VM Clock Synchronization Measurements

Jagmohan Chauhan, Dwight Makaroff and Anthony Arkles
Dept. of Computer Science
University of Saskatchewan
Saskatoon, SK, CANADA S7N 3C9
Email: {jac735, makaroff, aja042} @cs.usask.ca

***Abstract*—A modern twist to clock synchronization is the push towards server virtualization and cloud computing. Distributed applications now often run on geographically-distributed, virtualized servers that are isolated from the underlying hardware. Despite the fact that most applications run well in an abstracted environment without direct hardware access, tight real-time synchronized clocks are still necessary.**

We performed a set of experiments to quantify the additional clock error when running the NTP daemon inside a Virtual Machine. We found that the delay asymmetry is relatively small when running in a non-virtualized environment, but quite significant in a virtualized environment.

I. Introduction

Clock synchronization deals with the observation that internal clocks of several computers differ (clock offset) and that events from distributed and network applications require precise ordering for correct functionality and/or performance. Even when initially set accurately, real clocks will differ after some amount of time due to clock drift (or *skew*), caused by clocks counting time at slightly different rates.

Some types of distributed applications require tight synchronization between the components. In particular, networked games [1], high speed financial trading [2] and factory automation networks [3] are but three application areas where tight synchronization bounds are needed.

Synchronizing a client to a network server consists of several packet exchanges. The travelling time (delay) is estimated to be half of "the total delay minus remote processing time", assuming symmetrical delays. The more symmetric the round-trip time, the more accurate the estimate of the current time will be. Synchronization is challenging in virtualized environments, due to the extra software processing required between the probe packet timestamp and the actual time the packet was transmitted/received at the hardware level, leading to less accurate timestamps and poorer synchronization.

II. Background and Motivation

The enemy of precise network time synchronization is non-determinism, or more specifically, delay variability [2] or synchronization error, which has four sources [4]: a) Send Latency which includes kernel protocol processing, operating system delays, and the time required to transfer the message from the host to its network interface, b) Access Latency, the delay incurred waiting for access to the transmit channel, c) Propagation Latency (travel time), the time needed for the message to travel from sender to receiver once it has left the sender, and d) Receive Latency: the time for the receiver's network interface to receive the message from the channel and notify the host of its arrival.

In a cloud-computing environment, applications on several virtual machines may communicate over a LAN or over a WAN. The simplest technique for time synchronization is to do it in the VMs which need it. The implementation of timestamping in the virtual machine entails operating system overhead in passing system calls through to the hardware, context-switching and interference between the VMs [5]. Thus, if a system can implement timestamping as close to the hardware as practical, accuracy can be maximized [3].

III. Implementation and Testbed

We implemented a simple NTP client which obtains kernel level timestamps, measuring the send and the receive latency up to the point of the last possible kernel operation. The Linux kernel (2.6.27) was modified at the device driver level. The *ioctl* command `SIOCGSTAMPNS` informs the kernel that timestamping is required. By comparing the kernel timestamp and an application-generated timestamp, we measured the send latency. Likewise, we measured the receive latency by comparing the *netif_rx* timestamp with an application-level timestamp immediately after the *recvfrom()* call. The System under test used in the experiments was 2.4 Ghz Pentium quad core with 2GB of RAM. Virtual Box 2.1.4 is used to create and manage instances of VM on the host system. Benchmark software Bonnie++, Stress and iperf was installed.

IV. Experiments and Results

During each test, NTP packets were sent at regular intervals and corresponding send and receive latencies were recorded. The CPU and memory tests were done using the Stress benchmarking tool[1]. For the I/O operations stress tests, we use Bonnie++ benchmark[2]. For network stress testing we used *iperf*,[3] to create TCP and UDP data streams and measure the throughput of the network.

Four different kind of scenarios were considered: a) Measurement for NTP done at Host Level (NTP-host-base), b) Measurement for NTP done at Host Level running load on one Virtual Machine (NTP-host1vm), c) Measurement for NTP done at Host Level but running load on two Virtual Machines

[1] http://weather.ou.edu/āpwprojects/stress/
[2] http://www.coker.com.au/bonnie++/
[3] http://sourceforge.net/projects/iperf/

(NTP-host2vm), and d) Measurement for NTP done at VM Level (NTP-VM).

CPU load tests: In each test, we specified a certain number of threads for each scenario: 250, 500, 750 and 1000. Figure 1 shows the difference of the average between the receive and send latencies. The best results are observed in NTP-host-base with the worst results in NTP-VM, where the average latencies can go to around 2000 μs. For all scenarios except

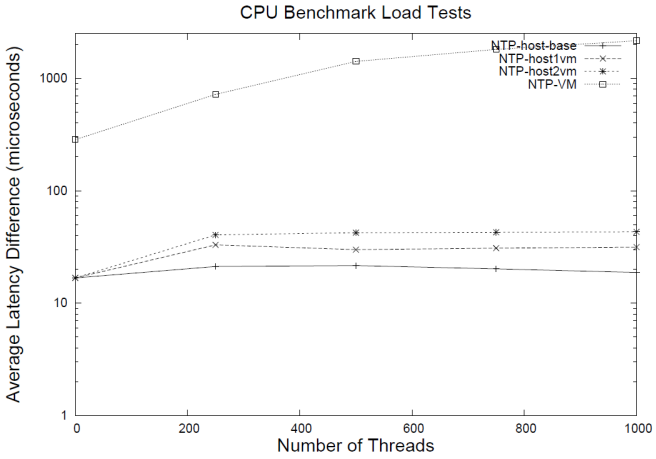

Fig. 1. Latencies for CPU Stress in system

NTP-VM, adding more threads does not affect the Send and Receive latencies much. In NTP-VM, the variability was also substantially higher than in all other scenarios. We observed a number of extreme outlier values. The graph shows the average with the extreme outliers removed. Further examination of the data reveals 2 modes in the latency difference distribution. The first mode is between 600 and 1000 μs, and the second mode is between 4000 and 4500 μs, depending on the load.

Memory load tests: We specified different number of threads for the memory stress tests: 1, 3 and 5. Each thread spins on malloc/free and mallocs/frees 256 MB of memory per operation, which is the default. The significant observation here is that the results get better with load in the VMs and measurements at the host (NTP-host1vm and NTP-host2vm) than NTP-host-base, which is opposite to CPU stress testing.

Fileystem I/O load: We used file sizes of 4, 8 and 12 GB on the host and 1, 2 and 3 GB on the guest for these Bonnie++ tests. The file sizes were chosen in a way that they should be greater than the memory of the system, which was 2 GB for Host and 512 MB for the VM. We observed similar trends as in previous results, which are shown in Table I.

Scenario	Average Latency Difference (μs)
NTP-host-base (4/8/12 GB)	26/25/26
NTP-host-1vm (1/2/3 GB)	34.5/35/37
NTP-host-2vm (1/2/3 GB)	37/40/42
NTP-VM (1/2/3 GB)	425/510/567

TABLE I
LATENCIES FOR BONNIE++ STRESS IN SYSTEM

Network Stress Test: Send latencies are always less than 29 μs. The worst average receive latencies are achieved with NTP-VM where it reaches 1445 μs for UDP and 2002 μs for TCP traffic load. Running heavy traffic load on VMs does not adversely affect the NTP calculations on the host and is, in fact, better than NTP-host-base.

We also performed tests to see if the NTP send and receive latencies are affected in case we are are running NTP client on one VM and the other VM is heavily loaded. We found that there is substantial interference between the VMs and guest-host synchronization is not precise.

V. RELATED WORK

In Local Area Networks, the IEEE 1588 Precision Time Protocol (PTP) standard has been studied in the context of sensor and control networks that support real-time applications [6]. Similar work in industrial control applications that communicate wirelessly indicates that the most accurate synchronization can be achieved by timestamping at the lowest level in the stack [3]. This work measures consequences of path delay instability caused by software processing. NTP has been shown by Broomhead *et al.* to be inadequate when conditions are not ideal [2]. In particular, they show 3 scenarios in which Xen virtualization has challenges. The *RADclock* solution provided resolves these virtualization issues under paravirtualization.

VI. CONCLUSIONS AND FUTURE WORK

Clock synchronization software (particularly NTP) requires network delays (send and receive) to be symmetric; any asymmetry ends up causing errors in the clock synchronization. On a non-virtualized system, this is a reasonable assumption; our host Linux system showed a low asymmetry between 16 and 82 μs. On a virtualized system, though, this assumption breaks down. Because of this dramatic shift in performance, we do not recommend running an NTP client inside a virtualized system when accurate clock synchronization is required. Alternatively, clock synchronization should happen on the host, with a different host-to-VM synchronization mechanism controlling the clocks inside the VMs. For our future work, we are going to create an effective solution for guest-host time synchronization in the virtualized environment.

REFERENCES

[1] M. Roccetti, S. Ferretti, and C. Palazzi, "The brave new world of multiplayer online games: Synchronization issues with smart solutions," *IEEE ISORC*, 2008.
[2] T. Broomhead, L. Cremean, J. Ridoux, and D. Veitch, "Virtualize Everything But Time," in *OSDI'10*, Vancouver, BC, Canada, Oct. 2010, pp. 1–6.
[3] A. Mahmood and G. Gaderer, "Timestamping for IEEE1588 based Clock Synchronization in Wireless LAN," in *ISPCS*, Brescia, Italy, Oct. 2009, pp. 1–6.
[4] H. Kopetz and W. Schwabi, "Global Time in Distributed Real-time Systems," Technischen Universitat Wien, Tech. Rep. 15/89, 1989.
[5] J. Schad, J. Dittrich, and J.-A. Quiané-Ruiz, "Runtime measurements in the cloud: observing, analyzing, and reducing variance," *VLDB*, vol. 3, pp. 460–471, September 2010.
[6] D. M. Anand, J. Fletcher, Y. Li-Baboud, and J. Moyne, "A Practical Implementation of Distributed System Control over an Asynchronous Ethernet Network Using Time Stamped Data," in *2010 IEEE CASE*, Toronto, Canada, Aug. 2010, pp. 515–520.