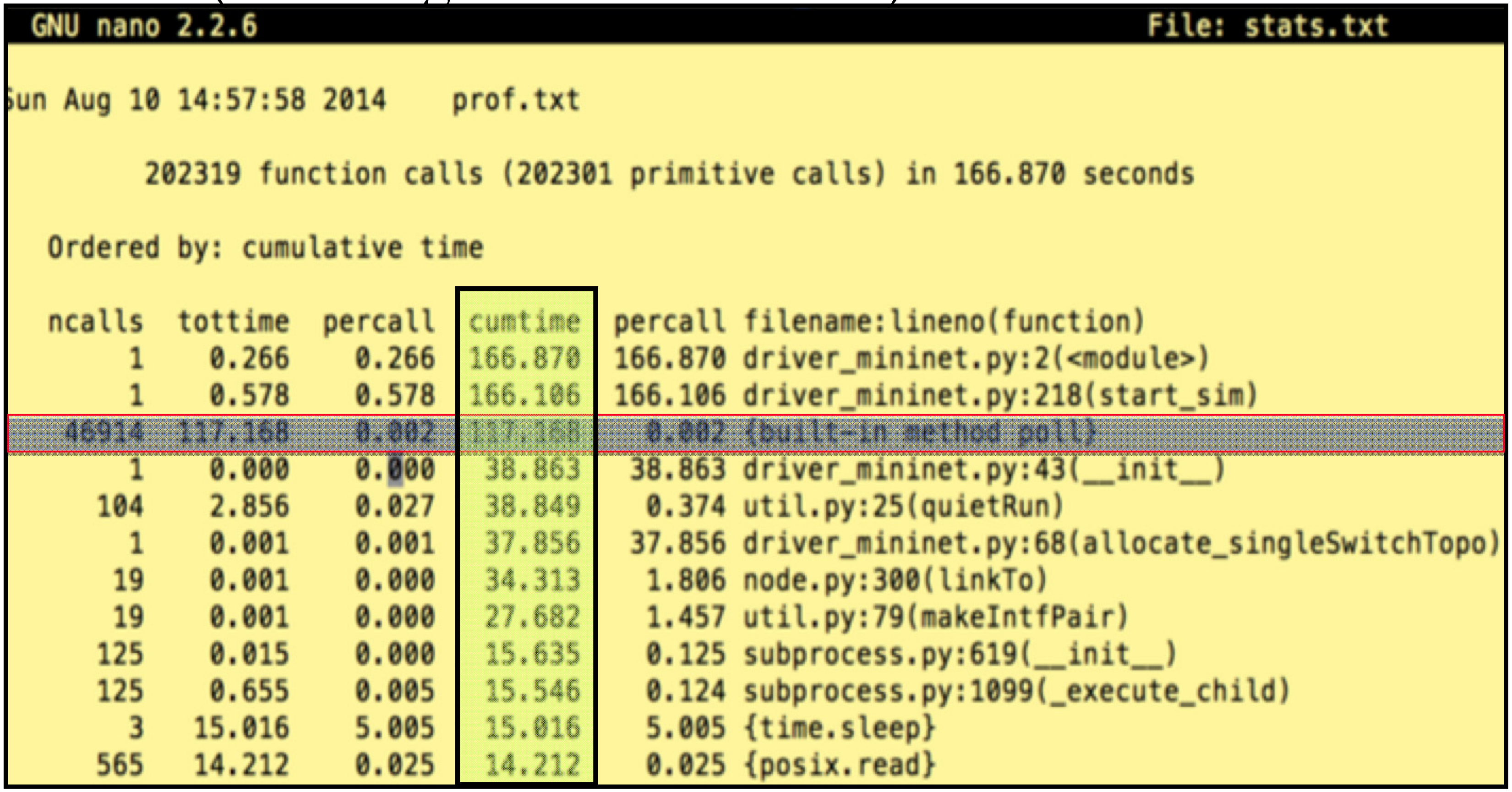
Shortcoming:

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

©2015 Raj Jain

Function Level Profiling

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



```
  GNU nano 2.2.6                                                 File: stats.txt

Sun Aug 10 14:57:58 2014    prof.txt

         202319 function calls (202301 primitive calls) in 166.870 seconds

   Ordered by: cumulative time

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.266    0.266  166.870  166.870 driver_mininet.py:2(<module>)
        1    0.578    0.578  166.106  166.106 driver_mininet.py:218(start_sim)
    46914  117.168    0.002  117.168    0.002 {built-in method poll}
        1    0.000    0.000   38.863   38.863 driver_mininet.py:43(__init__)
      104    2.856    0.027   38.849    0.374 util.py:25(quietRun)
        1    0.001    0.001   37.856   37.856 driver_mininet.py:68(allocate_singleSwitchTopo)
       19    0.001    0.000   34.313    1.806 node.py:300(linkTo)
       19    0.001    0.000   27.682    1.457 util.py:79(makeIntfPair)
      125    0.015    0.000   15.635    0.125 subprocess.py:619(__init__)
      125    0.655    0.005   15.546    0.124 subprocess.py:1099(_execute_child)
        3   15.016    5.005   15.016    5.005 {time.sleep}
      565   14.212    0.025   14.212    0.025 {posix.read}
```

Extract of a function level profile run

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

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(ϕMQ library function) and Python sleep functions take unduly long part of the run time.

Shortcomings:

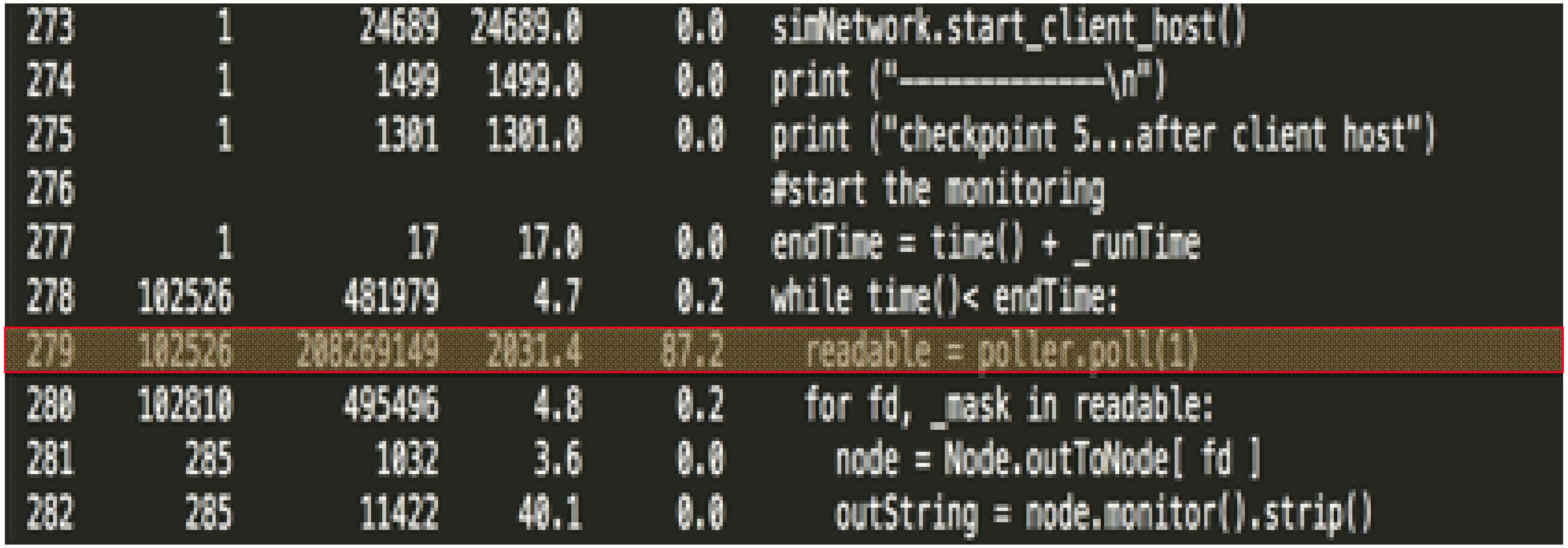
- ❑Does not tell which modules to look into to locate problems?
- ❑Does not give fine-grain profiling down to the statements level. So statements causing problems cannot be pin-pointed.

©2015 Raj Jain

Statement-Level Profiling

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

Line # Hits Time Per Hit %Time Statement



```
273        1     24689  24689.0     0.0  simNetwork.start_client_host()
274        1      1499   1499.0     0.0  print ("------------\n")
275        1      1301   1301.0     0.0  print ("checkpoint 5...after client host")
276                                      #start the monitoring
277        1        17     17.0     0.0  endTime = time() + _runTime
278   102526    481979      4.7     0.2  while time()< endTime:
279   102526 208269149   2031.4    87.2     readable = poller.poll(1)
280   102810    495496      4.8     0.2     for fd, _mask in readable:
281      285      1032      3.6     0.0       node = Node.outToNode[ fd ]
282      285     11422     40.1     0.0       outString = node.monitor().strip()
```

Time in seconds

A sample clip of Statement-level profiling

©2015 Raj Jain

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

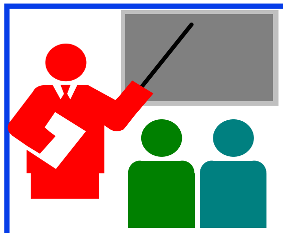
A large component of non-user, non-kernel time that could be explained by I/O waits. Some part of this time could be spent unproductively using up resources and contributing to energy consumption.

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.

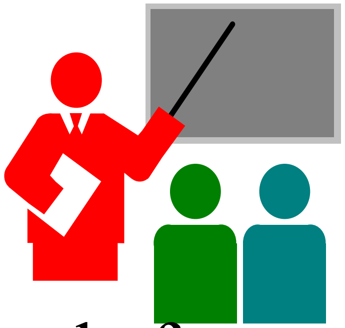


Summary

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.