Queuing Network Performance Models for Virtual Graphic Display

Zhang Ying, Cai Suo
Huazhong University of Science and Technology
Wuhan, 430074
Hubei, China

Zhou Bin, Zhang ShuDao
Wuhan Technology and Business University
Wuhan, 430065, Hubei, China
binzhou@mail.scuec.edu.cn

ABSTRACT: *The performance of graphics display of virtual machine improves with the strong support from development of virtualization technology and computer hardware. However, in terms of graphical display mechanism there is a difference between the virtual environment and the traditional environment. So how to measure the performance of graphics display in virtualization environment effectively is especially attended.This paper analyzed the virtual display system in theory by models; compares theoretical data and measured data accessed by the virtual machine system performance evaluation; and finally verified the reasonableness of the model.*

Keywords: Queuing network, Virtualization, Performance evaluation

Received: 17 August 2016, Revised 19 September 2016, Accepted 26 September 2016

1. Introduction

With the development of virtualization technology for more than forty years [1-3], Virtual Machine (VM) with the merits, such as being flexible, secure, reliable, and easy to manage and configure, reduces the hardware cost greatly. However, problems show up apart from the benefits, such as the unexpected complexity of virtual computing systems, and poor performance of virtual computing systems. That is why the important issue of how to measure the performance of the virtual computing system effectively becomes acute.

In recent years, the evaluation of the technology based on the virtualization has developed rapidly, and some representative

systems on virtual performance evaluation theory, methods of performance evaluation and the corresponding performance evaluation tools have been widely put in practice, such as XenMon[3,4],vConsolidate[5,6],VMmark[7,8] ,etc.

The paper analyzes the principle of VM graphic display, describes its underlying realization, builds Queuing Network(QN) models for a single domain U and multi-domain U respectively to predict the virtual graphic display performance by selecting the relevant scenarios, getting the experiment data , and validating the QN models.

The main contributions of this paper are as follows.

Firstly, it describes the Xen virtual machine environment display system of the underlying implementation details, analyzes the virtual graphic display principle in Xen, creates a virtual machine display performance evaluation system, which is tracing and profiling in different scenes through three working modules.

Secondly, it builds QN models for single domain u and multi-domain u respectively, and makes a theoretical performance of preliminary analysis and prediction.

Thirdly it obtains performance-related test results through the virtual machine display performance evaluation system, and validates the correct model of rationality QN.

The rest of the paper is organized as follows. Section 2 is about related works. Section 3 introduces design of the evaluation system of VM display performance including the framework of whole system as well as the various functional modules. Section 4 builds queuing network models for single domain U and multi-domain U respectively; and section 5 validates the two queuing models respectively, at last in section 6 a conclusion is drawn.

2. Related works

In recent years, some representative evaluation theories, methods of performance evaluation and the corresponding performance evaluation tools such as VMmark, 3DMark[9], etc. have turned up.

However, there is a shortage of the appropriate tools for evaluation of graphic display performance in virtual environment. The research of virtual graphical display is still in its infant. M. Dowty et al. [10] carried out the classification. One class used split front end and back end which the VM in domain U could indirectly use physics video card. VMGL [11] was the realization of this way. The other was that the Domain U operated hardware directly, which required IOMMU [12] support.

During the last 40 years, research showed that queue network models served as a fundamental tool to model computing systems [13,14]. In fact, queue network models have been successfully applied to areas such as capacity planning and performance analysis [15] etc. Modeling for dependability and performance evaluation has proven to be a useful and versatile approach in all the phases of the system life cycle.

3. Design the evaluation system of VM display performance

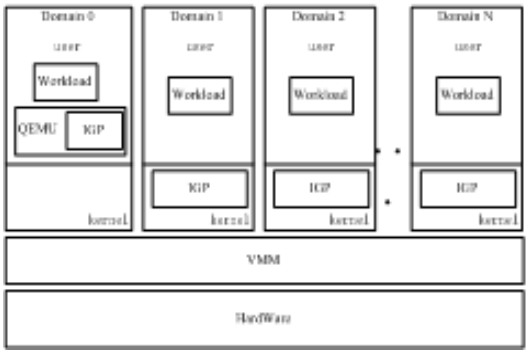

Figure 1. The evaluation system of VM display

In this section, the evaluation system of VM display performance is introduced, including the overall framework of the system, the various components of the functional module, as well as a specific algorithm.

3.1 Function module

The system mainly includes three parts: workload, information gathering procedures and the results of processing and analysing program, as shown in Figure 1.

3.1.1 Work Load (WL) module

In the system, the selected workloads take into account the graphical user interface of VM in different applications. Three representative applications are selected, such as text output application, video application, as well as an application to operate virtual frame buffer directly.

3.1.2 Information Gathering Procedures (IGP) module

When the workload running in Domain U VM makes the front-end drive located in the Domain U to detect the update operation, the IGP execute related codes to record the performance information in the log file.

3.1.3 Results of Processing and Analysing Program (RPAP) module

The results of the processing and analysing program process the records after the related information in the VM recorded into the log files. The mouse or keyboard operations will be generated continuously when users using the graphical interface. Furthermore, the output will also have some changes, even if the users do not have any operations. For example, the screen will be refreshed continuously when a movie is played. The new events will be trigged in the display system constantly. And the the corresponding recorders will be produced continuously in the log system.

The system repeats the measurement and recording many times to ensure the accuracy of the results. Finally, the RPAP module deals with the records and gets its average.

3.2 Virtual frame buffer test procedure

In order to detect the limited number of events that generats by the front-end driver corresponding to the virtual frame buffer in the unit time, the virtual frame buffer should be operated directly to accelerate the events generated rate. For the update times are not enough in the unit time, the common application program can not meet the requirement. This paper presents an easy and practical virtual frame buffer test procedure. The realization is writing the the display data through the direct operation of the virtual frame buffer equipment to generate the mandatory refresh operation. And then the corresponding front-end will send out data requests and event notifications.

The specific algorithm is as follows:

Step 1: Running compiled instruction RDTSC, recording the current time point;

Step 2: Opening the virtual frame buffer device / dev/fb0;

Step 3: Using memset instruction to set all memory regions to 0 in virtual frame equipment;

Step 4: Turning off the virtual frame buffer device;

Step 5: Using the RDTSC instruction to record the current time point;

Step 6: Re-opening the virtual frame buffer device to set all the regions to 1 in the virtual frame buffer memory, forcing to make a refresh operation , and then again to turn off the virtual frame buffer device;

Step 7: Jumping to step 1, repeat the above operations.

4. Latency Performance Modeling for Display

A display latency performance model of VM display was abstracted in Para-Virtualization (PV)[16] environment. The output ring buffer could be seen as a queue, and each display request of the VM should be acted as a customer. So when a VM was booted, the whole display system was a queue of a single queuing system.

When there were multiple VMs, each VM could be a customer. As the multiple VMs were scheduled by the VMM (Virtual Machine Monitor), the entire virtual machine system was equivalent to a limited source queuing system.

The latency performance model discussed in this paper, whose requests arrive at a variety of rules, and the request data arrived at the rule of identical interval time arrival. Morever, it could also be arrived at the law of random. It was closely related to the applications. The rule of the events arrival corresponding to the applications was appeared at the identical interval time.

As the basic service rule was FCFS, regardless of the scheduling, the output ring buffer was equivalent to a FCFS queue to a single VM. Moreover, to multiple VMs running in a Xen system, the default scheduling algorithm was credit. If all the VMs were running similar services, the principle of FCFS should be approximately applied.

4.1 QN Model for single VM

In a video or text output, or a process of direct operation for virtual frame buffer, the input stream was the identical interval. Assuming the service time interval submits to negative exponential distribution which was mutual independence, the parameters was μ. This queuing model had one server and one queue, and D/M/1 queuing theory model was used to describe the virtual display request queue.

(1) the probability $P(n = k)$ which the number of requested to be processed in the system was k:

$$P(n=k) = \left(\frac{\lambda}{\mu}\right)^k \left(1 - \frac{\lambda}{\mu}\right) \tag{1}$$

λ was the arrive requests number in unit time, and μ was the processed request number in unit time.

(2) L: the average number of requests in the system (including the request being processed)

$$L = \sum_{n=0}^{\infty} nP_n = \sum_{n=0}^{\infty} n\left(\frac{\lambda}{\mu}\right)^n \left(1 - \frac{\lambda}{\mu}\right) = \frac{\frac{\lambda}{\mu}}{1 - \frac{\lambda}{\mu}} = \frac{\lambda}{\mu - \lambda} \tag{2}$$

(3) W: the average processing time for the display request in the queue

$$W = \frac{L}{\lambda} = \frac{1}{\mu - \lambda} \tag{3}$$

(4) L_q: the average number of requests in the queue(not including the request being processed)

$$L_q = \sum_{n=2}^{\infty} (n-1)P_n = \frac{\left(\frac{\lambda}{\mu}\right)^2}{1 - \frac{\lambda}{\mu}} = \frac{\lambda^2}{\mu(\mu - \lambda)} \tag{4}$$

(5) W_q:the average waiting time for each request

$$W_q = \frac{L_q}{\lambda} = \frac{\frac{\lambda}{\mu}}{\mu\left(1 - \frac{\lambda}{\mu}\right)} = \frac{\lambda}{\mu(\mu - \lambda)} \tag{5}$$

Since the maximum queue length was 51, there was a state of 52, each of which corresponding to the number of representatives of the state of the queue having not been dealt yet with the number of data request. λ is the arrive requests number in unit time, while μ was the processed request number. As the rate of the transition of one state to the next one was λ, the rate of one state transiting to the last one was μ .

The state transition diagram were shown in Figure 2.

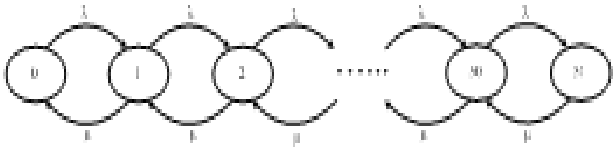

Figure 2. The state transition diagram for Single VM QN model

In particular, when $\lambda << \mu$, it was very hard to have any transition after state 1 since the rate of the data request by the server was too high. Then the different states after state 1 could be ignored and transition diagram could be greatly simplified.

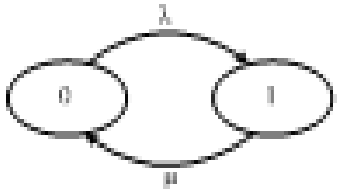

Figure 3. The simplified state transition diagram for Single VM QN model

4.2 QN Model for multiple VMs

There were *m* cores and *n* guest domains in the system. When multiple guest Domains had outputs at the same time, assuming that the arrival interval time for the display requests of each VM in domain U obeyed a negative exponential distribution with parameter λ as well as the server time obeying a negative exponential distribution with parameter μ. After a measurement we found that for each VM in the Domain U, it needed only a few thousand clock cycles for the server to process the requests, that was, a few microseconds. The update request of the VFB contents issued by the VM in the Domain U required the fastest 10ms interval. So $n\lambda << m\mu$ when the multiple Domain Us had output at the same time.

Each VM in the domain U being regarded as a customer, and which was regarded as a customer arrival when the request arrived, and it was used M/M/m/n/n model [16] to describe the display request queue. When the rate of dealing with data requested was far greater than the rate of request arrival, that was, when $\lambda << \mu$, $\frac{\lambda}{\mu} \approx 0$ there were major characteristics following according to M/M/m/n/n model:

(1) The $N(t)$ denoted the number of VMs which had the requests at time t, the $p_j = P\{N(t) = j\}$ denoted the probability of request for j VMs at the time t, and $j = 0,1,\ldots,n$

$$p_j = \begin{cases} \binom{n}{j}\left(\frac{\lambda}{\mu}\right)^j p_0 & j = 0,1,\ldots,m-1, \\ \binom{n}{j}\frac{j!}{m!m^{j-m}}\left(\frac{\lambda}{\mu}\right)^j p_0 & j = m,\ldots,n, \end{cases} \tag{6}$$

Where, $p_0 = \left(\sum_{i=0}^{m-1}\binom{n}{i}\left(\frac{\lambda}{\mu}\right)^i + \sum_{i=m}^{n}\binom{n}{i}\frac{i!}{m!m^{i-m}}\left(\frac{\lambda}{\mu}\right)^i\right)$ the $c\binom{n}{j}$ equalled to taking j combination from n and $\binom{n}{j} = n!/(j!(n-j)!)$, $0 \le j \le n$.

In particular, when the server number was only 1, that was, then

$$p_j = \frac{n!}{(n-j)!}\left(\frac{\lambda}{\mu}\right)^j P_o, \quad j = 1,\ldots,m, \tag{7}$$

And

$$p_o = \left(\sum_{j=0}^{n}\frac{n!}{(n-j)!}\left(\frac{\lambda}{\mu}\right)^j\right)^{-1} \tag{8}$$

(2) The N_1 denoted the number of the VM which had requests, its average was:

$$\overline{N_1} = \sum_{i=0}^{n} ip_i \tag{9}$$

(3) The average number of the VMs which issued the request in per unit time:

$$\overline{\lambda} = \lambda \sum_{i=0}^{n} (n\text{-}i)\, p_i = \lambda\,(n - \overline{N_1}) \tag{10}$$

(4) According to the little's law, the average staying time for the request was:

$$\overline{T} = \frac{\overline{N_1}}{\overline{\lambda}} \tag{11}$$

The state transition diagram was shown in Figure 4.

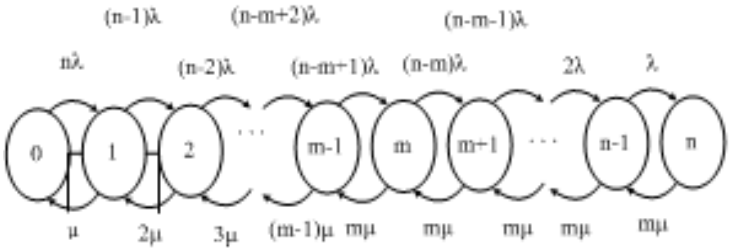

Figure 4. The state transition diagram for multiple VMs model

For display devices, processing speed was fast and the display request arrival rate was relatively slow. When there were requests in x domain U, the state transition rate from current state to next state, that was, the rate of the increasing number of the domain having the data request was $(n - x)\lambda$.

There were m servers, assuming that x domain U issued data requests. When $x<m$, the processing rate of data request was $x\,\mu$; when $x=m$, the largest processing rate of data request was $m\mu$; when $x>m$, the rate still remained at $m\mu$.

5. Experimental Evaluation

5.1The experimental setting

The configuration of test server was dual Intel Xeon quad-core CPU, and the frequency for each core was 1.60GHz. Furthermore the capacity of cache for each core was 4MB, while physical memory 4GB, and capacity of hard drive 80GB, and network bandwidth 100Mbps. The model in this paper was based mainly on the PV in Xen.

To test the model, a sequence of benchmarks on a VM and multiple VMs were run respectively. These benchmarks included the video output benchmark, the text output benchmark, as well as a VFB directly to write benchmark. For each benchmark, the test metrics included the VFB refresh rate; the data request frequency issued by the VM in domain U to Domain 0, and the frequency of detecting and processing the data requests by a VM in Domain 0, etc.

The front-end rate denoted the data request refresh rate by the front driver of VM in domain U to domain 0.

The back-end rate denoted the rate that the VM in the Domain 0 receiving and processing the display events issued by the VM in Domain u.

In the following figures, the horizontal axis was on behalf of VM's built-in refresh rate *xenfb_fps* in Domain U, nevertheless the longitudinal axis was on behalf of the average refresh interval of front-end and back-end on the fixed *xenfb_fps* respectively.

Multiple times were spent to get the average measurement values in the experiment, so long as to further reduce errors, jitter, and random factors.

5.2 Experiment in a single Domain U

Three benchmarks were run on a VM in Domain U, and a key parameter impacts the virtual display performance. The parameter was the VM's built-in refresh rate *xenfb_fps.*

5.2.1 Video benchmark

From the Figure 5, in the video benchmark, when the VM built-in refresh rate *xenfb_fps* is less than 60 times per second in domain U, with the *xenfb_fps* decreasing, the average of video refresh interval gradually increases.

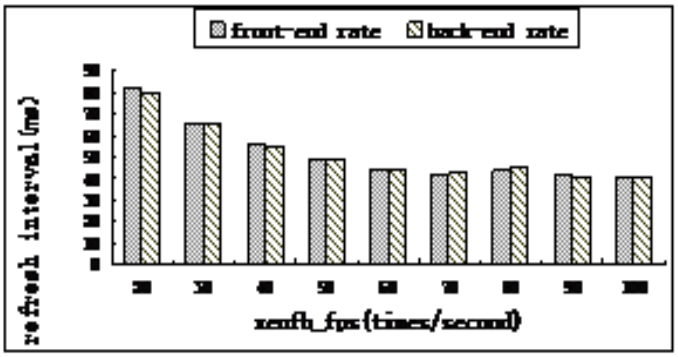

Figure 5. Video applications refresh interval in a single VM

When *xenfb_fps*>60, the average refresh rate of the video is basically stable, however the average front-end and back-end refresh rates are almost 40ms, corresponding to the frame rate being at around 25, just as the video inherent frame rate when it is encoded.

5.2.2 Text output benchmark

The Figure 6 shows a text output refresh interval in a VM benchmark. In this benchmark, the terminal was refreshed constantly, displaying multi-paragraph text circularly, and recording the refresh rate of front-end driver and the refresh rate of back-end driver. Compared to video benchmark, the refresh rate of text is obviously much higher.

When *xenfb_fps*<70, the corresponding average refresh interval of front-end and back-end in Domain U increases with the *xenfb_fps* declines. For the entire virtual display system, the performance bottleneck of the whole system is *xenfb_fps*.

When *xenfb_fps*>70, Corresponding front-end and back-end the average refresh interval remains basically unchanged, the application issues only 50 event notifications, and the average refresh interval is about 20ms. For the entire virtual display system, the performance bottleneck at this time is the text of the application itself.

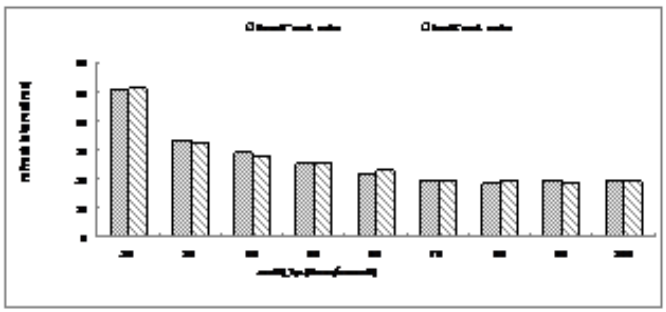

Figure 6. Text output refresh interval in a single VM

5.2.3 VFB directly to operation benchmark

Different with the two previous applications, the direct operation of the VFB device produced by the process of refresh rate is expected to reach thousands of clock cycles. It is impossible to become the performance bottleneck in the entire virtual display system. As can be seen from the figure8 when *xenfb_fps*<100, the rate of event notification issued by a front-end and the average rate of events received and processed by the back-end is decreasing with the *xenfb_fps* reduced.

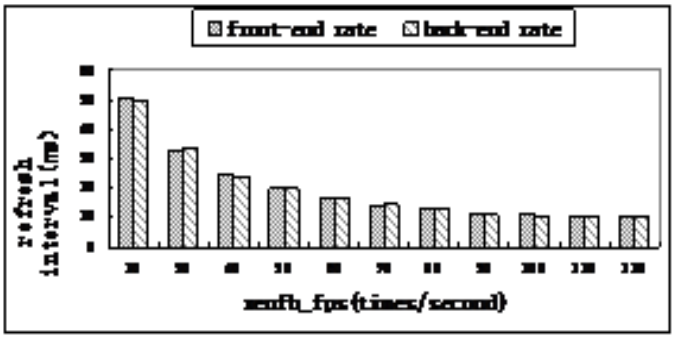

Figure 7. VFB directly to operate refresh interval in a single VM

In Figure 7 When *xenfb_fps*>100, the rate of event notification issued by a front-end and the average rate of events received and processed by the back-end remain unchanged, around 10ms, about one time slice allocated by VM using credit in Xen.

From Figure 8, we can see clearly that for the typical applications of these three, when the VM's built-in refresh rate below a certain value, we are able to increase this parameter to achieve an optimal value, reducing the refresh time interval to increase the VM's virtual display performance. At the same time, in experiment we also can significantly perceive the upgrade effect of the display with the parameter increased. But for the practical application, the optimal value is not the same.

Once this value reaches the optimal value, performance however cannot be obviously improved when the value is increased.

Of course, for different applications, the optimal value is not fixed.

Through the analysis of Figure 8, it is clear that no matter what kind of applications, its built-frequency limit of the more than 100 is meaningless. It also fits the actual scheduling of VM ,which in the Xen PV environment scheduling algorithm used by credit schedule once every 10ms, the frequency was 100 times /sec. Therefore, from this chart, scheduling in the VM to be shared after the time of each chip can only be issued up to one display relevant data request.

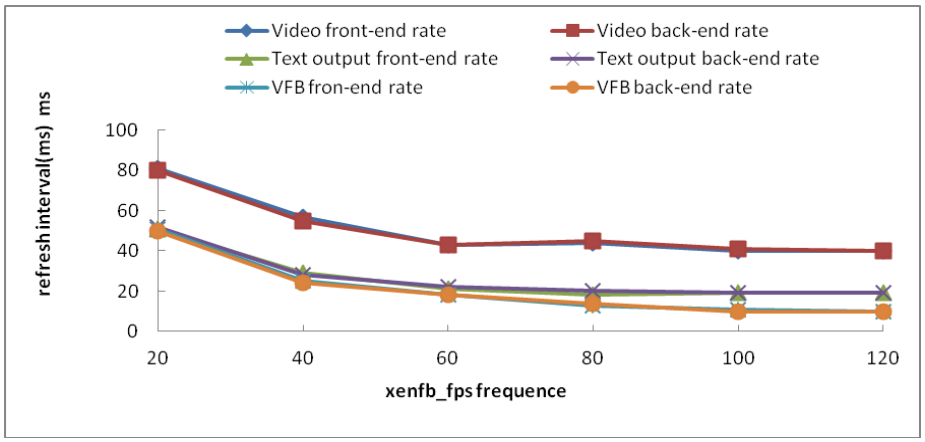

Figure 8. The comparison between the Xenfb_fps and refresh interval in the different benchmark

5.3 Time test in a single domain u

In order to minimize errors and jitter in the experiment, the averaging method is frequently applied by repeating access to data.

	Video	Text	VFB
Processing Time in ring	25.7	25.6	25.5
Processing time in Domain 0	2.5	2.7	2.7
The shortest interval of front-end notification *ms*	10	10	10

Table 1. Processing time in single domain U of the three benchmarks

From the experimental data shown in Table 1, three benchmarks are not much different in the Xen environment. However, the issue of front-end driven requests for the shortest time interval and the first three are not in an order of magnitude, showing the data request arrival rate being much smaller than the queue time and processing time. Between them there is a difference of almost three orders of magnitude, then the data request queue does not exceed the number 1, and queue length in the majority of time is 0.

Requesting time for the data buffer in the ring, as well as the waiting time for data requests in the Domain 0 only takes a few thousand to tens of thousands of CPU clock cycles, for the experiment here it is a few microseconds to several 10 microseconds.

The sum of the first two items in Table 1 is taken as the server time in the QN model for each data request, so we get $\frac{1}{\mu} \approx 28.3\ \mu s$. The time interval of the front end driver sending events to back-end driver is $\frac{1}{\lambda} \approx 10ms$, and from formula (2) we get the data requests average number of L(including the request being processed) in the virtual display system as follows:

$$L = \frac{\lambda}{\mu - \lambda} \approx \frac{1/(10ms)}{1/(28.3\mu s) - 1/(10ms)} \approx 0.0028;$$

With formula (3):

$$W = \frac{L}{\lambda} = \frac{1}{\mu - \lambda} \approx \frac{1}{1/(28.3\ \mu s) - 1/(10\ ms)} \approx 28.38\ \mu s$$

At the same time, in accordance with formula (4) we get the average number of requests in the queue:

$$L_q = \frac{\lambda^2}{\mu(\mu - \lambda)} \approx \frac{10000}{1000000/28.3(1000000/28.3 - 100)} \approx 0.08$$

Basically, these fit real data.

5.4 time test in multiple VMs

The common situation is that more than one VM running at the same time, for the following running multiple VMs.

Similar to single VM, the processing of virtual display data is divided into two parts: the processing time in the ring buffer and in domain 0. As can be seen from Figure 9, typical for different applications, spending time in this part is basically stable and there is no significant relationship with the number of running VMs.

In circumstances of multiple VMs, the processing time for data request in the ring buffer increases as the number of VMs increases. We can see in Figure 10 that as our physical server is two-way quad-core CPU, when the number of VMs reaches 8, the VM does not run very smooth; when the number of VMs reaches 10, the number of VMs being more than the number of physical CPU, the less use of physical CPU resources, the poorerly the VM begins running on. This shows that with the increase

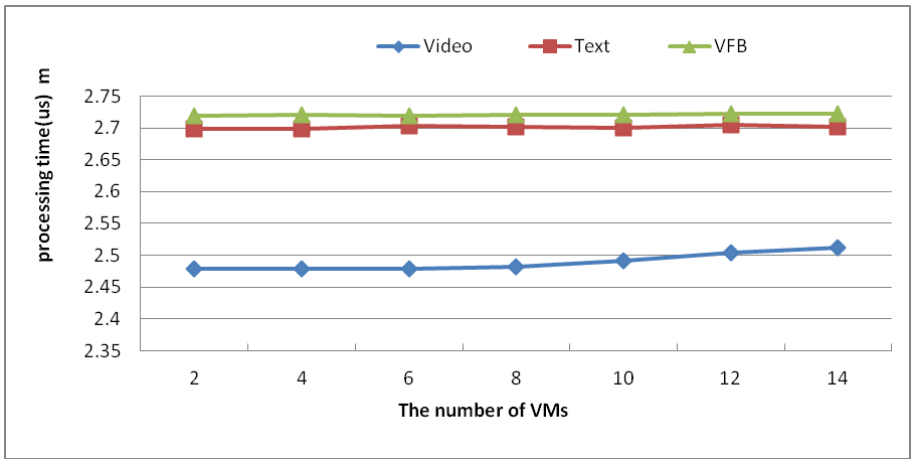

Figure 9. The processing time in domain 0 for multiple VMs

in the number VMs, the probability of scheduling for each VM is decreasing, while the time of the corresponding requesting the ring buffer increases.

In the following, we compared the experimental data with the theoretical data computed through the formula in section 5 (μ =1s/ 28.25us=35400, λ =1s/10ms=100 in the formula);

The W in the Figure (10) denoted the theoretical length of stay time of the display data request in the queue; Vedio_W*,Text_W*and VFB_W* denoted the actual length of stay time that we ran the different benchmarks to access.

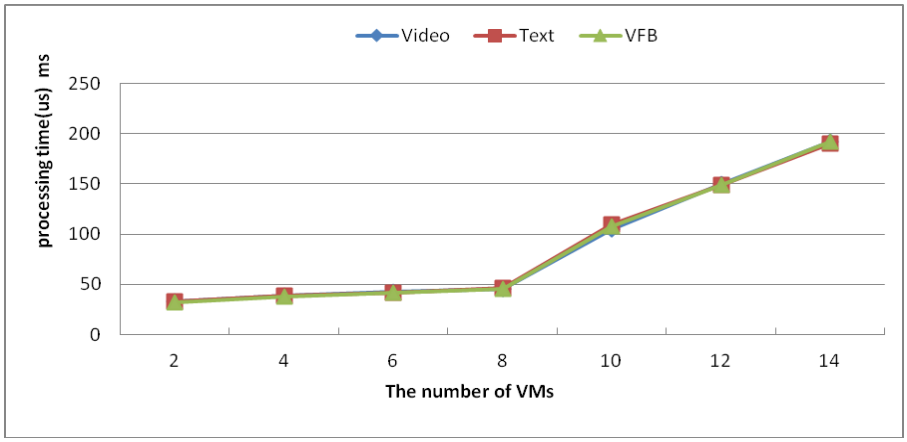

Figure 10. The processing time in the ring buffer for multiple VMs

Figure 11 shows when the number of VM is not more than 8, the experimental data and theoretical data are similar. Some errors turned up as when we using the instrumentation method to get the data, and experimental means impacted the accuracy of the data. When the number of VMs is more than 8, the actual measured data compared with the theoretical data has a large increase. The reasons are as follows:

As our physics machine was only 8 cores, with the increasement of the VM, cpu resources gave rise to competition, thus VM scheduling led to additional performance cost causing absurdities of the output. When there is an increase in the number of VMs, the VMs are scheduled frequently. Then the cache misses inevitably increase, which leads to a certain amount of performance degradation.

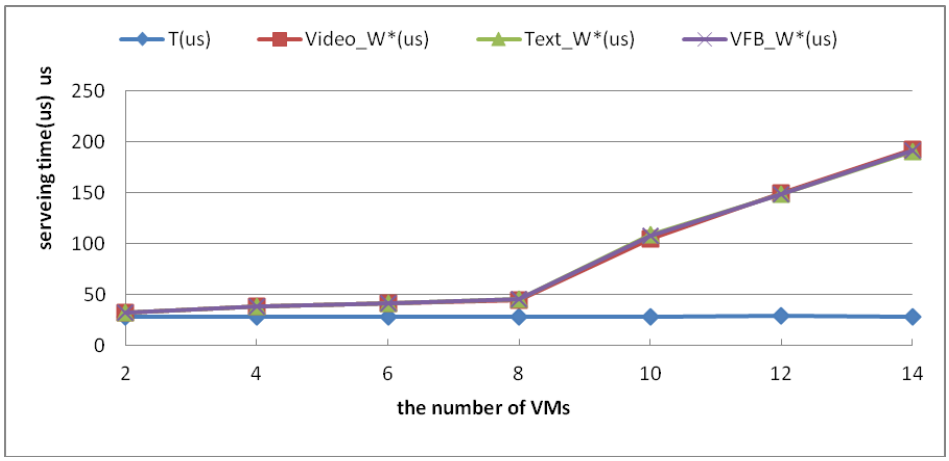

Figure 11. Experimental data compared with the theoretical data in the queue

6. Conclusions

By doing a research on the principle of virtual machine, we designed a performance evaluation system in Xen virtual environment. At the same time, QN models were built for virtual display system at a single VM and multiple VMs in the theory respectively. Specifically, the main research work included the following:

• Described the underlying implementation details of the virtual display system in the Xen virtual, and analyzed the display principle of Xen-based virtual machine;

• Proposed an M/M/m/n/n display latency performance model in Xen environment;

• Verified the reasonableness of the model through a series of experiments, and compared the measured data and the conclusions from the model.

Acknowledgments

This research was supported by Natural Science Foundation of Hubei Province (No. 2013CFB447).

References

[1] Challa, N R. (2012). Hardware Based I/O Virtualization Technologies for Hypervisors, Configurations and Advantages - A Study, *In:* IEEE International Conference on Cloud Computing in Emerging Markets 1-5.

[2] Uhlig, R. (2005). Intel virtualization technology. *Journal of Computer*, 38 (5) 48-56.

[3] Gupta, D., Gardner, R., Cherkasova, L. (2005). XenMon: QoS Monitoring and Performance Profiling Tool. *HPL-2005-187*.

[4] Kim, B K., Yoo, Y J., Yoo, C., Ko, Y. W, (2012). Design and Implementation of Resource Management Tool for Virtual Machine. *Computer and Information Science* CCIS 350 17-24.

[5] Verboven, S., Vanmechelen, K., Broeckhove, J. (2013). Black box scheduling for resource intensive virtual machine workloads with interference models. *Journal of Future Generation Computer Systems*. 29 (8) 871–1884

[6] Consolidate performance and power on Intel-based servers. Available at: *https://www.yumpu.com/en/document/view/24134559/vconsolidate-performance-and-power-on-intel-based-server*.

[7] Banu, M U., Saravanan, K. (2013).Optimizing the Cost for Resource Subscription Policy in IaaS Cloud. *International Journal of Engineering Trends and Technology* 6 (6) 296-301.

[8] Ye K, Che J, Jiang. X H., Chen. J H.(2010). vTestkit: A Performance Benchmarking Framework for Virtualization Environments. *In:* The Proceeding of Fifth Annual ChinaGrid Conference (ChinaGrid), 130-136.

[9] Futuremark Benchmarks and Performance Tests. (2014). Available at: *http://www.futuremark.com/*

[10] Dowty, M., Sugerman, J. (2009). GPU virtualization on VMware's hosted I/O architecture. *Journal of ACM SIGOPS Operating System Review*, 43 (3) 73-82.

[11] VMGL: OpenGL Hardware 3D Acceleration for Virtual Machines. 2014. Available at: *http://sysweb.cs.toronto.edu/vmgl*

[12] Amit N, Yehuda M B, Yassour B A, 2012. IOMMU: Strategies for Mitigating the IOTLB Bottleneck. *Computer Science* 6161 256-74.

[13] Doshi B T, 1986. Queueing systems with vacations - A survey. *Journal of Queueing Systems*, 1 (1) 29-66.

[14] Lazowska, E D., Zahorjan, J., Graham, G. S., Sevcik, K. C (1984). Quantitative system performance: computer system analysis using queueing network models. *Prentice-Hall, Inc.*

[15] Daniel A. Menasce, Almeida, V, 2001. Capacity Planning for Web Services: Metrics, Models, and Methods. *Prentice Hall PTR.*

[16] Zhou, B., Jin, H., Yuan, P .P,. (2010). Vdismodel: A Display Latency Performance Model in Virtualization Environment, *In:* Proceedings of 2010 Sixth International Conference on Semantics, Knowledge and Grids(SKG'10), 90-96.