Cloud Monitoring Framework

Hitesh Khandelwal, Ramana Rao Kompella, Rama Ramasubramanian
Email: {hitesh, kompella}@cs.purdue.edu, rama@microsoft.com

December 8, 2010

1 Introduction

With the increasing popularity of public cloud computing platforms like Amazon EC2 [1], Microsoft Azure [2], etc, more and more applications are now being deployed to leverage the scalability and free manageability provided by them. This flexibility comes at the expense of giving up control over the network. Cloud providers do not expose the network topology to cloud users. Applications deployed on clouds are agnostic to the internal topology and assumes all network links to be homogeneous.

Cloud owners provide few monitoring tools(like Amazon CloudWatch[1]) to provide customers with visibility into resource utilization and operational performance from end-host perspective. Metrics exposed such as CPU utilization, disk reads and writes, and network traffic are measured from end-host's usage pattern. What cloud users really want is measurements from network's perspective. Metrics such as latency, available bandwidth of a network-link would be more informative and beneficial. Unfortunately, because of security reasons cloud owners do not provide any information about the location of a VM instance or network topology, and hence currently it is infeasible to know how a network-link is performing.

We took an intermediate approach by measuring performance of end-to-end path between two VM instances, rather than each link between them. In this work, we envision to build a monitoring framework which continuously provides measurement information to the cloud user. To build such a framework, we have combined various measurement tools and proposed few techniques to adopt them in high speed cloud environment. We can easily modify most of the popular applications that run in cloud environment to take advantage of such a framework. Here we describe some of the scenarios which can leverage additional network information:

- Load balancer in *3-Tiered web architecture* can use the latency information provided by the framework to distribute the queries across different links connected to application and database servers. Using uncongested links will result into smaller end-to-end latency and better end-user experience.
- Master in *Map-reduce*[2] can use available bandwidth information provided by the framework to appropriately choose mapper and corresponding reducer. Using mapper and reducer connected by high bandwidth path will result into better throughput for the system.
- Using traffic and latency measurements between each pair of VM, we can single out poorly performing VMs. Continuously releasing such bad VMs and demanding new VMs from the cloud provider, will result into an optimal set of VMs, with good network connectivity between them. After many number of iterations we may be able to find VMs co-located in the same rack.

Though building such a framework is part of a bigger vision, in this work, we will try to equip ourselves with appropriate tools, and presents a preliminary experimental results of performance of Amazon EC2 platform. We conducted measurements on Amazon EC2, using iperf for estimating available bandwidth and Ping for estimating round trip time. Section 2 discusses requirements of an appropriate framework and measurement tools in high speed data center network. In section 3 we evaluates performance of EC2 using 19 small instances. Finally we conclude in section 4.

[1]http://aws.amazon.com/ec2/
[2]http://www.microsoft.com/windowsazure/

2 Framework design and measurement tools

We designed a framework that performs end-to-end measurements at VM instances. Different applications are sensitive to different performance metrics. A 3-tier web application have numerous small flows with strict latency requirements, while batch processing systems like map-reduce have large and fat flows that demands for high throughput. Commonly measured metrics are latency and throughput, although both availability and reliability metrics have started to gain popularity. In this work we will focus on bandwidth(related to achievable TCP throughput) and latency(related to Round trip time) of path between VM instances. Through an API, cloud applications will have access to this information, and as discussed earlier they can exploit it to improve their performance. Before we divulge into details about measurement of available bandwidth and latency, we list some of the requirements for our framework:

- *Lightweight:* It should be lightweight and non-intrusive to the flows of normal applications. Also a measurement flow should not interfere with other measurements, this may result into wrong readings.
- *Scalable:* It should be scalable to thousands of VM instances, without using high CPU on each instance.
- *Up-to-date:* It should continuously monitor the network and report up-to-date performance metrics to the cloud user. Measurements should performed frequently without being intrusive to the normal traffic.
- *Accurate:* It should be accurate with appropriate metric resolution.

We started building the framework as a standalone daemon, which provides applications with measurement results. To reduce interference with normal application traffic, we can piggyback measurement packets on application data. Ideally we would like to export measurement API, which applications like web-servers and map-reduce, can extend to piggyback the measurement packets. To solve the problem of interference between measurement flows, we will use a centralized scheduler. For scalability purposes, it is also possible to use a distributed consensus algorithm at each participating VM. In the subsequent sections we will discuss about tools and techniques for bandwidth and latency measurement.

2.1 Bandwidth measurement

Estimating available bandwidth along a network path has received considerable attention in recent years and multiple tool have been proposed [9] [7] [8] [5] [6] [4] [3]. Informally an end-to-end available bandwidth of a path is defined as the capacity of the link with minimum unused capacity(tight link).

Spruce and Pathload represents two different class of techniques employed by all the measurement tools. Spruce estimates bandwidth by sending packet pairs spaced back to back according to the capacity of the tight link. Amount of packet dispersion at receiver end determines the amount of cross-traffic in the tight link. Pathload estimates bandwidth by creating short-lived congestion in the tight link. It detects the congestion by finding trends in one way probe packet delays.

All the methods of measurement assume simplified homogeneous network conditions. Some of the other assumptions are:

1. FIFO queuing at routers, with no reordering of packets.
2. Cross-traffic is fluid, with infinitely small packet size.
3. Cross-traffic rate is stable during measurement period.
4. They ignore the network compression that may happen just after congestion is cleared.
5. Assuming uniform per packet processing and timing. Network adapter interrupt coalesce breaks this assumption.

Moreover, all the tools were designed for low speed Internet links and are unsuitable for high speed data center network. In new environment various assumptions become invalid and tools are stretched to their limits. [7] conducted an experimental study on high speed Gigabit network and benchmarked majority of the popular tools. They found that Pathload and Pathchirp are the most accurate tools under such conditions. They also discussed about other challenges in high speed networks, like the issue of time precision, interrupt coalesce etc. Here we list some of the other limitations of Spruce like tools, and why we chose Pathload for our measurement framework.

- It needs supplementary tool for absolute capacity estimation, making it useless for DCs where we dont have any information about network topology.
- It assumes the same link to be both narrow and tight along the path.
- It requires system clock sync across all the machines.
- UNIX timestamp resolution is not sensitive enough for high speed paths.

2.2 Latency measurement

Ping is the most popular tool available on all the OSs to measure round trip time of a path. Ping uses ICMP ECHO messages to get an estimate of the round trip time(approximately $2\times$ latency). Measuring latency at milli-seconds level, in data centers is not much challenging. Ping can very well serve our purpose of measuring RTT.

3 Evaluation

3.1 Setup

We performed all the measurements on Amazon Elastic Compute Cloud(EC2). In all our experiments we used *small instances* with Ubuntu 10.04 server version running on them. We also used a single *micro instance* for control and management purposes. As mentioned in section 2.1 measurements in our framework are centrally coordinated by a scheduler which defines the order of measurements. To emulate the centralized scheduler we used a predefined serialized *schedule* file at each VM instance. This schedule identifies a timestamp, along with the nodes that should communicate to produce a single reading of the appropriate metric. Serial schedule ensures that no two measurement flows are interfering with each other.

Similar to most of the other cloud providers, Amazon EC2 also do not provide any information about the location of the VM instances or network topology connecting them. Each instance have a private IP to communicate with other VMs, over internal Data-center network.

3.2 Pathload accuracy

We used Pathload(with bw_resolution = 25Mbps) to measure available bandwidth. We also ran iperf in conjunction with Pathload, for each path, to get achievable TCP throughput. Achievable TCP throughput is always less than available bandwidth, because of slow start and congestion control mechanisms in TCP. Iperf is a popular tool for end-to-end performance measurement, and its results are widely accepted by research community. To quantify the accuracy of Pathload in high speed data center network, we ran experiments on 5 small instances. There are 20 links(read paths) between 5 instances, and we generated a serial schedule for running Pathload and iperf alternatively on each such link. We started Pathload sender and iperf server on each of the 5 VMs. Framework daemon reads the schedule file and appropriately starts Pathload receiver or iperf client, to get a reading for the link.

It takes 15 seconds for Pathload before its algorithm converges or terminates due to an error. Just after taking reading for Pathload we ran iperf for 12 secs with default client buffer of 16K and default server buffer of 85.3K. We repeated the experiment 5 times to avoid any transitory losses or hypervisor scheduling issues. Each round lasts for approximately 540 secs(20*(15+12)), and after each round we output two 5×5 matrices corresponding to pathload's available bandwidth and iperf's achievable TCP throughput estimations each.

–	(599.20,631.58)	(1500.00,1714.29)	(0.00,1500.00)	(596.40,666.67)
(0.00, Error)	–	(750.00,750.00)	(600.00,631.58)	(631.58,666.67)
(750.00,800.00)	(516.17,534.55)	–	(571.43,530.08)	(666.67,705.88)
(496.00,513.04)	(600.00,631.58)	(375.00,600.00)	–	(631.58,705.88)
(600.00,800.00)	(750.00,672.73)	(428.57,705.88)	(666.67,705.88)	–

(a) Pathload available bandwidth matrix

–	(494)	(443)	(469)	(551)
(584)	–	(333)	(469)	(596)
(722)	(686)	–	(507)	(685)
(521)	(637)	(275)	–	(636)
(550)	(616)	(335)	(465)	–

(b) Iperf available bandwidth matrix

Figure 1: Available bandwidth estimation

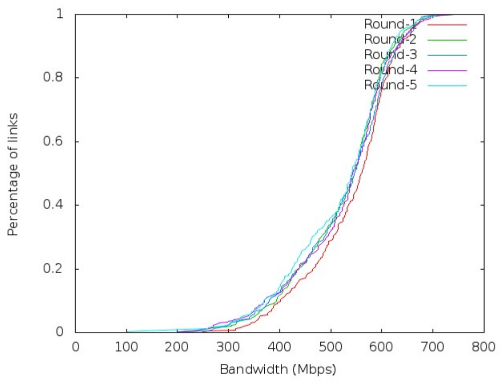
Pathload analysis:
Figure 1 shows available bandwidth(in Mbps) matrix for Pathload and iperf, for round 5 only. We chose round 5 because Pathload results are most correct in this round.

- Experiment terminates prematurely and code is buggy(eg. element (2,1) of the matrix). The problem is even more prevalent in other rounds.
- Pathload returns arbitrary bandwidth estimations for some of the links(eg. elements (1,3) (1,4)).
- It takes a long time before its algorithm converges. In our setup Pathload took 15 secs on average, while iperf can reach steady state in 2-3 seconds only. Long convergence time demotivates the use of Pathload to measure link bandwidth in serial fashion.
- Most of the readings are unstable with large variation in each link's bandwidth estimation across 5 rounds. While iperf have a stable reading across all the rounds, for each link.
- Resolution is too large and insensitive to small changes in the bandwidth. In high speed data center network, it takes approx. 12 usecs to transmit a packet of size 1500 bytes at the rate of 1000 Mbps. Next feasible transmission rate is 923 Mbps corresponding to 13 usecs. Because of UNIX timestamp granularity of 1 usecs it is infeasible to achieve resolution of less than 100 Mbps. Pathload was designed to work on low speed Internet links with bandwidth 100 times smaller than those in data center network.
- Contrary to the findings of previous work [?], we found that iperf's estimation are lower than Pathload bandwidth estimations.

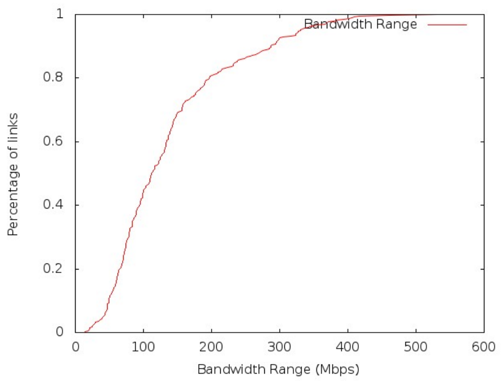
We conclude that Pathload is un-suitable for bandwidth estimation in data center network.

3.3 Available bandwidth

Based on our experience with Pathload on 5 instances in section 3.2, we decided to use only iperf for rest of the measurements. Although iperf eats a lot of bandwidth for measuring throughput, we sacrifice on efficiency for accuracy of results. For rest of the section, we will interchangeably use the term achievable throughput with available bandwidth. Next we conducted measurements on 19 instances with 342 links between them. As before, we generated a serial schedule for starting iperf clients. We ran iperf for 6 seconds to get a reading for a link. Each round lasts for approximately 30 minutes.



(a) CDF of link bandwidth estimation for all the rounds

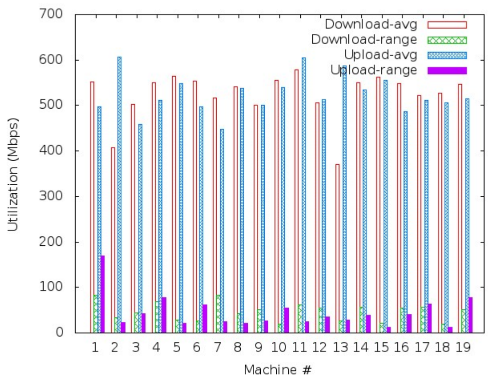


(b) CDF of link bandwidth variation across all the rounds

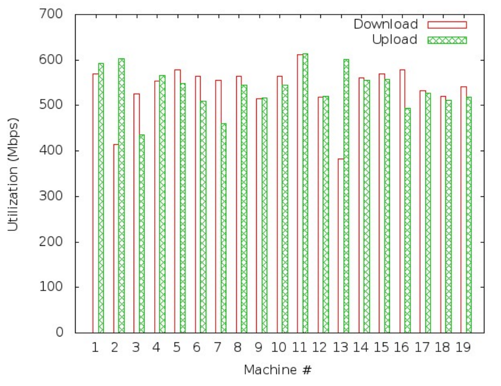
Figure 2: Iperf Bandwidth estimation

Figure 2(a) plots CDF of achievable TCP throughput estimation for each round. For each round we use corresponding throughput matrix, having 342 estimated values. We can observe that only less than 10% links have available bandwidth less than 400 Mbps, hence Amazon EC2 platform is optimally utilized with plenty of bandwidth available to each user. Also there is no long lasting congestion in links.

We define *range* as the difference between maximum and minimum value of the metric measured across all the rounds. Figure 2(b) shows CDF of link's available bandwidth range across 5 rounds. We can conclude, that for more than 80% of the links, available bandwidth is consistent across the time. Only around 20% links have variation of more than 200 Mbps.



(a) Average and variation in download/upload bandwidth across all the rounds



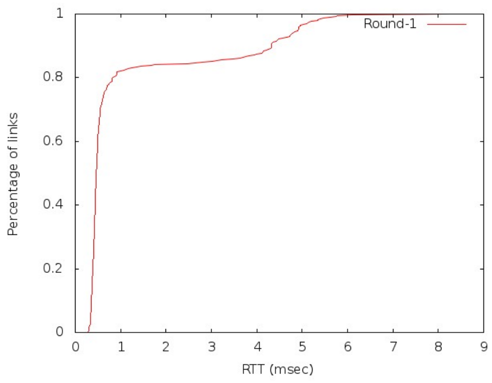
(b) Average download/upload bandwidth for round-1

Figure 3: VM instance download and upload bandwidth

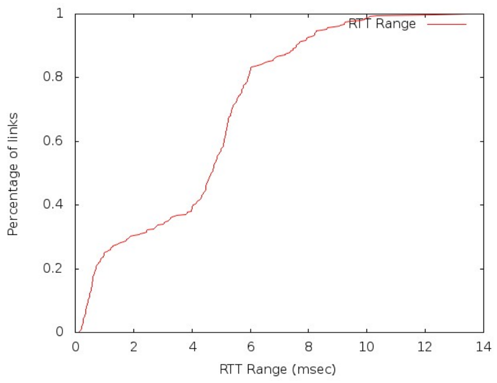
Figure 3(a) shows average download/upload available bandwidth and its range(max-min), for each instance, across all rounds. Consistent with observation in 2(a) almost all the instances have average download/upload bandwidth more than 400 Mbps. There are some instances that we can single out for poor performance, based on range in bandwidth(like 1, 4, 7, 19..). Figure 3(b) shows average download/upload available bandwidth for each instance, for round 1 only. Graph is similar to above fig 3(a).

3.4 Latency

We used ping to measure RTT between any two VMs.One important difference between latency and bandwidth measurement techniques, is that, due to small number and size of Ping packets they do not interfere with each other and normal application traffic. Hence, we generated a parallel schedule than a serialized one. We conducted parallel measurements on 19 instances and one control instance. We used 6 ICMP ECHO messages in each Ping measurement with an interval of 1 seconds between them. There are 30 rounds of experiments and each round lasts for 6 seconds.



(a) CDF of link RTT for rounds 1



(b) CDF of link RTT variation across all the rounds

Figure 4: Ping RTT estimation

Figure 4(a) plots a CDF of RTT for all the 342 links, for round 1 only. From our logs we found that median RTT is 0.3 msecs and RTT for any link not facing congestion is less than 1 msec. Figure shows that more than 20% of the links are facing congestion(RTT $>$ 1 msec).

In figure 4(b) we show a CDF of link's RTT range across 30 rounds. We can conclude that more than 80% links are facing congestion at some point. Reading the logs we found that absolute RTT values goes as high as 14 msec. From figure 2(a) we concluded that there is no long lasting congestion in Amazon EC2, but there is some transitory congestion which lasts only for few seconds. This transitory congestion will affect the performance of applications having large number of small flows, by forcing them to abort.

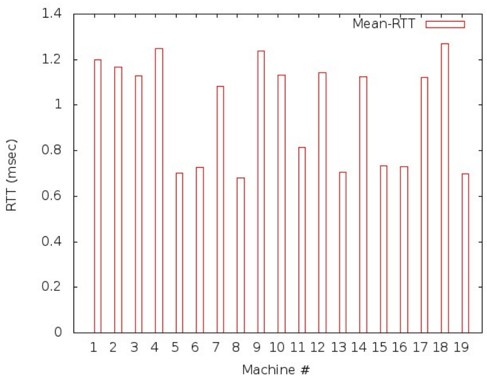


Figure 5: Average RTT across all the rounds

Figure 5 shows average RTT of a link, for each instance, across all rounds. It is evident that congestion is not specific to any particular instance, and all the instances are observing congestion at some point in time. This shows that most of the traffic in EC2 is short lived and bursty in nature.

4 Conclusion

In this paper, we first enumerated all the requirements for a better cloud monitoring framework. Currently, we focused on popular metrics like available bandwidth and round trip time between each pair of VM instances. Towards our goal of building such a framework, we combined various bandwidth and latency estimation tools. Through experiments on 19 small instances on Amazon EC2, we found that most of the bandwidth estimation tools proposed in past are inaccurate in high speed data center network. Also Amazon EC2 platform is optimally utilized with ample bandwidth available to all the tenants.

Though there is a lot of temporal congestion in EC2, but very few long lasting congested links exist in the network. A smart Web load balancer which not only takes computation resources but network resources also into consideration will improve the end-user performance. For batch processing systems like map-reduce, if network communication forms a substantial part of the total time, it makes sense to place mapper and reduce along a well connected path. We envision that in future, public cloud providers will also expose some of their networking details to cloud users. Our proposed framework could be complemented by that information to improve performance of applications in public clouds.

References

[1] Amazon cloudwatch. `http://aws.amazon.com/cloudwatch/`.

[2] Dean, J., and Ghemawat, S. Mapreduce: simplified data processing on large clusters. *Commun. ACM 51* (January 2008), 107–113.

[3] Hu, N., Member, S., Steenkiste, P., and Member, S. Evaluation and characterization of available bandwidth probing techniques. *IEEE Journal on Selected Areas in Communications 21* (2003), 879–894.

[4] Jain, M., and Dovrolis, C. End-to-end available bandwidth: measurement methodology, dynamics, and relation with tcp throughput. *IEEE/ACM Trans. Netw. 11* (August 2003), 537–549.

[5] Kapoor, R., Chen, L.-J., Lao, L., Gerla, M., and Sanadidi, M. Y. Capprobe: a simple and accurate capacity estimation technique. In *Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications* (New York, NY, USA, 2004), SIGCOMM '04, ACM, pp. 67–78.

[6] Ribeiro, V. J., Riedi, R. H., Baraniuk, R. G., Navratil, J., and Cottrell, L. pathchirp: Efficient available bandwidth estimation for network paths. In *In Passive and Active Measurement Workshop* (2003).

[7] Shriram, A., Murray, M., Hyun, Y., Brownlee, N., Broido, A., Fomenkov, M., and Claffy, K. C. Comparison of public end-to-end bandwidth estimation tools on high-speed links. In *PAM* (2005), pp. 306–320.

[8] Sommers, J., Barford, P., and Willinger, W. A proposed framework for calibration of available bandwidth estimation tools. *Computers and Communications, IEEE Symposium on 0* (2006), 709–718.

[9] Strauss, J., Katabi, D., and Kaashoek, F. A measurement study of available bandwidth estimation tools. In *Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement* (New York, NY, USA, 2003), IMC '03, ACM, pp. 39–44.