FERMILAB-CONF-07-470-CD

End-to-End Network/Application Performance Troubleshooting Methodology

Wenji Wu, Andrey Bobyshev, Mark Bowden, Matt Crawford, Phil Demar, Vyto Grigaliunas, Maxim Grigoriev, Don Petravick

Fermilab, P.O. 500, Batavia, IL, 60510

wenji@fnal.gov

Abstract. The computing models for HEP experiments are globally distributed and grid-based. Obstacles to good network performance arise from many causes and can be a major impediment to the success of the computing models for HEP experiments. Factors that affect overall network/application performance exist on the hosts themselves (application software, operating system, hardware), in the local area networks that support the end systems, and within the wide area networks. Since the computer and network systems are globally distributed, it can be very difficult to locate and identify the factors that are hurting application performance. In this paper, we present an end-to-end network/application performance troubleshooting methodology developed and in use at Fermilab. The core of our approach is to narrow down the problem scope with a divide and conquer strategy. The overall complex problem is split into two distinct sub-problems: host diagnosis and tuning, and network path analysis. After satisfactorily evaluating, and if necessary resolving, each sub-problem, we conduct end-to-end performance analysis and diagnosis. The paper will discuss tools we use as part of the methodology. The long term objective of the effort is to enable site administrators and end users to conduct much of the troubleshooting themselves, before (or instead of) calling upon network and operating system "wizards," who are always in short supply.

1. Introduction

High-energy physics (HEP) is a very data-intensive science. The computing models for HEP experiments are globally distributed and grid-based, with analysis most effectively carried out in widely dispersed computing facilities. Site and long-haul networks serving large HEP institutions are provisioned for, and carry, large scientific traffic loads. But with the upcoming Large Hadron Collider (LHC) experiments at CERN each expecting several petabytes of raw data annually, and sites like Fermilab already moving multiple petabytes monthly, the need for efficient exploitation of available network resources by commodity computers and operating systems has never been greater.

As LHC experiments commission their data handling and computing infrastructures and the Tevatron experiments at Fermilab do more analysis "on the grid," they sometimes encounter network throughputs that disappoint their expectations. Obstacles to good network performance arise from many causes and can be a major impediment to the success of the grid-based computing models for HEP experiments. Factors that affect overall network/application performance exist on the host themselves (application software, operating system, hardware), in the local area networks that support

the end systems, and within the wide area networks. Since the computer and network systems are globally distributed, it can be very difficult to locate and identify the factors that are hurting application performance. Fermilab's Computing Division has an ongoing program of boosting data throughput across the wide area, and has developed a methodology for systematic identification and resolution of obstacles to high performance, whether they are in the host, the site network, or in the wide area. The core of our approach is to narrow down the problem scope with a divide and conquer strategy. The overall complex problem is split into two distinct sub-problems: host diagnosis and tuning, and network path analysis. After satisfactorily evaluating, and if necessary resolving, each sub-problem, we conduct end-to-end performance analysis and diagnosis. The methodology proposed in the paper can also be applied to other grid-based large-scale scientific collaborations.

The remainder of the paper is organized as follows: Section 2 gives the network performance analysis methodology. In Section 3, we give a case study to show how to apply the network performance analysis methodology. And finally in Section 4, we conclude the paper.

2. Network Performance Analysis Methodology

The factors that affect the overall network/application performance could exist in hosts themselves (application software, operating system, and hardware), in the local area networks that support the hosts, or within wide area networks. Since the hosts and network devices are globally distributed, usually across organizational and management boundaries, it can be very difficult to locate and identify the exact factors hurting distributed application performance. Fermilab has been developing a network/application performance troubleshooting methodology in order to analyze these types of performance problems in a structured manner. The core of our approach is to narrow down the problem space with a divide and conquer strategy. An end-to-end performance problem is viewed as potentially three distinct sub-problems:

- Application-related problems
- Host diagnosis and tuning
- Network path analysis

Application-specific problems are beyond the scope of any standardized problem analysis. Instead, the methodology first seeks to analyze and appropriately tune the hosts. Next, the network path analysis is conducted, with remediation of detected problems where feasible. If host and network path analysis do not uncover significant problems or concerns, packet trace analysis of individual data flows will be done to compare expected versus actual packet trace patterns. An absence of problem indications or other anomalies in all of the above areas strongly points to application-related issues as being the source of the lower-than-expected performance.

2.1. Performance Analysis Network Architecture

When conducting performance analysis, we need to collect host information and measure network path characteristics. We deploy diagnostic tools to facilitate the analysis. Our performance analysis network architecture is as shown in Figure 1. The network architecture consists of:

- Host diagnostic server. Currently, we use the Network Diagnostic Tool (NDT) [1] to identify performance and configuration problems in hosts. NDT can collect the various TCP network parameters on the hosts and identify their configuration problems. Since NDT is deployed in our own local area networks, it also helps to identify local network infrastructure problems such as faulty Ethernet connections, malfunctioning NICs, and Ethernet duplex mismatch.
- Network path diagnostic server. We use OWAMP (One-Way Active Measurement Protocol) [2] to collect and diagnose one-way network path statistics, rather than round trip metrics. Since OWAMP applications consist of owampd daemon and owping client, the network path diagnosis

server means the owampd daemon here. The owping client needs to run on the other end to measure the network path statistics. To diagnose the network path characteristics, other tools like traceroute, pathneck [3], and iperf [4] could be deployed. Traceroute obtains the sequence of routers along the path. Pathneck detects the path's bandwidth bottleneck location. Iperf is a tool to measure maximum TCP throughput. Iperf reports bandwidth, delay jitter, datagram loss. Finally, other network monitoring services like MRTG [5], Nagios [6], PerfSONAR [7], and MonALISA [8], if available, could also be referenced to diagnose the network path characteristics.

- Packet trace diagnostic server. If host and network path analysis do not uncover significant problems or concerns, packet trace analysis of individual data flows will be done to compare expected versus actual packet trace patterns. Any performance bottleneck will manifest itself in the recorded packet traces. The packet trace diagnosis server is connected to the border router, which can mirror any other interface in the border router. As such, it has the capability to record any traffic flow passing in or out of the Fermilab site network. In the server, tcpdump [9] and tcptrace [10] are employed to record and analyze the traffic flows. Furthermore, xplot [10] is used to examine the recorded traces visually.

These servers are logical entities, and physically could be deployed on the same node. Since the end-to-end performance analysis requires cooperation from both sides, we encourage our data movement partner sites to deploy similar tools to facilitate the performance analysis. We also use other tools like vmstat, ps, ifconfig, ethtool, and netstat, but some of them are OS-dependent we do not discuss them all here.

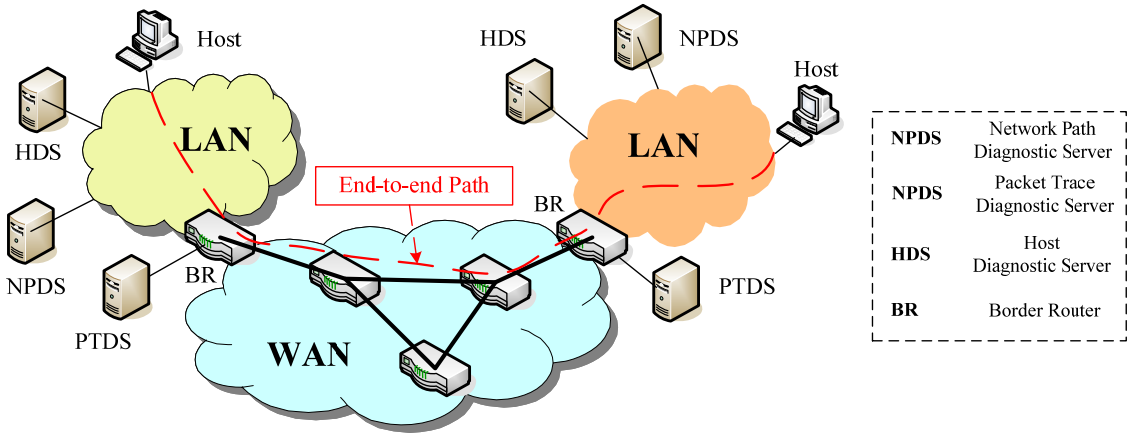

Figure 1 Performance Analysis Network Architecture

2.2. Performance Analysis

In general, five steps are used to conduct the network/application performance analysis:

Step 1: Definition of the problem space.

End user collects and provides the basic information on the scope and characteristics of the problem, as well as specific information on the system & application involved.

Step 2: Collection of host information & network path characteristics.

The host information sought is shown in Table 1. The information could be obtained and collected by examining the hosts, observing with a monitoring system like Nagios, or diagnosing with NDT.

The end-to-end path characteristics we are interested in are: packet drop rate & loss pattern (loss distance, loss period); packet delay & delay variation (min, max, average, percentiles); and packet reordering. OWAMP can provide detailed path characteristics on both directions. We are also interested in the sequence of routers along the path, and the possible bottleneck location. That information could be obtained by traceroute and pathneck respectively.

Hardware	**CPU**	CPU speed; CPU model;
	Memory	Memory size;
	Bus, Disk	Maximum bus bandwidth; Maximum disk I/O bandwidth;
	NIC	Maximum bandwidth; Interrupt coalescing supported? TCP offloading supported? Jumbo frame supported?
Software	**Operating System**	Operating system type, versions; 32bit/64 bit? For Linux, identify the kernel version;
	System Loads	Network applications running context; Maximum system background loads;
	Network Applications	Network application traffic generation pattern; Storage system involved?
	TCP Parameters	Send/Receive buffer size; Timestamp option enabled? Window scaling option enabled? Window scaling parameter; TCP reordering threshold; Congestion control algorithm; Total TCP memory size; Maximum Segment Size; SACK enabled? D-SACK enabled? ECN enabled?
	NIC Driver Parameter	Device driver send/recv queue size; TCP offloading enabled? Interrupt coalescing enabled? Jumbo Frames enabled?

Table 1 Network End Systems Information

Step 3: Host system diagnosis

When diagnosing the hosts, we first need to make sure: (1) the system performance is not limited by hardware such as CPU, disk, memory, NIC, and bus; (2) the network applications have enough CPU share. An overloaded system may not give enough CPU time to network applications, and can drop packets inside the kernel itself.

We then diagnose the host with NDT. NDT can collect the various TCP network parameters on the hosts and identify their configuration problems. If NDT is also available at the remote site, far-end problems can sometimes be identified quickly. If not, local NDT diagnosis can still help narrow down the problem scope. We need to make sure that TCP features for high performance transfer, such as window-scaling, BDP send/recv socket buffer size, time-stamp, and SACK, are enabled. Especially in long distance and high bandwidth networks, we must make sure that the OS supports a congestion control algorithm suitable for high-speed TCP, and that it is correctly configured. NIC features like interrupt coalescing, TCP offloading, and jumbo frames, should also be enabled. And the NIC driver parameters such as layer 2 send/recv queue sizes must be correctly configured. Interested readers may refer to [11] for details.

If any system settings are changed, we repeat the performance tests to see if there is any performance improvement. If not, we proceed to Step 4.

Step 4: Network path performance analysis

TCP performs poorly in networks with packet losses or severe packet reordering. Long round-trip times are well known to exacerbate such problems. With one-way metrics for these path characteristics, it is much easier to isolate the network problems in the forward path or the reverse path. In [12], V. Paxson has pointed out that "conducting an Internet measurement study in a sound fashion can be more difficult than it might first appear." We need to follow the strategies in [12] for sound Internet measurement.

In this step, we first need to identify the end-to-end paths with the tools such as traceroute, in both directions. We examine the paths, and check whether they are stable. As discussed in [13], routing changes can cause severe packet drops. Also, check if the paths traverse any "known" congested networks or hot spots. Some networks, like Internet2, publish their network status [14], or provide network monitoring services like MRTG, Nagios, perfSONAR or MonALISA to monitor the live network conditions. Usually, the provided network performance data of interest are 1-minite or 5-minute averages (or even longer). These averages may hide the instantaneous network status [15]. In that case, cumulative network performance data (total packets dropped, for example) might also need to be referenced.

Then, we check whether the one-way path is lossy ($\geq$ 0.5% considered very lossy). Since network congestion usually causes packet queuing delay in routers and leads to large one-way delay variance, we could examine the one-way delay variance to determine the possible network congestions. If the one-way delay variance is large, investigation of possible congestion along the path with available tools and services (MRTG, Nagios, perfSONAR, MonALISA, and pathneck) is indicated. If the path is congested, we could inform the cognizant NOC of the congestion bottlenecks along the path, and consider alternate less congested paths if available, or consider upgrading the network capacity. If delay variance is not large and packet drop is severe, we need to investigate for network infrastructure failures such as dirty fibers and malfunctioning line cards. This sort of system debugging might require coordination among different administrative domains, and the related system performance counters along the paths might need to be examined one by one.

We also need to examine if the packet reordering is severe ($\geq$ 3 is considered severe). If it is, we need to increase the "TCP reordering threshold" parameter in the protocol stack.

Rerun performance tests if significant congestion or system packet loss is corrected.

Step 5: Evaluate packet trace pattern

Due to the limitations of current measurement methods [12], sometimes the network path characteristics might not be measured accurately. For example, OWAMP might miss the detection of the path packet drops, or packet reordering due to the coarse measurement resolution. Since any performance bottleneck will manifest itself in the recorded packet traces, we resort to the packet traces analysis as the final tool, and it also helps to debug the network application design.

We record the application packet traces with TCPDUMP, and then run TCPTrace to analyze and generate the graphic outputs. TCPTrace will generate summary TCP statistics such as RTT, retransmission count, maximum advertised windows, and more. Those statistics help us to identify the performance bottleneck. Furthermore, xplot can be used to examine the recorded traces visually. The most valuable graphic output generated by TCPTrace is the Time Sequence Diagram, in which we can see window size changes, packet loss and retransmission, and other events.

If many packets are retransmitted or RTO happens frequently, it means that the packet drop rate is high. If RTT variance is also high, there is high probability that somewhere along the path there is high congestion. If the packet drop rate is high, and the RTT variance is low, network faults are more likely.

3. A Case Study

We apply the network performance analysis methodology to optimize the data transmission performance between Fermilab (FNAL) and the Rio de Janeiro State University (UERJ). In the computing model of the upcoming CMS experiment at the Large Hadron Collider nearing completion at CERN, Fermilab is the Tier-1 center for data distribution and analysis in the western hemisphere; and UERJ is a Tier-2 regional center in Brazil. We ran iperf data transmission experiments to verify the effectiveness of our proposed methodology and verified results with the experiment's own data handing software. During the experiments, data are transmitted in one direction between two computer systems from FNAL to UERJ. Each experiment lasted for 100 seconds. The data transmission results before and after the optimizations are compared.

Some of the sender and receiver's characteristics are shown in Table 2. At their full transmission capacities, both systems can saturate the Gigabit Ethernets. Please note that we show here only a subset of all the information called for in Table 1.

	FNAL Sender	**UERJ Receiver**
CPU	Two Intel Xeon CPUs (3.0 GHz)	Two Intel Xeon CPUs (3.0 GHz)
Memory	4G	4G
NIC	Intel 82563EB, 1Gbps	Intel 82563EB, 1Gbps
OS	Linux 2.6.22, 32bit	Linux 2.6.9, 32bit

Table 2 Sender & Receiver Characteristics

Following the methodology, the first problem we found is that the sizes of TCP send buffer and receive buffer in the receiver were configured too small. The maximum send/receive buffer size was 131071 bytes. The round trip time between FNAL and UERJ is around 250ms. With such configuration and according to (1), the maximum throughput that could be achieved would be 4.2 Mbps (131071*8/0.25). The iperf data was consistent with this estimate. We varied the congestion control algorithms in the sender, and the best throughput we could achieve was 3.9 Mbps (with high performance TCP variants). Pathneck showed the available bandwidth at the time from FNAL to UERJ was around 300Mbps.

We increased the maximum send/receive buffer size to 6000000 bytes in the receiver, and reran the experiments. We varied the congestion control algorithms with results shown in Table 3.

Experiment No.	**RENO/NewRENO**	**CUBIC**	**BIC**	**SCALABLE**
1	22.6 Mbps	94.7 Mbps	98.1 Mbps	116 Mbps
2	22.6 Mbps	110 Mbps	101 Mbps	98.2 Mbps
3	22.7 Mbps	110 Mbps	85.9 Mbps	70.6 Mbps

Table 3 Iperf experiments with various congestion control algorithms, large receiver buffer size

The experiment results clearly demonstrate that (1) with buffer tuning in the receiver, the performance is greatly improved; (2) high performance TCP variants like CUBIC and BIC perform much better than Reno/NewReno. We suggest high performance TCP variants be used in the sender.

Following the methodology, we perform the network path analysis. For simplicity, the path from FNAL to UERJ is called the forward path, and the path in the other direction is called the reverse path. The round trip time (RTT) between FNAL and UERJ is around 250 ms. Traceroute shows that the forward and reverse paths are not symmetric. The forward path has 14 hops, whereas the reverse path has 12 hops. This conclusion is further verified by the OWAMP tests. Table 4 gives two OWAMP tests of the paths. In our OWMAP tests, we try to mimic the behaviors of TCP burstiness with packet trains. A packet train consists of 10 packets. Each packet within a train is sent out back to back. The intervals between consecutive trains are exponentially distributed with an average of 1.0 second.

Test 1	./owping -L 20 –c 10000 -i 1.0e,0f,0f,0f,0f,0f,0f,0f,0f,0f was.fnal.gov --- owping statistics from [200.143.197.197]:35820 to [was.fnal.gov]:32778 --- 10000 sent, 10 lost (0.100%), 0 duplicates one-way delay min/median/max = 93/152/197 ms, (err=29.5 ms) one-way jitter = 30.8 ms (P95-P50) Hops = 12 (consistently) 1-reordering = 0.010011% ; 2-reordering = 0.010012% ; 3-reordering = 0.010013% 4-reordering = 0.010014% ; 5-reordering = 0.010015% ; 6-reordering = 0.010016% no 7-reordering --- owping statistics from [was.fnal.gov]:32779 to [200.143.197.197]:35821 --- 10000 sent, 20 lost (0.200%), 0 duplicates one-way delay min/median/max = 42.2/80.6/141 ms, (err=29.5 ms) one-way jitter = 55.7 ms (P95-P50) Hops takes 2 values; Min Hops = 14, Max Hops = 18 no reordering
Test 2	[root@prod-frontend owping]# ./owping -L 30 -c 10000 -i 1.0e,0f,0f,0f,0f,0f,0f,0 f,0f,0f was.fnal.gov Approximately 1031.8 seconds until results available --- owping statistics from [200.143.197.197]:36146 to [was.fnal.gov]:32769 --- 10000 sent, 0 lost (0.000%), 0 duplicates one-way delay min/median/max = 76.8/91.8/117 ms, (err=27.6 ms) one-way jitter = 21.1 ms (P95-P50) Hops = 12 (consistently) no reordering --- owping statistics from [was.fnal.gov]:32770 to [200.143.197.197]:36147 --- 10000 sent, 0 lost (0.000%), 0 duplicates one-way delay min/median/max = 116/141/161 ms, (err=27.6 ms) one-way jitter = 12.5 ms (P95-P50) Hops = 14 (consistently) no reordering

Table 4 Path Analysis Experiment Results

Test one was performed without synchronized clocks on the OWAMP client and server. Nevertheless, test one still gives interesting measurements for the paths. From test one, it can be seen

that the packet drops & reordering conditions for both the forward and reverse paths are not severe. Since the difference between the one-way delay min and max is so large, the packet drops might be caused by network congestions. The pathneck results also report large fluctuations of available bandwidth along the paths, which confirms our conclusions from another perspective. No evidence points to packet drops due to network infrastructure failures. We also tried to examine the network status along the paths but were frustrated by the large number of administrative domains involved, some of which do not publish their network status information. The packet reordering of the reverse path in Test one suggests the TCP packet reordering threshold in the sender should be raised. Another interesting result about Test one is that it detected route changes in the forward path. Normally, the forward path has 14 hops, but it was observed as high as 18 hops. As discussed previously, route changes might cause transient loss of connectivity and lead to packet drops. We notified NOCs of this situation for possible solutions.

The time clocks for Test two are synchronized. The one-way delay for the forward path and the reverse path are reliable. It can be seen from Test two that the one way delay of the forward path is almost 50ms larger than that of the reverse path. Since the forward and reverse paths are not symmetric, and also there is such a large difference of one-way delay between them, we might have the potential to shorten the forward path to further optimize the performance.

Upon the conclusion of this diagnosis and remediation, the physics data transfer application, which was built upon SRM and GridFTP, showed a fivefold improvement in throughput from FNAL to UERJ.

4. Conclusion

Factors that affect overall network/application performance exist on the hosts themselves (application software, operating system, hardware), in the local area networks that support the end systems, and within the wide area networks. Since the computers and network systems are globally distributed, it can be very difficult to locate and identify the causes of poor application performance. In this paper, we first systematically analyze factors that degrade performance. Then we present an end-to-end network/application performance troubleshooting methodology developed and used at Fermilab. The core of our approach is to narrow down the problem scope with a divide and conquer strategy. The overall complex problem is split into two distinct sub-problems: host diagnosis and tuning, and network path analysis. After satisfactorily evaluating and, if necessary, resolving each sub-problem, we conduct end-to-end performance analysis and diagnosis. The long term objective of the effort is to enable end users to conduct much of the troubleshooting themselves, before (or instead of) calling upon network and end system "wizards," who are always in short supply.

References

[1] http://e2epi.internet2.edu/ndt/
[2] http://e2epi.internet2.edu/owamp/
[3] N. Hu et al., "Locating Internet Bottlenecks: Algorithms, Measurements, and Implications," Proceedings of ACM SIGCOMM 2004, Portland, Oregon, USA, Pages: 41 – 54.
[4] http://dast.nlanr.net/Projects/Iperf/
[5] http://www.mrtg.com/
[6] http://www.nagios.org/
[7] http://www.perfsonar.net/
[8] http://monalisa.cacr.caltech.edu/monalisa.htm
[9] http://www.tcpdump.org/
[10] http://jarok.cs.ohiou.edu/software/tcptrace/
[11] http://www.psc.edu/networking/projects/tcptune/

[12] V. Paxson, “Strategies for sound internet measurement,” Proceedings of ACM Internet Measurement Conference, 2004.
[13] F. Wang et al., “Measurement Study on the Impact of Routing Events on End-to-End Internet Path Performance,” Proceedings of ACM SIGCOMM, September, 2006.
[14] http://www.net.internet2.edu
[15] K. Papagiannaki et al., “network performance monitoring at small time scales,” Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement, Miami Beach, FL, USA, 2003.